UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Response-Based and Counterfactual Learning for Sequence-to-Sequence Tasks in NLP

By

Carolin (Haas) Lawrence, M.A.

Supervisor & First Evaluator: Prof. Dr. Stefan Riezler

Second Evaluator: Prof. Dr. Artem Sokolov

Dissertation

26th November 2018

Submitted in Partial Fulfilment of the Requirements
for the Degree of

Doktor der Philosophie (Dr. phil.)

in Computational Linguistics
at the Institute for Computational Linguistics
at the Faculty of Modern Languages
at the Heidelberg University, Heidelberg, Germany, 2018.

Hierbei handelt es sich um eine Heidelberger Dissertation.

We all need people who will give us feedback.
That's how we improve.

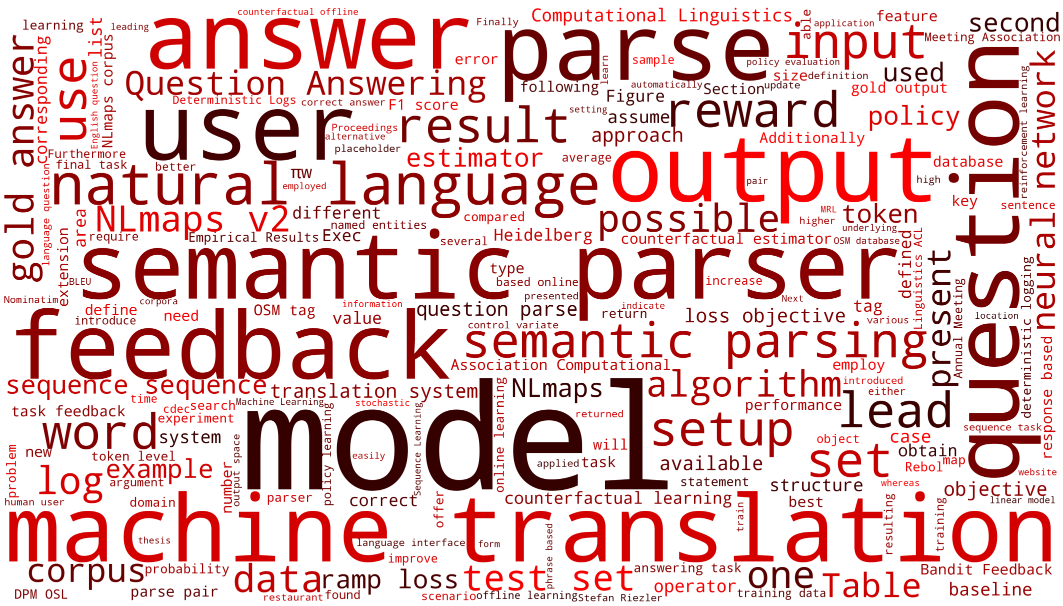
- Bill Gates, TED Talks Education, May 2013

Abstract

Many applications nowadays rely on statistical machine-learnt models, such as a rising number of virtual personal assistants. To train statistical models, typically large amounts of labelled data are required which are expensive and difficult to obtain. In this thesis, we investigate two approaches that alleviate the need for labelled data by leveraging feedback to model outputs instead. Both scenarios are applied to two sequence-to-sequence tasks for Natural Language Processing (NLP): machine translation and semantic parsing for question-answering. Additionally, we define a new question-answering task based on the geographical database OpenStreetMap (OSM) and collect a corpus, NLmaps v2, with 28,609 question-parse pairs. With the corpus, we build semantic parsers for subsequent experiments. Furthermore, we are the first to design a natural language interface to OSM, for which we specifically tailor a parser.

The first approach to learn from feedback given to model outputs, considers a scenario where weak supervision is available by grounding the model in a downstream task for which labelled data has been collected. Feedback obtained from the downstream task is used to improve the model in a response-based on-policy learning setup. We apply this approach to improve a machine translation system, which is grounded in a multilingual semantic parsing task, by employing ramp loss objectives. Next, we improve a neural semantic parser where only gold answers, but not gold parses, are available, by lifting ramp loss objectives to non-linear neural networks. In the second approach to learn from feedback, instead of collecting expensive labelled data, a model is deployed and user-model interactions are recorded in a log. This log is used to improve a model in a counterfactual off-policy learning setup. We first exemplify this approach on a domain adaptation task for machine translation. Here, we show that counterfactual learning can be applied to tasks with large output spaces and, in contrast to prevalent theory, deterministic logs can successfully be used on sequence-to-sequence tasks for NLP. Next, we demonstrate on a semantic parsing task that counterfactual learning can also be applied when the underlying model is a neural network and feedback is collected from human users. Applying both approaches to the same semantic parsing task, allows us to draw a direct comparison between them. Response-based on-policy learning outperforms counterfactual off-policy learning, but requires expensive labelled data for the downstream task, whereas interaction logs for counterfactual learning can be easier to obtain in various scenarios.

Keywords: natural language processing, sequence-to-sequence learning, semantic parsing, question-answering, machine translation, learning from feedback, response-based on-policy learning, grounded learning, counterfactual off-policy learning, off-policy reinforcement learning, bandit learning.



Kurzfassung

Viele Anwendungen basieren heutzutage auf statistischen, maschinell erlernten Modellen, wie z.B. eine steigende Anzahl von virtuellen persönlichen Assistenten. Um statistische Modelle zu trainieren, sind typischerweise große Mengen an parallelen Daten erforderlich, welche teuer und schwer zu beschaffen sind. In dieser Arbeit werden wir zwei Ansätze untersuchen, die den Bedarf an parallelen Daten verringert, indem stattdessen "Feedback" für Modellausgaben verwendet wird. Beide Szenarien werden auf zwei "Sequence-to-Sequence" Aufgaben für "Natural Language Processing (NLP)" angewendet: Maschinelle Übersetzung und semantisches Parsen für die Beantwortung von Fragen. Zusätzlich definieren wir eine neue Aufgabe für die Beantwortung von Fragen auf Basis der geographische Datenbank OpenStreetMap (OSM). Hierfür sammeln wir einen Korpus, NLmaps v2, mit 28.609 Frage-Parse Paaren. Mit dem Korpus bauen wir semantische Parser für spätere Experimente. Darüber hinaus sind wir die Ersten, die eine natürlichsprachliche Schnittstelle zu OSM entwerfen, wofür wir speziell einen Parser anpassen.

Der erste Ansatz, um von "Feedback" für Modellausgaben zu lernen, sieht ein Szenario vor, bei dem ein schwaches Lernsignal vorhanden ist. Das Modell wird in einer nachfolgenden Aufgabe verankert für die parallele Daten vorhanden sind. Das "Feedback", welches die nachfolgenden Aufgabe vergibt, wird zur Verbesserung des Modells in einem "response-based on-policy learning" Setup verwendet. Dieser Ansatz wird zunächst verwendet, um ein maschinelles Übersetzungssystem zu verbessern. Dieses ist in einem multilingualen semantischen Parsen Problem verankert und es werden "ramp loss objectives" zur Verbesserung des Systems verwendet. Als nächstes verbessern wir einen semantischen Parser für den nur Gold-Antworten, aber keine Gold-Parse, vorhanden sind, in dem wir "ramp loss objectives" auf nicht-lineare neuronale Netzwerke anwenden. Im zweiten Ansatz, um aus "Feedback" zu lernen, wird, anstelle der Sammlung teurer paralleler Daten, ein Modell eingesetzt um Nutzer-Modell Interaktionen in einer Logdatei zu sammeln. Diese Log Datei wird verwendet, um ein Modell in einem "counterfactual off-policy learning" Setup zu verbessern. Wir verwenden diesen Ansatz zunächst um ein maschinelles Übersetzungssystem an eine neue Domäne anzupassen. Hier zeigen wir, dass dieser Ansatz auf Aufgaben mit großen Ausgabemengen angewendet werden kann und, im Gegensatz zu gängiger Theorie, können deterministische Logdateien erfolgreich bei "Sequence-to-Sequence" Aufgaben für "NLP" eingesetzt werden. Als nächstes demonstrieren wir an Hand eines semantischen Parsers, dass der Ansatz auch dann angewendet werden kann, wenn das zugrunde liegende Modell ein neuronales Netzwerk ist und das "Feedback" von menschlichen Nutzern gesammelt wurde. Die Anwendung beider Ansätze auf dasselbe Problem für semantisches Parsen, ermöglicht es uns einen direkten Vergleich zu ziehen. "Response-based on-policy learning" übertrifft "counterfactual off-policy learning", aber es benötigt teure parallele Daten für die nachfolgende Aufgabe, während Logdateien von Nutzer-System Interakationen für "counterfactual off-policy learning" in verschiedenen Szenarien einfacher zu erhalten sind.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

B2S Bandit-to-Supervised.

$\hat{c}$DC $\hat{c}$ Doubly Controlled.

$\hat{c}$DR $\hat{c}$ Doubly Robust.

CFG Context-Free Grammar.

CGI Common Gateway Interface.

CSS Cascading Style Sheets.

DC Doubly Controlled.

DM Direct Method.

DPM Deterministic Propensity Matching.

DPM+OSL One-Step-Late Reweighted Deterministic Propensity Matching.

DPM+R Reweighted Deterministic Propensity Matching.

DPM+T Token-level Deterministic Propensity Matching.

DPM+T+OSL Token-Level One-Step-Late Reweighted Deterministic Propensity Matching.

DR Doubly Robust.

EM Expectation-Maximisation.

GRU Gated Recurrent Unit.

HTML Hypertext Markup Language.

IPS Inverse Propensity Scoring.

IPS+R Inverse Propensity Scoring with Reweighting.

JSON JavaScript Object Notation.

LSTM Long Short-Term Memory.

MLE Maximum Likelihood Estimation.

MRL Machine Readable Language.

MRT Minimum Risk Training.

MRT+neg Minimum Risk Training with negative rewards.

NER Named Entity Recognition.

NLP Natural Language Processing.

OSL One-Step-Late.

OSM OpenStreetMap.

QA Question-Answering.

Ramp Ramp loss with y^+ and y^- set according to an external metric.

Ramp+T Token-level Ramp loss with y^+ and y^- set according to an external metric.

Ramp1 Ramp loss with $y^+ = \hat{y}$ and y^- set according to an external metric.

Ramp2 Ramp loss with $y^- = \hat{y}$ and y^+ set according to an external metric.

Rampion Rampion as defined by Gimpel and Smith (2012).

Rebol Response-based Online Learning.

RNN Recurrent Neural Network.

SCFG Synchronous Context-Free Grammar.

SMT Statistical Machine Translation.

XML Extensible Markup Language.

Chapter 1

Introduction

Many applications nowadays are built upon machine-learnt statistical models which, given an input, produce an output that is presented to a user. For example, virtual personal assistants have been on the rise to help an increasing number of people with an increasing number of everyday tasks, such as placing an order for a product online, navigating a music playlist or question-answering on various topics. The underlying statistical models are typically trained using fully supervised data where inputs are paired with corresponding gold targets, i.e. labelled data is available for direct supervision. However, the creation of such labelled data is often costly and time-consuming and thus it is fruitful to explore alternative learning methods. One option is to learn from weaker supervision signals instead, such as from feedback obtained for model outputs. This reduces the need for expensive gold targets. Additionally, there are often alternative outputs that are as good as the annotated gold target, but are never discovered if a model relies only on learning from annotated gold targets. Finally, learning from feedback also enables models to continue improving over time and it is possible to personalise a model to specific user needs.

Given model outputs and corresponding extrinsic feedback, we investigate how to improve sequence-to-sequence models in Natural Language Processing (NLP) tasks. This setup is visualised in Figure 1.1. Given a pre-trained model, we explore two distinct approaches that leverage feedback given to model outputs to further improve a model. The two approaches are applied to two different tasks, machine translation and semantic parsing for question answering, and to two different model types, linear and non-linear. For both approaches we have a concrete application in mind. This application is the creation of a natural language interface to the geographical database OpenStreetMap (OSM). With two approaches to learn from feedback and one application, this thesis is comprised of three parts that we elaborate on in the following.

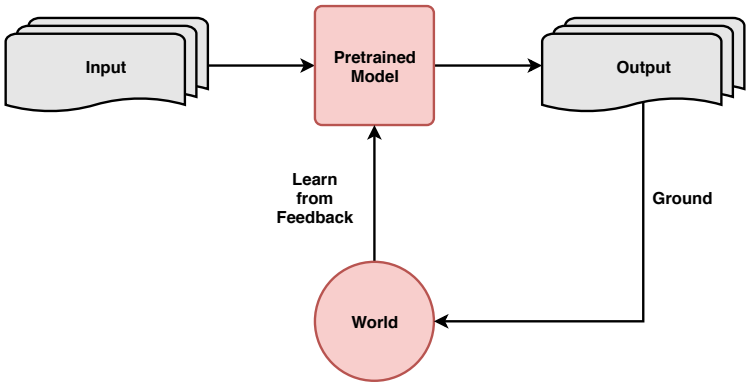


Figure 1.1.: General setup of experiments conducted in this thesis.

1.1. A Natural Language Interface to OpenStreetMap (OSM)

We introduce and define the task of building a natural language interface to OpenStreetMap (OSM), which is a geographical database populated by volunteers with GPS coordinates and relevant facts about points of interests all over the world. The basis of the interface will be a semantic parser that maps natural language questions to parses that can be executed against the OSM database. To achieve this goal, we first define an appropriate Machine Readable Language (MRL) that can express natural language questions as semantic parses. Based on this, we create a Question-Answering (QA) corpus, named NLmaps, which consists of question-parse pairs. Corresponding answers can be obtained by executing the parses against the OSM database. To the best of our knowledge, this constitutes the first publicly available corpus for semantic parsing in the OSM domain. We compare both corpora to two similar question-parse corpora, namely Geoquery (Wong and Mooney, 2006) and Free917 (Cai and Yates, 2013). Two extensions to the corpus lead to a larger corpus, referred to as NLmaps v2, which totals 28,609 question-parse pairs.

Using both versions of the NLmaps corpus, several semantic parsing models are built in a supervised manner which serve as baselines for various tasks throughout the thesis. We employ two different frameworks for the different semantic parsing models. The first is Cdec (Dyer et al., 2010), a hierarchical phrase-based Statistical Machine Translation (SMT) framework, which is modified to be suitable for semantic parsing. The second is a sequence-to-sequence framework based on Recurrent Neural Networks (RNNs), called Nematus (Sennrich et al., 2017).

Furthermore, we describe a specifically tailored graphical web interface that users can access online to pose natural language questions to the OSM database. For this, we introduce a few modifications to the baseline models which aim to offer additional conveniences to users of the interface. This constitutes the application-oriented contribution of this thesis. In the following we turn to the two distinct approaches for learning from feedback to improve statistical models for sequence-to-sequence tasks in NLP.

1.2. Response-Based On-Policy Learning

The first approach that we explore is **response-based on-policy learning**. In some situations, it is more beneficial to learn from an indirect supervision signal rather than using directly supervised data. In grounded learning a system learns by viewing its outputs as actions in a downstream task and outputs are considered successful if they lead to correct task feedback. For response-based on-policy learning, we ground a model in a downstream task for which labelled data has been collected.

Grounded learning has several advantages over supervised learning. First, the indirect supervision signal might be easier or cheaper to obtain than directly supervised data. Second, it allows the model to search for suitable outputs within the its own output space. Third, it can alleviate the need for expensive gold targets because it is possible to find multiple, alternative outputs that lead to positive task feedback. Learning from multiple positive outputs rather than the single gold target, can lead to a more robust system. Fourth, for some underlying models, it might not be possible to produce the annotated gold target. Response-based on-policy learning can instead identify a possible alternative that is close to the gold target, but can be produced by the system, and this alternative can serve as a pseudo-gold target.

We apply response-based on-policy learning to two different tasks. In the first task, we aim to tune a machine translation system in a multilingual semantic parsing pipeline for question-answering. A machine translation system is given a question in a source language and translates this question into a target language. We assume a semantic parser exists for this target language and it can map questions in the target language to a semantic parse. The parse can then be executed against a database to obtain an answer. The machine translation system is grounded in this final task of obtaining the correct answer given a question in the source language. With this grounding, the machine translation system learns to work better in conjunction with the semantic parser and this raises the overall final task performance. Concretely, we assume that the machine translation system is given questions in German from the NLMaps corpus and translates these questions into English. The English question is then handed to a semantic parser and the resulting parse is executed against the OSM database.

For this response-based learning scenario on multilingual semantic parsing, we introduce two algorithms. Both algorithms employ a ramp loss objective (Collobert et al., 2006) that identifies a hope translation, which we want to encourage, and a fear translation, which we want to discourage. For both algorithms, we require the existence of the gold answer to obtain feedback. This is advantageous because with an available gold answer, feedback can be obtained for arbitrarily many model outputs. The first response-based algorithm, called Ramp, solely utilises binary feedback obtained at the answer level. The second algorithm, Rebol, additionally leverages feedback derived from comparing the output of the machine translation system to a correct reference translation. Rebol outperforms Ramp, but requires the existence of two gold structures, namely the gold answer and a gold reference translation, which might be too expensive to obtain in praxis. The algorithms are inspired by Gimpel and Smith (2012), who use similar ramp loss objectives to tune a machine translation system on the basis of available gold references, but they do not ground their system

in a final task. Executing a parse and obtaining feedback by comparing it to the gold result, is inspired by Goldwasser and Roth (2013), who used such a signal to improve a semantic parser.

In a second task, we directly improve a semantic parser based on the NLmaps v2 corpus and with an underlying neural network. We assume that an initial parser was trained using a small amount of supervised question-parse pairs as training data. Further data is only available in the form of question-answer pairs and the model is guided by comparing the answers obtained for model outputs to the gold answers. This is a prominent research topic in the semantic parsing community in recent years (Clarke et al. (2010); Berant et al. (2013); Pasupat and Liang (2015); Rajpurkar et al. (2016); *inter alia*) because for many domains it is significantly easier to obtain gold answers rather than gold parses. For neural semantic parsing from question-answer pairs, the use of MRT (Liang et al., 2017; Guu et al., 2017), objectives inspired by the REINFORCE algorithm (Liang et al., 2017; Mou et al., 2017; Guu et al., 2017) or objectives based on classical structured prediction (Iyyer et al., 2017; Misra et al., 2018) have been explored.

We lift several ramp loss objectives (Gimpel and Smith, 2012) to neural network models and compare them to MRT. In experiments we can show that it is crucial to carefully identify a hope parse and a fear parse which are encouraged and discourage, respectively. Additionally, we analyse why the best ramp loss objective, Ramp, outperforms the MRT objective. Furthermore, we introduce a novel ramp loss objective, Ramp+T, that operates at the token level. It can outperform its sequence-level counterpart Ramp by more effectively contrasting tokens of the hope and fear parses against each other.

For both tasks, response-based on-policy learning requires the existence of at least one gold target for each input, albeit the weakly supervised gold targets of the downstream task might be cheaper to obtain than directly supervised gold targets. In the second approach to learn from feedback, we investigate how we can learn without any expensive gold targets by leveraging feedback collected for model outputs from user-system interactions instead.

Overview: Response-based On-Policy Learning

In the following we provide an overview of the novelties that our application of response-based on-policy learning introduces in this thesis and the advantages and disadvantages this approach offers.

Novelties

- Models are grounded in a downstream task. First, we ground a machine translation system in a semantic parser for a multilingual semantic parsing pipeline. Second, we improve a semantic parser using question-answer pairs rather than question-parse pairs.
- Ramp loss objectives are applied to non-linear neural networks.
- Introduction of a token-level ramp loss objective.

Advantages

- It is possible to obtain feedback for arbitrarily many output structures.
- The approach allows us to find alternative outputs with positive task feedback, rather than only learning from one annotated gold target.
- For log-linear models, the gold target might not be reachable and this approach allows us to identify and promote an alternative pseudo-gold target.
- The need for expensive gold targets is alleviated because for some tasks it is easier to obtain a downstream task's gold target than a gold target for direct supervision. This is for example the case in semantic parsing for question answering, where in certain domains gold answers are easier to obtain than gold parses.

Disadvantages

- Although the gold target might be easier to obtain for a downstream task, this approach still requires at least one gold target per input.
- The approach is potentially slow because for each input several outputs have to be scored by the feedback function, which might be time consuming.

1.3. Counterfactual Off-Policy Learning

The second approach to learn from feedback is **counterfactual off-policy learning**. Deployed applications can easily and cheaply collect logs of user-system interactions in large quantities. For such a log, we record the user input, the output of the application's underlying statistical model and feedback for the presented output, which is elicited either explicitly or implicitly from the user. Learning from such a log is an interesting endeavour because the log collection does not incur any cost and the approach does not require the existence of expensive gold targets.

The log can be used to improve the logging model or any other model. Because the data in the collected log was not produced by the to-be-improved model, this leads to an off-policy learning setup (Sutton and Barto (2018), Section 13.3).[1] Such a more complex learning scenario could be avoided by improving the logging model while deployed, but we opt against this course of action for several reasons. In such an approach, the model would be updated after every interaction, but this can be both dangerous and expensive. A model that evolves while deployed could degenerate without notice and this would lead to user dissatisfaction and monetary loss. Further, it could be very expensive or impossible to explore various hyperparameters of the model while it is deployed, but this is easily possible for off-policy methods. Additionally, off-policy learning is safer because models can be validated on separate test sets before deployment which prevents fielding potentially degenerate models.

[1] Note that even if we choose to improve the logging model, once a single update has been made to its parameters, we are also faced with an off-policy learning scenario.

Due to the these advantages, we focus on off-policy learning from a previously collected log.

Off-policy learning from a collected log of user-system interactions is a complex learning scenario for several reasons. First, feedback exists only for the one output shown to the user. This is a bandit learning scenario, because it remains unknown what feedback would have been obtained if another output was chosen, i.e. other outputs cannot be judged. Second, the collected log is biased towards the choices of the deployed logging model. This leads to the following counterfactual question: *How would the target model have performed if it had been in control during logging?* Counterfactual estimators (Bottou et al. (2013); Swaminathan and Joachims (2015a); Thomas and Brunskill (2016); Dudik et al. (2011); *inter alia*) have been developed that attempt to answer this question by explicitly correcting the data bias. Further problems arise during the application of these estimators and these are outlined below.

Previously introduced counterfactual estimators have so far only been applied to structured prediction problems with modestly sized output spaces, but sequence-to-sequence NLP tasks typically have very large output spaces. We apply counterfactual learning to two NLP tasks, namely machine translation and semantic parsing for question-answering, to show that it is possible to employ counterfactual estimators on tasks with large output spaces.

Furthermore, theory suggests that the logging model needs to explore the output space sufficiently to correct the data bias, e.g. by sampling outputs from the model distribution. But such a setup is dangerous in commercial NLP applications because sampling from the model distribution carries a high risk of presenting bad outputs to the user, which can lead to a monetary loss. Consequently, in commercial applications the user is always presented with the most likely output under the logging model. As a result the logging is deterministic and it is no longer possible to correct the data bias.

We present a simple estimator based on empirical risk minimisation for deterministic logging. This estimator is extended twice by using control variates which correct deficiencies in the previous estimators and reduce variance. To show that deterministic logging if feasible for sequence-to-sequence learning in NLP, we compare the deterministic estimators to stochastic counterparts. This is done on a domain adaptation task for machine translation where we simulate feedback to be able to draw direct comparisons between deterministic and stochastic estimators. The experiments show no significant difference between the best deterministic and the best stochastic estimators and we provide a possible explanation why deterministic logging is admissible for sequence-to-sequence learning in NLP. Furthermore, we also present mathematical proofs and intuitive explanations of the deficiencies in two counterfactual estimators for both their deterministic and stochastic instantiations.

On a semantic parsing task based on the NLmaps v2 corpus, we show that counterfactual learning is also applicable if the underlying model is a non-linear neural network. State-of-the-art neural networks for sequence-to-sequence learning are trained using stochastic gradient ascent[2] with small minibatches. However, one important extension of the basic

[2]Because we formulate our counterfactual estimators as *reward* estimators, we employ the term gradient ascent, rather than gradient descent. By negating the rewards, we would recover equivalent *risk* estimators with which gradient descent could be performed.

counterfactual estimator relies on the use of large minibatches, which is not possible for neural networks due to hardware limitations. We offer an alternative estimator that preserves all important properties and can be applied to stochastic gradient ascent with small minibatches. Furthermore, we introduce a counterfactual estimator that can incorporate feedback at the token level. This enables blame assignment within a sequence and facilitates better learning.

Finally, it is necessary to show that counterfactual learning for sequence-to-sequence tasks is possible if the feedback in the log is collected from actual human users. Using our semantic parsing task based on the NLmaps v2 corpus, we set up a corresponding feedback collection form. Given a question, the semantic parser produces a parse, which is executed against a database to retrieve an answer and this answer is then shown to the user. However, in most cases, it will be impossible for the user to judge whether or not the given answer is correct or not. For example, for the question "*How many hotels are there in Paris?*", we cannot expect a user to verify that "*951*" is the correct answer. Similarly, we cannot ask the user to verify a semantic parse because the underlying MRL will be unknown to everyday users. Instead, we propose to automatically convert the parse into a set of human-understandable statements that can be judged as correct or incorrect by non-expert users.

For this proposed feedback collection method, we recruit 9 human users and we can show that the collection method is very efficient with the majority of the feedback forms being filled out in 10 seconds or less. With this feedback, we can show that counterfactual learning is possible if feedback is collected from human users. Additionally, our proposed feedback collection method naturally allows us to obtain feedback at the token-level, which enables us to use superior token-level counterfactual estimators.

By employing the same semantic parsing tasks based on the NLmaps v2 corpus for both response-based on-policy and counterfactual off-policy learning, we can conclude by drawing a direct comparison between both approaches of learning from feedback.

Overview: Counterfactual Off-Policy Learning

In the following we provide an overview of the novelties that our application of counterfactual off-policy learning introduces in this thesis and the advantages and disadvantages this approach offers.

Novelty

- First application of counterfactual learning to problems with large output spaces, namely to two sequence-to-sequence tasks in NLP: machine translation and semantic parsing for question-answering.
- Insight that deterministic logging is on par with stochastic logging for sequence-to-sequence tasks in NLP.
- Insight that counterfactual learning is possible for non-linear neural networks.

- Introduction of counterfactual estimators that can incorporate token-level feedback.
- Modification of a counterfactual estimator to be suitable for stochastic minibatch gradient ascent.

Advantages

- The approach is built upon theoretically well-understood policy gradient methods (see for example (Sutton and Barto, 2018), Chapter 13).
- Gold targets are not required.
- Off-policy learning is safer, more hyperparameters can be explored and models can be validated against separate test sets before deployment.

Disadvantages

- Feedback is only available for one output structure; feedback for other output structures cannot be obtained.
- Collected logs are biased towards the logging model, which complicates learning.
- Simple counterfactual estimators suffer from degeneracies and high variance which need to be countered with more sophisticated estimators.

1.4. Summary of Contributions

The contributions of this thesis can be grouped into three distinct categories and can be summarised as follows:

Semantic Parsing for Question-Answering

- A QA corpus, NLmaps, consisting of 2,380 question-parse pairs (Haas and Riezler, 2016) and an extension, NLmaps v2, with 28,609 question-parse pairs (Lawrence and Riezler, 2018).
- A natural language web interface to OpenStreetMap (OSM) (Lawrence and Riezler, 2016).
- A method to automatically convert parses into a set of statements which can be easily and efficiently judged by human users as correct or incorrect (Lawrence and Riezler, 2018).

Grounded Learning

- Successful grounding of a machine translation system in its downstream task for a multilingual semantic parsing pipeline (Haas and Riezler, 2016).
- Application of ramp loss objectives to non-linear neural networks, detailed evaluation and comparison to MRT objectives for semantic parsing (Jehl et al., 2019).
- Introduction of a token-level ramp loss objective (Jehl et al., 2019).

Off-policy Reinforcement Learning

- Definition of deterministic counterfactual estimators: Deterministic Propensity Matching (DPM), Reweighted Deterministic Propensity Matching (DPM+R), Doubly Controlled (DC) and $\hat{c}$ Doubly Controlled ($\hat{c}$ DC) (Lawrence et al., 2017b).
- Empirical evidence and intuitive explanation that deterministic logging for counterfactual learning provides sufficient exploration on sequence-to-sequence tasks in NLP (Lawrence et al., 2017b; Lawrence and Riezler, 2018).
- Empirical evidence that counterfactual learning can be applied to problems with large output spaces (Lawrence et al., 2017b; Lawrence and Riezler, 2018).
- Empirical evidence that counterfactual learning can be applied to non-linear neural network models (Lawrence and Riezler, 2018).
- Mathematical proofs and intuitive explanations of potential degeneracies in the counterfactual estimators: Inverse Propensity Scoring (IPS), Deterministic Propensity Matching (DPM), Reweighted Inverse Propensity Scoring (IPS+R) and Reweighted Deterministic Propensity Matching (DPM+R) (Lawrence et al., 2017a).
- Definition of the estimator One-Step-Late Reweighted Deterministic Propensity Matching (DPM+OSL), which modifies the Reweighted Deterministic Propensity Matching (DPM+R) estimator to be applicable for stochastic (minibatch) gradient ascent (Lawrence and Riezler, 2018).
- Definition of the counterfactual estimator Token-level Deterministic Propensity Matching (DPM+T), which allows rewards to be decomposed over the tokens of the output sequence (Lawrence and Riezler, 2018).

1.5. Structure

This thesis is a cumulative effort. Consequently significant parts of this thesis have been previously published in peer-reviewed publications, co-authored with my supervisor and colleagues. Additionally some sections have previously appeared in my Master's thesis. The papers are referenced in the introduction of the corresponding chapters of this thesis and work not done by my person is clearly marked.

Relevant background knowledge as well as notation used throughout this thesis is presented in Chapter 2. Part I presents the question-answering task defined on the geographical database OpenStreetMap (OSM). Chapter 3 details the construction of the QA corpora NLmaps and NLmaps v2, both of which are used in subsequent chapters to demonstrate the effectiveness of various algorithms. Two semantic parsing frameworks and their extensions tailored for the NLmaps domain are introduced in Chapter 4. The chapter also presents the graphical user interface constructed for the natural language interface to OSM, which connects to a semantic parser in the background.

The first approach to learn from feedback, response-based on-policy learning, is presented in Part II. It considers scenarios where models are grounded in a downstream task for which gold targets are available and which can provide a weak supervision signal. The approach is applied to two different model types, linear and non-linear, and to two different tasks. In Chapter 5, we build a multilingual semantic parser by employing a machine translation system and a semantic parser in a pipeline setup. The machine translation system is grounded in the semantic parsing task and is improved by leveraging final task feedback provided on the basis of gold answers. In a similar scenario, we assume that only gold answers but no gold parses are available to improve a semantic parser in Chapter 6.

The second approach to learn from feedback, counterfactual off-policy learning, is presented in Part III. It assumes that neither immediate gold targets are available, nor is it possible to ground the system in a task with available gold targets. Instead, we assume that a system has been deployed and users interacting with the system provide feedback for the one model output presented to the user. As with the previous approach, this approach is also applied to two different model types, linear and non-linear, and to two different tasks. In Chapter 7 we improve a machine translation system in this setup using both deterministic and stochastic logs which are created with simulated feedback. Chapter 8 presents an approach to improve a semantic parser by leveraging feedback provided by real human users. We conclude by drawing a direct comparison between the two approaches based on identical experimental setups in Chapter 6 and Chapter 8.

Chapter 2

Background

Semantic parsers and machine translation systems are the focus in this thesis. Both are sequence-to-sequence tasks and can be cast as structured prediction problems. Here we briefly introduce relevant terminology and background information. Section 2.1 introduces general notation and a basic overview of the task of structured prediction. An overview of phrase-based SMT is given in Section 2.3 and an overview of recurrent neural networks for sequence-to-sequence learning in Section 2.4. Section 2.2 provides an overview of relevant reinforcement learning concepts.

2.1. Structured Prediction

For the scope of this thesis, we are interested in the task of structured prediction conditioned on some input. Given an input sequence $x = x_1, x_2, \dots x_{|x|} \in \mathbf{X}$, we assume the existence of a parametric model $\pi_w(y|x)$ with parameters w that defines a conditional probability distribution over all possible output sequences $y \in \mathbf{Y}(x)$, where $\mathbf{X}$ and $\mathbf{Y}(x)$ are the set of all possible input and output values, respectively. The most likely output sequence under $\pi_w(y|x)$ is defined as

$$\hat{y} = \arg\max_{y \in \mathbf{Y}(x)} \pi_w(y|x). \tag{2.1}$$

The goal of a machine learning algorithm is to find a set of optimal parameters w, so that, given an input x, $\pi_w(y|x)$ assigns the highest probability to the correct gold target sequence $\bar{y} = \bar{y}_1, \bar{y}_2, \dots \bar{y}_{|\bar{y}|}$.

For an input x and corresponding gold target $\bar{y}$, we can define a loss function $\mathcal{L}(\hat{y}, \bar{y}))$ which registers a penalty if $\hat{y} \neq \bar{y}$. Assuming a probability distribution $p(x, y)$ that maps $\mathbf{X} \times \mathbf{Y}$ to the interval $[0, 1]$, the expected risk is defined as

$$\mathcal{L}_{expected} = \int_x \int_y \mathcal{L}(\hat{y}, \bar{y}))p(x, y)\mathrm{d}x\mathrm{d}y = \mathbb{E}_{(x,y)\sim p(x,y)}[\mathcal{L}(\hat{y}, \bar{y})]. \tag{2.2}$$

As $p(x, y)$ is generally unknown, this distribution needs to be approximated. This can be done using a training set of size n that is independent and identically distributed (i.i.d.) with a probability distribution $\tilde{p}(x, y)$ and for each input x_t a corresponding gold target $\bar{y}_t$ has been collected. Thus the expected risk is approximated using the empirical risk,

$$\mathcal{L}_{empirical} = \frac{1}{n} \sum_{t=1}^{n} \mathcal{L}(\hat{y}, \bar{y}_t). \tag{2.3}$$

The empirical risk will approach the expected risk as $n \to \infty$.

To improve a model, we can perform **empirical risk minimisation**. Concretely, in this thesis we modify the parameters w using gradient descent. The parameters of w are adjusted by moving in the negative direction of the gradient that results from the differentiation of the loss function with regards to w. Thus for each input-output pair $(x, \bar{y})$, w is updated as follows:

$$\Delta w = -\eta \nabla \mathcal{L}(\hat{y}, \bar{y}), \tag{2.4}$$

where η is a learning rate to be set separately.

If we want to employ rewards rather than losses, we can define a reward function instead:

$$\mathcal{V}(\hat{y}, \bar{y})) = -\mathcal{L}(\hat{y}, \bar{y}). \tag{2.5}$$

We then use the term gradient ascent instead, with the corresponding update defined as:

$$\Delta w = \eta \nabla \mathcal{V}(\hat{y}, \bar{y}). \tag{2.6}$$

Instead of updating after each seen input-output pair (x, y), we can also operate in a batch setup where we iterate over the entire training set of size n and average the loss before performing an update. Batch algorithms more closely approximate the expected risk, however it can be expensive to compute because it requires an iteration over the whole training set before an update is performed. Alternatively, we utilise minibatches of size $m < n$. While minibatches are less accurate due to $m < n$, it offers a trade-off between the two extremes.

Repeatedly updating w on the basis of the training set leads to the danger of overfitting to the training set, which results in bad performance on new, previously unseen data. To prevent this from occurring, we employ development sets. We periodically evaluate the performance on the development set and chose the set of parameters w where the performance on the development set is highest. The resulting model $\pi_w(y|x)$ is then deployed on a test set and the resulting performance is used to judge the success of $\pi_w(y|x)$.

We speak of full or direct supervision if data is available that pairs inputs x with gold targets $\bar{y}$ for the task. In the case of machine translation, an input sentence in a source language is paired with a reference translation in the target language. For semantic parsing, input questions are paired with a correct parse that by definition executes to the correct answer. We

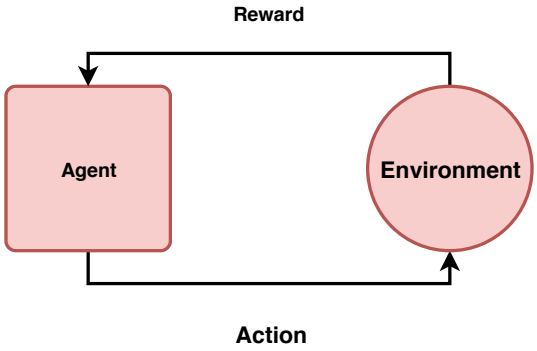


Figure 2.1.: Schematic view of reinforcement learning.

speak of weak supervision if gold targets are not available and instead feedback to model outputs y are used as a learning signal. This is elaborated on further in the following section.

2.2. Reinforcement Learning

Parts II and III of this thesis employ ideas from reinforcement learning. Here we present relevant concepts and terminology. For more details see (Sutton and Barto, 2018), which serves as a basis for this short overview.

In fully supervised learning we are given a training set where inputs x are associated with gold targets $\bar{y}$. As described in Section 2.1, if a system's most likely output $\hat{y}$ does not equal the gold target $\bar{y}$, we can measure a loss $\mathcal{L}(\hat{y}, \bar{y})$ and update the underlying parametric model accordingly. In reinforcement learning, gold targets $\bar{y}$ are not available. However, we assume there is an environment or world, to which we can give a chosen model output y, also called the action in reinforcement learning terms, and the world returns a reward $\delta(y)$ or, equivalently, a loss $\Delta(y) = -\delta(y)$.[3] The environment is otherwise unknown to us and is treated as a black box. The goal of reinforcement learning is to improve an agent so that the expected reward on a given task is maximised. See Figure 2.1 for a graphical overview.

We assume that the agent is fully specified by a parametric policy $\pi_w(y|x)$ that defines a conditional probability distribution over all possible outputs $y \in \mathbf{Y}(x)$ for a given $x \in \mathbf{X}$. Outputs from $\pi_w(y|x)$ can be generated either stochastically or deterministically. A **stochastic policy** draws samples from the output space, e.g. by sampling from the model distribution $\pi_w(y|x)$. A **deterministic policy** on the other hand always returns the same output given the same input. The most straightforward option is to compute the most likely output under the current model, i.e. $\arg\max_{y \in \mathbf{Y}(x)} \pi_w(y|x)$.

Given an output, we assume that the environment can return a reward for this output. If the environment can evaluate only one output per input and thus only return one reward, we

[3] Note that we use the terms loss and reward interchangeably in this thesis, whereas we assume that $\Delta(y) = -\delta(y)$ unless otherwise specified.

refer to this as a **bandit** setup (Bubeck and Cesa-Bianchi, 2012). The phrase "*bandit*" stems from the analogy of having to choose a slot machine (colloquially referred to as "*one-armed bandit*") amid others. Ideally, the chosen machine obtains the best reward compared to the reward that any other slot machine would have returned. However, once a machine (or output) is chosen, it will remain unknown if any of the other machines (or other outputs) would have led to a better reward. This scenario occurs frequently for deployed systems where only one output can be presented to a human user and thus feedback can only be collected for the one output.

To improve our agent, we would like to minimise the expected risk of the model $\pi_w(y|x)$,

$$\mathcal{L}(\pi_w(y|x)) = \mathbb{E}_{p(x)}\mathbb{E}_{\pi_w(y|x)}[\Delta(y)], \tag{2.7}$$

where $p(x)$ is the probability distribution over inputs x. This is equivalent to maximising the expected reward,

$$\mathcal{V}(\pi_w(y|x)) = \mathbb{E}_{p(x)}\mathbb{E}_{\pi_w(y|x)}[\delta(y)]. \tag{2.8}$$

We would like to increase the expected reward using **policy gradient techniques** (Sutton and Barto (2018), Chapter 13). Thus, we need to compute

$$\nabla_w \mathcal{V}(\pi_w(y|x)) = \nabla_w \mathbb{E}_{p(x)}\mathbb{E}_{\pi_w(y|x)}[\delta(y)]. \tag{2.9}$$

However, this is not directly possible because the reward function $\delta(y)$ is unknown to us and can consequently not be derived. This problem can be circumvented using the **score function gradient estimator** (Fu, 2006):

$$\begin{aligned}
\nabla_w \mathcal{V}(\pi_w(y|x)) &= \nabla_w \mathbb{E}_{p(x)}\mathbb{E}_{\pi_w(y|x)}[\delta(y)] \\
&= \nabla_w \int_x \int_y \delta(y)\, p(x)\mathrm{d}x\, \pi_w(y|x)\mathrm{d}y \\
&= \int_x \int_y \delta(y)\, p(x)\mathrm{d}x\, \nabla_w \pi_w(y|x)\mathrm{d}y \\
&= \int_x \int_y \delta(y)\, p(x)\mathrm{d}x\, \nabla_w \frac{\pi_w(y|x)}{\pi_w(y|x)}\, \pi_w(y|x)\mathrm{d}y \\
&= \int_x \int_y \delta(y)\, p(x)\mathrm{d}x\, \nabla_w \log \pi_w(y|x)\, \pi_w(y|x)\mathrm{d}y \\
&= \mathbb{E}_{p(x)}\mathbb{E}_{\pi_w(y|x)}[\delta(y)\nabla_w \log \pi_w(y|x)].
\end{aligned} \tag{2.10}$$

The final expectation of Equation 2.10 can be approximated using Monte Carlo simulation. Observing an input x, we obtain a model output y from $\pi_w(y|x)$ and a corresponding reward $\delta = \delta(y)$. Based on this, we can formulate the following gradient ascent update rule, generally known as the REINFORCE algorithm (Williams, 1992), to adjust w:

$$\Delta w = \eta \, \delta \, \nabla_w \log \pi_w(y|x), \tag{2.11}$$

where η is a learning rate to be set separately.

Due to obtaining only one sample, the algorithm can suffer from high variance which can be reduced by employing **control variate**s (Ross (2013), Chapter 9) and we consider this further in Chapters 7 and 8.

To effectively learn, a policy needs to explore the output space to find outputs that lead to high rewards. The most straightforward way to explore the output space is to sample from the model distribution. But such stochastic policies can be dangerous for real-world setups where a sampled bad output might lead to dissatisfied users and a monetary loss. The risk can be reduced by peaking the model distribution, which further increases the probability of high probability outputs. For example, for a log-linear model,

$$\pi_w(y|x) = \frac{e^{\alpha w \phi(x,y)}}{\sum_{y \in \mathbf{Y}(x)} e^{\alpha w \phi(x,y)}}, \tag{2.12}$$

we can set a factor $\alpha \in \mathbb{R}^+$ so that as $\alpha \to \infty$ we approach a deterministic policy that chooses the most likely output and thus exploits the current model. If a deterministic policy is used, learning might be limited because the policy does not have the chance to explore alternative, potentially better outputs. Such a deterministic policy would however greatly reduce the risk for deployed real-world systems. We are faced with the problem of finding a good **exploration-exploitation** trade-off. For sequence-to-sequence tasks of machine translation and semantic parsing, the danger of exploring is particularly high because there are vastly more bad options to choose compared to an output that would lead to a good reward. In such setups, we would like to always choose the most likely output under the current model. We will investigate in Chapter 7 if deterministic policies are viable for machine translation or if their lack of explicit exploration poses a problem.

A further complication arises if we want to improve a policy $\pi_w(y|x)$, but outputs y and associated rewards $\delta(y)$ have been produced by another policy $\mu(y|x)$. In this scenario, there is an inherent bias towards the policy $\mu(y|x)$ in the collected data. This leads to a **counterfactual** setup because we do not know what outputs and associated rewards would have been collected if $\pi_w(y|x)$ had been in charge during the data collection. This is also referred to as an **off-policy** setup in reinforcement learning terms. As a contrast to this, in an **on-policy** setup, the agent that is to be improved is also the agent that chooses the outputs.[4] In Part II we are concerned with on-policy setups, whereas in Part a III we consider off-policy scenarios.

[4]The definition for on-policy and off-policy follows the definition of (Sutton and Barto (2018), Section 5.4).

2.3. Phrase-Based Machine Translation

We give a short introduction on phrase-based machine translation which is employed in Chapters 4, 5 and 7. In particular, we use the hierarchical phrase-based framework Cdec (Dyer et al., 2010). For an extended discussion of the topic see Koehn (2010), which is also the source upon which this summary is based.

Given a text segment in a language **x**, the goal of a machine translation system is to output the most likely translation of this text segment in language **y**. For a segment x in language **x**, the probability of translating it into a text segment y in language **y** is $p(y|x)$. In machine translation, a text segment typically equates a sentence. The probability $p(y|x)$ can be re-written using Bayes' rule:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} \propto p(x|y)p(y). \tag{2.13}$$

The highest scoring translation $\hat{y}$ in the output space $\mathbf{Y}(x)$, which contains all possible sentences of language **y**, can then be found using

$$\hat{y} = \arg\max_{y \in \mathbf{Y}(x)} p(x|y)p(y), \tag{2.14}$$

where the division by $p(x)$ may be omitted as it does not impact the ranking of translation candidates. Note that calculating the probability for every element $y \in \mathbf{Y}(x)$ is typically infeasible due to the size of $\mathbf{Y}(x)$. Thus 2.14 is generally approximated using a greedy search or a beam search algorithm. The reformulation using Bayes' rule allows a rudimentary machine translation system to be split into two components: $p(x|y)$ is the translation table that assigns probabilities for translating x into y, and $p(y)$ is the language model which judges how likely sentence y is for language **y**.

Modelling an entire sentence is difficult, thus sentences are split into smaller units, $x = x_1, x_2, \ldots x_{|x|}$ and $y = y_1, y_2, \ldots y_{|y|}$. Units are typically words, but could also be sub-word units (Sennrich et al., 2016). The translation table is thus typically decomposed into the chosen smaller unit (here assumed to be words):

$$p(x|y) = \prod_{i=1}^{|y|} p(x_i|y_{a(i)}), \tag{2.15}$$

where $a(i)$ is an alignment function that maps the target language word at position i to to a source language word at position j, i.e. $a : i \rightarrow j$. Word alignments are learnt in an unsupervised fashion from parallel corpora that have been aligned already at sentence level.

If either the word alignments or the word translation probabilities were known, it would be straightforward to estimate the other using maximum likelihood estimation. As neither are known, a different approach is necessary. Starting out with uniform probabilities of aligning a source word to any target word, the Expectation-Maximisation (EM) algorithm can be used to iteratively uncover word alignments and word translation probabilities. In its expectation step, the algorithm collects counts of co-occurrences between words of both languages using the current word translation probabilities. In the maximisation step, the word translation probabilities are updated using the current counts. Once the algorithm has converged, the word translation probabilities and the most likely alignments can be extracted. This setup is referred to as the IBM 1 model (Brown et al., 1993). Latter extensions to the model led to further quality improvements, finally accumulating in the IBM 5 model (Brown et al., 1993). To obtain more robust alignments, it is also typical to run the EM algorithm for both translation directions and to heuristically combine the resulting alignments. Various strategies are possible, for example, the intersection heuristic only retains alignment points that are present in both alignment directions.

The language model $p(y)$ is trained in an unsupervised fashion using monolingual data of language $\mathbf{y}$. Given a sequence of n words (called an n-gram), maximum likelihood counts on the monolingual data can assign a probability for the next word y_j given the history of $(n-1)$ words:

$$p(y_j|y_{j-1},\ldots,y_{j-(n-1)}) = \frac{\text{count}(y_{j-(n-1)},\ldots,y_{j-1},y_j)}{\text{count}(y_{j-(n-1)},\ldots,y_{j-1})}, \tag{2.16}$$

where the function count$(\cdot)$ counts the number of occurrences in the given monolingual data for $(y_{j-(n-1)},\ldots,y_{j-1},y_j)$ and $(y_{j-(n-1)},\ldots,y_{j-1})$, respectively.

The choice of n is crucial: small n-grams might not take enough context into account but large n-grams might lead to unreliable estimates because longer sequences occur less frequently. One option is to employ a model with a large n but back off to a model with $(n-1)$ if a prospective n-gram is not contained in the larger model. Alternatively, the different models can be interpolated. Additionally, a problem arises if a word has not been seen at all in the monolingual data because it will obtain a probability of 0. This can be counteracted by smoothing all counts with some small factor α. Further smoothing methods adjust the empirical counts to make them more robust, such as Kneser–Ney smoothing (Kneser and Ney, 1995).

Translating a sentence word-by-word can be suboptimal in many cases. For instance, one word in the source language might be translated into several words in the target language and vice versa. Alternatively, idiomatic phrases might not be translated correctly if the individual words in the phrase are translated in isolation. Furthermore, phrases take context into account and can thus be helpful to resolve word ambiguities. Luckily, parallel phrases can be directly extracted from word alignments and this easily allows us to move from word-based models to phrase-based models.

A further extension to the phrase-based approach are hierarchical phrases (Chiang, 2005). Hierarchical phrases are built from phrase pairs that contain several smaller phrase pairs.

One or several of the phrase pairs are removed and replaced by a special symbol, e.g. X. Such discontinuous phrases can handle reordering elegantly. Based on these phrases, a Synchronous Context-Free Grammar (SCFG) can be extracted, which maps non-terminal symbols to two phrases, one in either language.

```
[X] → das [X_1] haus ||| the [X_1] house
[X] → kleine ||| small
```

Listing 2.1: An example for a small SCFG.

For example, given the input "*das kleine haus*" the grammar from Listing 2.1 can produce the translation "*the small house*" even if the phrase "*das kleine haus*" was never seen in the training data. In this thesis, we employ SCFGs as part of the hierarchical phrase-based framework Cdec (Dyer et al., 2010).

The translation and language models are the backbone of any traditional SMT system. Further features can be added by employing a **log-linear model**, also called **Gibbs model**,

$$\pi_w(y|x) = \frac{e^{w\phi(x,y)}}{\sum_{y' \in \mathbf{Y}(x)} e^{w\phi(x,y')}} \propto e^{w\phi(x,y)}, \tag{2.17}$$

where ϕ is a feature vector with individual entries representing different feature functions, such as $p(x|y)$ and $p(y)$, and w is a weight vector that assigns importance weights to the different features. The weights w are typically tuned in a discriminative setup with the final evaluation metric in mind. The most commonly used evaluation metric is the BLEU (Papineni et al., 2002) score which we also employed in this thesis. It is a precision based metric that calculates n-gram overlaps of a suggested translation with regards to one or more reference translations. The BLEU metric as introduced by Papineni et al. (2002) operates on the corpus level. But for the tuning algorithms, we require sentence-level metrics. We specifically mention when we use a sentence-level approximation for BLEU (Nakov et al., 2012). However, final evaluation scores are always calculated using the corpus level metric.

We employ the tuning algorithms Mert (Och, 2003), Mira (Crammer and Singer, 2003) and Rampion (Gimpel and Smith, 2012), but they all require that a gold reference translation is available. In Chapters 5 and 7 we present new algorithms which can incorporate feedback from other sources, such as from downstream tasks or human users.

2.4. Neural Networks for Sequence-to-Sequence Learning

State-of-the-art recurrent neural networks for both semantic parsing and machine translation are employed in Chapters 4, 6 and 8. In particular, we use the framework Nematus (Sennrich et al., 2017), which employs the fast computation framework Theano (Theano

Development Team, 2016). An extensive review of the topic can be found in Cho (2015) or Goodfellow et al. (2016), both of which serve as the basis for this overview.

Assume $\mathbf{V}_x$ is the set of tokens in the source language **x** and $\mathbf{V}_y$ is the set of tokens in the target language **y**. In each set, we include a special token that marks the end of a sequence and a special token that is reserved for tokens not found in $\mathbf{V}_x$ and $\mathbf{V}_y$, respectively. Each token in $\mathbf{V}_x$ is assigned a unique index and a sequence of tokens1 in language **x** can be represented by a sequence of indices $x = x_1, x_2, \dots x_{|x|}$. Equivalently, each token in $\mathbf{V}_y$ is assigned a unique index and a sequence in language **y** can be represented by a sequence of indices $y = y_1, y_2, \dots y_{|y|}$.

The neural networks employed in this thesis are based on an encoder-decoder setup (Cho et al. (2014), Sutskever et al. (2014)), augmented with an attention mechanism (Bahdanau et al., 2015). At the top level, a neural network can be understood as a parametric model $\pi_w(y|x)$, which assigns a probability of how likely it is to map an input sequence x to an output sequence y.

Initially, each x_i in the source sequence $x = x_1, x_2, \dots x_{|x|}$ is converted into a one-hot vector of length $|\mathbf{V}_x|$ where all entries are zero except the entry at position x_i which is one. Typically an additional dimension is added to the one-hot vector. This dimension is always one and acts as a bias term. For every token x_i, a Recurrent Neural Network (RNN) recursively defines a hidden state h_i, i.e.

$$h_i = g(x_i, h_{i-1}), \tag{2.18}$$

where g is some non-linear function and the first state h_0 is initialised with 0.

In an RNN, the simplest option for g would be:

$$h_i = \sigma(Wx_i + Uh_{i-1}), \tag{2.19}$$

where σ is typically the sigmoid function, W is a weight matrix of size $\mathcal{H} \times |\mathbf{V}_x|$ [5] with $\mathcal{H}$ indicating the number of hidden units and U is an additional weight matrix of size $\mathcal{H} \times \mathcal{H}$ that can capture context from previous states. However, in state-of-the-art sequence-to-sequence neural networks g is typically either a Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) (e.g. in Sutskever et al. (2014)) or a Gated Recurrent Unit (GRU) (Chung et al., 2014) (e.g. in Sennrich et al. (2017)). These are more complex units which are more adept at handling long distance dependencies and can prevent vanishing gradients (Cho (2015), Section 4.3.4). Here we introduce GRUs which are used in Nematus (Sennrich et al., 2017), the framework employed in this thesis. Mathematically it is defined via the following equations:

[5] $|\mathbf{V}_x| + 1$ if a bias term is included.

$$
\begin{aligned}
r &= \sigma(W_r x_i + U_r h_{i-1}) \\
u &= \sigma(W_u x_i + U_u (r \odot h_{i-1})) \\
\tilde{h}_i &= \sigma(W x_i + U(r \odot h_{i-1})) \\
h_i &= (1-u) \odot h_{i-1} + u \odot \tilde{h}_i,
\end{aligned}
\tag{2.20}
$$

where $\odot$ signifies element-wise multiplication and $r, u \in [0,1]^{\mathcal{H}}$. r is referred to as a reset gate whereas u is called the update gate. Intuitively the larger r is, the more information from the previous state is used. Similarly, the larger the value of u, the more information will be kept from the current state. Thus, to capture long dependencies, a high value of u can ensure that information is retained and a higher value r will indicate that it is time to use previously captured information. W_r, U_r, W_u and U_u are additional weight matrices of size $\mathcal{H} \times \mathcal{H}$.

Once a hidden state h_i has been computed for every input x_i, the hidden states can be compacted into a final representation of the entire input,

$$
c = q(\{h_1, \ldots, h_{|x|}\}),
\tag{2.21}
$$

where q could simply be the last hidden state, $q(\{h_1, \ldots, h_{|x|}\}) = h_{|x|}$ (Sutskever et al., 2014), or it is defined to obtain an average, $q(\{h_1, \ldots, h_{|x|}\}) = \frac{1}{|x|} \sum_{i=1}^{|x|} \mathbf{h}_i$, as is the case in Nematus. The vector c represents the encoded input sequence and is hence also referred to as a context vector. Often the sequence is also reversed, i.e. $\overleftarrow{x} = x_{|x|}, \ldots, x_1$, and encoded backwards, leading to a second hidden vector for each token x_i. In that case the hidden vectors from the forward pass are referred to via $\overrightarrow{h}_i$ and the hidden vectors from the backward pass as $\overleftarrow{h}_i$. An overall hidden representation is then obtained via concatenation, $h_i = [\overrightarrow{h}_i; \overleftarrow{h}_i]$.

With the input sequence encoded, the decoder RNN can output tokens in the target language. It is initialised by setting $s_0 = c$ and at each time step τ the decoder state is updated via

$$
s_\tau = z(s_{\tau-1}, y_{\tau-1}, c_\tau),
\tag{2.22}
$$

where z is a GRU with an attention mechanism and c_τ is a context vector obtained from the attention mechanism (Bahdanau et al., 2015). With a context vector c_τ for each decoder state, it allows the decoder to attend to different parts of the input sequence with varying strength depending on its current state. The attention mechanism at each step τ is defined by

$$
c_\tau = \sum_{i=1}^{|x|} \alpha_{\tau i} h_i,
\tag{2.23}
$$

where the weight $\alpha_{\tau i}$ for each input hidden vector h_i is computed by a $softmax$ function

$$\alpha_{\tau i} = \frac{e(\epsilon_{\tau i})}{\sum_{k=1}^{|x|} e(\epsilon_{\tau k})} \tag{2.24}$$

and $\epsilon_{\tau i} = a(s_{\tau-1}, h_i)$ is the alignment model that quantifies how important a token x_i is at time step τ. The function a itself defines a further neural network, typically a feed-forward network, that is trained alongside the other elements in the network. Over all entries i and τ, $\alpha_{\tau i}$ defines an alignment matrix over the sequence pair (x, y) once the translation is completed, where $\alpha_{\tau i}$ represents the probability that token y_τ is aligned to the token x_i. This alignment can be visualised using a heat map. In a heat map the values $\alpha_{\tau i}$ map to a colour depending on their value, whereas colours shift from black to white as the values move from 0 to 1. For an example see Figure 2.2.

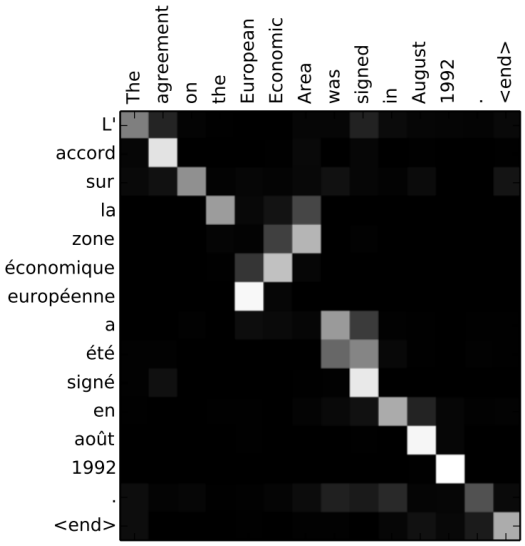


Figure 2.2.: Visualisation of alignment weights between an English sentence and a French translation. Taken from Bahdanau et al. (2015), Part (a) of Figure 3.

For each decoder state at time step τ, we can now define a probability distribution Y_τ over the output vocabulary $\mathbf{V}_y$:

$$\pi_w(Y_\tau | y_{<j}, x) = \frac{e(s_\tau)}{\sum_{v=1}^{|\mathbf{V}_y|} e(s_{\tau_v})}, \tag{2.25}$$

where π_w symbolises the neural network model and w represents all parameters that have to be learnt and $y_{<j} = y_1, y_2 \dots y_{j-1}$. $\pi_w(Y_\tau | y_{<j}, x)$ is a vector of length $|\mathbf{V}_y|$ and each entry Y_{τ_o} is the probability for translating the word that maps to the index o in the output vocabulary $\mathbf{V}_y$. At time τ, the probability of the token that is mapped to index y_j is thus:

$$\pi_w(y_j = \tau_o | y_{<j}, x) = \frac{e(s_{\tau_o})}{\sum_{v=1}^{|\mathbf{V}_y|} e(s_{\tau_v})}. \tag{2.26}$$

The probability of an entire sequence is given by

$$\pi_w(y|x) = \prod_{j=1}^{|y|} \pi_w(y_j|y_{<j}, x), \tag{2.27}$$

where we shorten $y_j = \tau_o$ to y_j for notational convenience.

With the network defined, appropriate weights w need to be learnt. This is typically done using supervised data, i.e. parallel training data where input sequences x are aligned with gold target sequences $\bar{y}$. To begin with, the weight matrices are initialised randomly, e.g. by drawing numbers from a Gaussian distribution. Applying the network to an input sequence, we look up the probability the network currently ascribes to the supplied target sequence $\bar{y}$. Given n inputs, we obtain the probability of the n corresponding output sequences and the network's parameters are updated so that probability of the training data is increased. This is possible by defining the following Maximum Likelihood Estimation (MLE) objective which increases the probability of the tokens $\bar{y}_j$ in the supplied gold target sequence $\bar{y}$:

$$\mathcal{L}_{MLE} = -\frac{1}{n} \sum_{t=1}^{n} \sum_{j=1}^{|y|} \log \pi_w(\bar{y}_{t,j}|\bar{y}_{t,<j}, x_t). \tag{2.28}$$

As every function in the network is differentiable, the gradient for the network on the basis of the above loss can be calculated using partial derivatives and the chain rule ($\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$). Via this backpropagation technique, the various steps in the RNN can be unrolled until all weight matrices are updated:

$$\frac{\delta \mathcal{L}}{\delta w} = \frac{\delta \mathcal{L}}{\delta \bar{y}_{|\bar{y}|}} \frac{\delta \bar{y}_{|\bar{y}|}}{\delta s_{|\bar{y}|}} \cdots \frac{\delta h_1}{\delta x_1} \tag{2.29}$$

At test time, we want to find the most likely output sequence given an input sequence:

$$\hat{y} = \arg\max_{y \in \mathbf{Y}(x)} \pi_w(y|x). \tag{2.30}$$

But for most problems, the space $\mathbf{Y}(x)$ is too large to be searched exhaustively. A simple solution is to greedily generate the most likely token at each time step. Alternatively, one can employ a beam search or sample a token according to the model distribution.

Instead of using Equation 2.28 to update the parameters w, we use different loss functions in Chapters 6 and 8 that do not require the existence of gold target sequences, but instead incorporate feedback from external sources, such as downstream tasks or human users.

Part I.

A Natural Language Interface to OpenStreetMap

Chapter 3

Question-Answering Task

The geographical database OpenStreetMap (OSM) contains over 3.4 billions GPS coordinates with associated information for points of interest all over the world. But a significant portion of this information cannot be accessed by non-expert users with the currently available search functions. Simple string-matching methods are not sufficient to extract complex information. For example, for the question "*Where are 3 star hotels in Paris*" or "*3 star hotels Paris*", the main search tool[6] returns no results, even though corresponding database objects exist. To be able to find them, it is necessary to issue a complex database query using the Overpass[7] query language and this is infeasible for every day users. We want to offer an alternative solution, where non-expert users can query the database for complex object requirements using natural language.

For a natural language interface to OSM, two components are required. First, we define a question-answering task and collect a corpus where natural language questions are paired with parses that can be executed against the database to return the correct, gold answer to the posed question. Second, we need a semantic parser that learns how to map natural language questions to parses. In this chapter, we setup the question-answering task and collect appropriate data. In the following chapter, we turn to training various semantic parsers on the basis of the collected corpora.

The first step in formulating the question-answering task, is the definition of an appropriate Machine Readable Language (MRL) for the parses. A good basis is the Overpass query language, but it is not fine-grained enough for our purpose. Thus, we introduce additional operators and define a MRL that wraps around the Overpass query language. Later, semantic parsers will learn to map natural language questions to parses written in the defined MRL.

In a second step, supervised data needs to be collected. We have the option to either collect questions and corresponding parses or questions and corresponding gold answers. Many recent approaches advocate the collection of question-answer pairs (Berant et al. (2013); Iyyer et al. (2014); Yang et al. (2015); Dunn et al. (2017a), *inter alia*) because it is easier to collect answers than complex parses. However, this is not the case in our domain. For example, to answer the question "*How many hotels are there in Paris?*", one would have to

[6] `http://www.openstreetmap.org`, 1st September 2018
[7] `https://wiki.openstreetmap.org/wiki/Overpass_API`, 1st September 2018

count 951 hotels. It is highly doubtful that this task can be accomplished without error and even if so, it would take an unreasonable amount of time. Furthermore, most questions cannot be answered without resorting to Overpass queries, as is for example the case for the above mentioned example: "*Where are 3 star hotels in Paris*". Finally, answers can change as the database is updated. By collecting parses, the gold answer can always be obtained for the up-to-date database. In addition to this, question-parse pairs make learning easier than question-answer pairs because question-answer pairs only provide indirect, weak supervision (Yih et al., 2016).

We assemble 2,380 question-parse pairs which results in the NLMaps corpus. The corpus is the first publicly available corpus for semantic parsing in the OSM domain. Boye et al. (2014) previously employed the OSM database to build a dialogue system for pedestrian routing, but no resources were provided. We compare our corpus to two other available question-parse corpora. The Geoquery corpus (Wong and Mooney, 2006; Kate et al., 2005) contains questions about a small number of geographical points in the United Sates. It is compositionally and syntactically rich but has a restricted vocabulary due to its closed domain. Free917 (Cai and Yates, 2013) is a corpus operating on the open domain database Freebase.[8] Due to the open domain, it has a significantly larger vocabulary, but questions are structurally simpler. Our corpus offers a middle ground between the two other corpora.

In a next step, NLMaps is further extended, both manually and with automated means, to 28,609 question-parse pairs, leading to the NLMaps v2 corpus. With either corpus, it is possible to train semantic parsers that learn to map questions to parses that can be executed against the OSM database to retrieve answers.

The main contributions of this chapter are as follows:

- Definition of a question-answering task on the basis of the geographical database OpenStreetMap (OSM).
- Design of a Machine Readable Language (MRL) suitable for the question-answering task and a corresponding Context-Free Grammar (CFG) that can determine if a parse is valid under the defined MRL.
- Connecting the MRL to the database query language Overpass and the OSM database.
- Collection of the NLMaps corpus with 2,380 question-parse pairs.
- Extension of NLMaps to 28,609 question-parse pairs, resulting in the corpus NLMaps v2.

The structure of this chapter is as follows: In Section 3.1 we introduce the OSM database and two tools that can query the database. Section 3.2 introduces the Machine Readable Language (MRL) suitable for the question-answering task. With the MRL defined, we describe the creation process of the first question-answering corpus, NLMaps, in Section 3.3. The extension of the corpus, NLMaps v2, is presented in Section 3.4.

[8] https://developers.google.com/freebase/, 1st September 2018

The work presented in this chapter has been previously published in peer-reviewed publications. In particular, Sections 3.1, 3.2 and 3.3 appear in Haas and Riezler (2016) and Section 3.4 in Lawrence and Riezler (2018).

3.1. The Geographical Database OpenStreetMap (OSM)

OpenStreetMap (OSM)[9] is an open-source project where volunteers can contribute geographical knowledge of points of interest in the world to a global database. It counts over 3.4 billions GPS coordinates submitted by more than 2 millions users.[10] As such it provides an unparalleled, publicly available wealth of geographical knowledge.

The database defines three data structures. The most basic structure is a "*node*". Next to the GPS coordinate, every node is given an automatically assigned node ID and, optionally, an elevation. Defining an ordered list of nodes produces a "*way*" structure. Should the first and last element in the list be the same node, the way is also referred to as a "*polygon*"; otherwise it is an "*open way*". Open ways are typically used to outline rivers or streets. Most polygons are considered an "*area*" which can contain any of the other database structures. Should the concept of containment not be desired, the structure is referred to as a "*closed way*" (restricted to circular paths such as a roundabout or barriers). The final structure is a "*relation*" which may group any number of nodes, ways or further relations together to form a cohesive unit. Relations can for example be used to delineate boundaries, such as country outlines, or to indicate that several buildings belong to a common owner, e.g. a company. An area concept can again be defined, creating in this case a "*multi-polygon*".

Any of the three database structures may be enriched with further information. Information points in the database are defined via key-value pairs where one key-value pair is referred to as a tag (e.g. "*tourism=hotel*" is a tag attached to points of interests that are hotels). These key-value pairs can be freely defined by the contributor, although the community has a set of agreed upon common keys and key-value pairs. Which further information the volunteers supply, is their own prerogative. Consequently, the data maybe be incomplete, biased or erroneous.

For some database keys, the corresponding value should be chosen freely. This is for example the case for keys such "*name*", "*website*" or "*phone*". For other keys, it is more suitable to have a predefined list of possible values. For example, the key "*highway*" has a community-defined set of values that include "*motorway*", "*residential*" or "*pedestrian*". These definitions attempt to keep the use of key-value pairs consistent throughout the world. However, the final decision is still left up to the individual contributor. As of December 14th, 2015, there are over 57,000 distinct keys and 76 million distinct tags in the database. Out of all keys, 186 keys are denoted as common features on the OSM Wiki.[11] A statistic overview of the OSM database may be found in Table 3.1.

[9] http://wiki.openstreetmap.org/, 1st September 2018
[10] Statistics taken from http://www.openstreetmap.org/stats/data_stats.html, 14th December 2015
[11] http://wiki.openstreetmap.org/wiki/Map_Features, 1st September 2018

# users	2,389,374
# objects	3,464,399,738
# nodes	3,139,787,926
# ways	320,775,580
# relations	3,836,232
# tags	1,259,132,137
# distinct tags	76,204,309
# distinct keys	57,159

Table 3.1.: Statistics of the OSM database as of December 14th, 2015.

The database may be searched by visiting the website `www.openstreetmap.org` (1st September 2018). The website offers a search box with which one can query the database. The search query is sent to two separate search tools, GeoNames[12] and Nominatim.[13] GeoNames is a database containing over 11 million place names. It annotates the place names with further features that may be of use to disambiguate place names, e.g. by annotating population numbers or alternative spellings. As it contains only place names, it cannot be used to search for other points of interests in the database, such as train stations or individual street names. The second tool, Nominatim, has been specially designed for OSM and is described in greater detail below.

3.1.1. Search Tool: Nominatim

Nominatim uses string-matching techniques to search values belonging to a certain group of keys and hand-crafted features to rank the returned list. The keys whose values are indexed are keys pertaining to names and addresses. Next to the "*name*" key, the group also includes keys such as "*alt_name*", for alternative spellings of a name, or "*int_name*", for international spelling variants. For addresses, the group includes keys such as "*addr:street*" for street names, "*addr:housenumber*" for house numbers or "*addr:city*" for city names. In combination, one can find specific addresses with these fields. For disambiguation, Nominatim defines a hierarchy of importance over the various fields. This is for example useful for areas that have the same name: an area with a city tag will receive a higher score than another area by the same name but only a village tag. Next to this string-based search, the Nominatim tool also supports the search by GPS coordinate as well as performing reverse geo-coding: given a GPS coordinate, reverse geo-coding attempts to synthetically construct an appropriate address for this point.

Furthermore, Nominatim employs a special list of terms[14] that enables users to search for objects with a specific tag. For example, without this list, it would be impossible to search for all restaurants (i.e. objects with the tag "*amenity=restaurant*") in an area. The hand-written list links English keywords to appropriate OSM tags and a simple lookup allows the conversion from one to the other. However, this list has several shortcomings. First, being hand-written

[12] `http://www.geonames.org`, 1st September 2018
[13] `http://nominatim.openstreetmap.org/`, 1st September 2018
[14] `http://wiki.openstreetmap.org/wiki/Nominatim/Special_Phrases/EN`, 1st September 2018

it is incomplete and cannot adapt to future user needs. Second, terms need to be entered exactly as defined on the list, so no natural language variation is possible. For example, while "*Pub in Heidelberg*" returns objects with the tag "*amenity=pub*", the search fails when one simply types "*Pub Heidelberg*". This search only returns objects that have the word "*Pub*" in their name. Third, the list is limited to a small subset of tags, largely consisting of tags for various buildings of public interest. Other, more abstract tags, such as "*wheelchair=yes*", are not covered. But such tags can be of utmost importance to every day users of OSM.

Fourth, the handwritten list offers no composition. For example, while it is possible to search for restaurants, it is not possible to search for more specific restaurants, such as Italian restaurants. Such compositionality at large scale is impossible to achieve with a hand-written list because enumerating all possible combinations would quickly lead to an exploding number of entries and would require an unrealistic amount of time to collect. Exactly this crucial shortcoming can be effectively addressed by employing machine learning to train a statistical model that acts as a semantic parser. Such a parser can learn to generalise and compositionality can be achieved without an exorbitant annotation effort. Assume the following training instances of natural language expression and corresponding OSM are available:

```
Pub in Heidelberg & amenity=pub
smoking Pub in Heidelberg & amenity=pub, smoking=yes
Restaurant in Heidelberg & amenity=restaurant
Hotel in Heidelberg & tourism=hotel.
```

Listing 3.1: Possible training instances for a semantic parser.

From this, a semantic parser can induce that "*smoking Restaurant in Heidelberg*" should map to "*amenity=restaurant, smoking=yes*" and "*smoking Hotel in Heidelberg*" should be converted to "*tourism=hotel, smoking=yes*", even though these exact combinations haven't been observed. This drastically reduces the required annotation effort.

3.1.2. Query Language: Overpass

To find database objects that satisfy specific constraints, two APIs are available to search the OSM databse. The main API of OSM is tailored to ease the editing of data. But the Overpass API offers a comprehensive, programming-like language to filter objects from the database that fulfil any tag criteria specified, including tag compositions. Furthermore, it can take the location of objects into account via an area concept. It is also possible to run a radius search around a specified OSM object. However, writing an Overpass query requires understanding the underlying language and is thus not suitable for non-expert users. But with the powerful features it offers, Overpass is a perfect fit as the basis for the Machine Readable Language (MRL) of the question-answering task. In the following, we review the key operators that the Overpass language offers and which we plan to use for our MRL:

tag filtering. After specifying the database object type (*node*, *way* or *relation*), any number of tags or keys may be specified. Each tag or key has to be surrounded by its own set of square brackets. For example,

```
node["amenity"= "restaurant"]["website"];
```

retrieves nodes with the tag "*amenity= restaurant*" and ensures that the node has a key named "*website*".

union. Wrapping several tag filters in round brackets creates a union of the individually retrieved objects. Thus,

```
(node["amenity"="restaurant"] ["website"];
node["amenity"="bar"]["website"];);
```

returns a set of both restaurants and bars with a "*website*" key. It is not always clear if a restaurant is tagged as a node, way or relation. Thus, to find all restaurants, one should formulate a union of all three database types:

```
(node["amenity"="restaurant"]; way["amenity"="restaurant"];

relation["amenity"="restaurant"];);
```

As a consequence, Overpass queries become quite cumbersome.

area restriction. Often, we want to apply tag filtering to a specific area only, rather than searching the entire database. This can be done by first specifying an area and binding it to a variable. The variable can then be called within the tag filter. The query

```
area["name"="Heidelberg"]->.searchArea;
node["amenity"="restaurant"](area.searchArea);
```

ensures that only areas called "*Heidelberg*" are searched for nodes fulfilling the listed requirement.

radius search. It is possible to search for objects using tag filtering in a radius around some reference point. First, a reference point has to be defined and saved to a variable. Then, the operator "`around`" can be used to search in a specified radius around the reference point for objects with certain tags. For example, to search for restaurants with a "*website*" key in a radius of 5km around a street called "*Kastellweg*", the following Overpass query can be formulated:

```
area["name"="Heidelberg"]["de:place"="city"]->.a;
(node(area.a)["name"="Kastellweg"];
way(area.a)["name"="Kastellweg"];
relation(area.a)["name"="Kastellweg"];)->.b;
(node(around.b:5000)["amenity"="restaurant"]["website"];
way(around.b:5000)["amenity"="restaurant"]["website"];
relation(around.b:5000)["amenity"="restaurant"]["website"];);
out body; >; out skel qt;
```

Listing 3.2: Example for an Overpass query that returns restaurants in the area "*Heidelberg*" and which are within a 5km radius around a street called "*Kastellweg*".

Listing 3.2 demonstrates how complex Overpass queries can quickly become. It contains variables that need to be assigned and re-used correctly and has a stringent definition for placing brackets as well as semi-colons to indicate the end of a command. The MRL for our question-answering task should drastically simplify the queries so that semantic parsers can more easily learn to produce valid parses.

Furthermore, executing an Overpass query returns all available information about OSM objects that satisfy the constraints outlined in the query. Consequently, using Overpass for the question-answering task would mean displaying more information to the user than they requested. For example, assume a user requests all restaurants in an area and the corresponding website links. The correct Overpass query would return all restaurants with website links, but each restaurant entry would be accompanied with all the other information that is also available for that restaurant. The user would have to scan the returned data for the information they actually desired, i.e. the website links, which would be tiresome and time consuming. As an example, Figure 3.1 shows the first database entry found for the query from Listing 3.2 in XML format. Thus, the aim of the new MRL is to simplify Overpass queries and to retrieve more fine-grained information than Overpass can.

The project Overpass TURBO also attempts to make the Overpass query language more accessible to users. It is a web interface that offers a text editor to enter Overpass queries which can be executed. Returned database objects can be viewed either as an XML or JSON document or on a world map with annotated pins. Similar to the hand-written list for Nominatim, the Overpass TURBO 1web interface offers a wizard where a few key terms in natural language are automatically expanded into a corresponding Overpass expression. The underlying abbreviation list of the wizard is useful to save time typing Overpass queries in some instances, but it still requires the use and knowledge of OSM tags and is thus also unsuitable for non-expert users. Additionally, analogous to the compositionality issue of the Nominatim list, this key word list is quickly exhausted and offers no options for complex, compositional queries. Such queries have to be typed out fully in the Overpass language. Lastly, the Overpass TURBO website offers the option to save Overpass queries. The collected log[15] of these queries serves as the basis for the creation of our question-parse corpus.

[15]Thanks to Martin Raifer for freely sharing this resource and to the creator of the Overpass API, Roland Olbricht, for pointing this author in the right direction.

```
<node id="137863096" lat="49.4129429" lon="8.7045098">
  <tag k="addr:city" v="Heidelberg"/>
  <tag k="addr:housenumber" v="3"/>
  <tag k="addr:postcode" v="69117"/>
  <tag k="addr:street" v="Marstallhof"/>
  <tag k="amenity" v="restaurant"/>
  <tag k="drink:club-mate" v="served"/>
  <tag k="name" v="Zeughaus-Mensa"/>
  <tag k="opening_hours" v="Mo-Sa 11:30-22:00"/>
  <tag k="phone" v="+49 6221 54 26 44"/>
  <tag k="toilets:wheelchair" v="yes"/>
  <tag k="website" v="http://www.studentenwerk-heidelberg.de"/>
  <tag k="wheelchair" v="yes"/>
</node>
```

Figure 3.1.: First database entry returned for the Overpass query from Listing 3.2 in XML format. If a user is only interested in the website address, they have to skim over all tags for the database entry to find the relevant information.

3.2. Definition of a Machine Readable Language (MRL)

The MRL for the question-answering task should fulfil a few requirements. First, it should be easy to recover Overpass queries but they should be shortened and simplified. Second, once an Overpass query is executed, the MRL should offer additional filtering operators to extract more fine-grained information. Third, it should be possible to judge whether a parse is valid under the defined MRL or not. Fourth and finally, it should be possible to tokenise the MRL so that sequence-to-sequence methods can easily be applied.

We choose to define our MRL as a variable free language that consists of a pre-defined set of operators. Some operators can take OSM tags or keys as values, others closely resemble Overpass operators. Each operator can take a set of pre-defined other operators as its arguments and the scope of an operator is delimited using brackets. Concretely, we define the following operators:

- **`query.`** This operator indicates the start of a MRL parse. For the simplest parse, it is required to take at least the `nwr` and `qtype` operators defined below.
- **`keyval.`** The `keyval` operator can take any OSM tag as its argument. Several comma-separated keyval operators ensure that the database objects fulfil all listed requirements. This easily converts to the Overpass tag filtering (see Section 3.1.2).
- **`nwr.`** The name `nwr` indicates the three data types of OSM, namely "*node*", "*way*" and "*relation*". This operator exclusively takes `keyval` operators as its arguments. There needs to be at least one such operator, but there may be arbitrarily many. As explained in Section 3.1.2, it is never clear which data type a desired database object has. Thus the `nwr` operator, when converted into Overpass language, automatically copies its `keyval` arguments, once each for node, way and relation. These three are

then grouped into a union so that all data types fulfilling the tag requirement are returned. For example,

```
nwr(keyval('amenity','restaurant'))
```

is automatically expanded to

```
(node["amenity"="restaurant"]; way["amenity"="restaurant"];

relation["amenity"="restaurant"];);.
```

- **`area.`** Similar to `nwr`, this operator takes at least one `keyval` as its argument. It directly corresponds to the area concept of the Overpass API (see Section 3.1.2).

- **`around.`** In the case of a radius search, the `around` operator is needed. This directly translates to the around operator of the Overpass API (see Section 3.1.2). The `around` operator allows us to process fuzzy natural language terms such as "*nearby*", "*close*" or "*within walking distance*". To do this, the `around` operator takes 3 further operators to define the search correctly. First, a reference point needs to be defined. Second, the objects of interest around the reference point. Third, the size of radius to be searched. The operators for the former two are defined in the subsequent two paragraphs.

  The radius size can be one of four pre-defined variables, namely `WALKING_DIST=1km`, `DIST_INTOWN=5km`, `DIST_OUTTOWN=20km` and `DIST_DAYTRIP=80km`. Fuzzy natural language terms are mapped to these four types. Consider the questions "*Where are restaurants close to Heidelberg?*" and "*Where are airports close to Heidelberg?*". The two questions only differ in the words "*restaurant*" and "*airport*". However, different radii are implied in the two questions via world knowledge: most people would be willing to undertake a 80km trip to reach an airport, but this is typically not the case for a visit to a restaurant. Minock and Mollevik (2013) have also previously investigated this concept. Annotating this concept in the MRL parses via the radius operator, allows a later semantic parser to learn to distinguish between these scenarios.

- **`center.`** This operator is an argument of `around` and can itself take the operators `area` or `nwr` as arguments. The specified database object serves as the reference point of a radius search.

- **`search.`** Similar to the `center` operator, the `search` operator can also optionally take an `area` operator and must take a `nwr` operator. It defines which objects around the reference point should be returned in a radius search.

- **`qtype.`** To be able to extract more fine-grained information from database entries returned by an Overpass query, we need to be able to define which information should be extracted. All specifications relevant to this are placed as arguments of the `qtype` operator and are detailed in the following paragraphs.

- **`latlong.`** Returns the GPS coordinates of all database entries returned by an Overpass query.

- **`count.`** Returns the number of objects in the set of database objects returned by an Overpass query.

- **`findkey`.** Takes any OSM key as its argument (e.g. "*website*"). The corresponding value (e.g. the concrete website link) is extracted from the database entries returned by an Overpass query.

- **`least`.** Returns either true or false. True, if there are at least x number of objects in the set of database objects returned by an Overpass query. x can be any positive integer number and is supplied as the argument of the `topx` operator defined below.

- **`topx`.** This operator is required for the `least` operator. It can also be used in conjunction with `latlong` and `findkey`. In that case it returns only the first x elements in the set of database objects. Furthermore, it can be used as an argument of `around` where it indicates that only the x *closest* elements to the reference point should be displayed.

- **`dist`.** The `dist` operator can calculate the distance between two database objects. For ways and relations, the mean coordinate of all participating GPS coordinates is calculated. As an argument, the operator can simply take a `query` that contains the `around` operator. Then the distance between the object defined via `center` and the object defined by `search` is returned. If the `search` operator returns several database objects, the closest distance is returned.

  Alternatively, the `dist` operator can take two separate, fully formed `query` operators, both of which do not contain the `around` key word. The `dist` operator also takes a further argument, namely the operator `unit`. If "*km*" is the argument of `unit`, then the calculated distance is presented in kilometres. "*mi*" displays the distance in miles. A further, optional argument for `dist` is the `for` operator. This operator takes either "*car*" (e.g. "*Is a car needed…*") or "*walk*" (e.g. "*Is it possible to walk…*") as its argument. In this case, the distance value is measured against the variable `WALKING_DIST`. For "*walk*", the reply is "*yes*" if the distance is below this value and "*no*" otherwise. For "*car*" the answers are reversed.

- **`north / east / south / west`.** These four operators form another group that helps to map fuzzy natural language to machine understandable concepts. It is used in questions such as "*Where are restaurants in the north of Heidelberg?*". These four cardinal direction operators can wrap around any `area` operator combined with an `nwr` operator. It ensures that the elements returned by the Overpass query are further filtered to verify that the remaining elements are in the specified cardinal direction of the area.

- **`nodup`.** This operator removes duplicates from a list of answers. This is for example desired in a questions such as "*Which cuisines are there in Heidelberg?*". Here, one does not want to have a cuisine repeated as many types as there are restaurants with that tag in the city, but rather a set of unique entries.

- **`*`.** The wildcard operator `*`, can be used instead of a value in a key-value pair. It indicates that any value is acceptable. This directly maps to the wildcard principle that Overpass offers (see Section 3.1.2). This is required to successfully map certain information needs to the particularities of the OSM database. For example, the correct MRL parse for the question "*How many historic sites can be found in Nantes*", should include "`keyval('historic',*)`". With the wildcard operator it is possible to search for any historic place, be it a tower, castle etc.

- **`and`.** The `and` operator may take two `keyval` operators as its argument. This is for example used for questions such as "*Where is the closest bank and butcher in Heidelberg?*". In this case, the `and` operator is

  ```
  and(keyval('amenity', 'bank'),keyval('shop','butcher'))
  ```

  and from this two Overpass queries are automatically generated. The first query contains "`keyval('amenity','bank')`" and the second "`keyval('shop','butcher')`". After the two answers are returned and post-processed, they are concatenated with "*and*". Furthermore, this operator can be used as an argument for `findkey` to ensure that two pieces of information are retrieved. For example the correct MRL parse for "*Give me the name and website of restaurants in Heidelberg*" would contain "`findkey(and (name,website))`".

- **`or`.** This operator models the "exclusive or" and is for example needed in a question such as "*Give me the closest bar or restaurant.*". In the resulting Overpass query, this is first represented using a union (see Section 3.1.2). Once all elements are retrieved, a post-processing step extracts the closest element in the set, be it a bar or a restaurant.

Using the presented operators, we can formulate the Overpass query from Listing 3.2 in the condensed form of our MRL. Furthermore, we can extract more fine-grained information via the tag `qtype`. The parse in Listing 3.3 retrieves the websites of restaurants in a radius of 5km around the street Kastellweg in Heidelberg:

```
query(around(center(area(keyval('name','Heidelberg')),
nwr(keyval('name','Kastellweg'))),
search(nwr(keyval('amenity','restaurant'))),
maxdist(5000)),qtype(findkey('website')))
```

Listing 3.3: Example for a NLmaps parse that returns the websites of restaurants in a radius of 5km around the street Kastellweg in Heidelberg.

With 173 characters, the parse in the newly defined MRL is significantly shorter compared to the corresponding Overpass query which contains 352 characters. Yet a simple, deterministic script can easily convert the relevant sections into fully formed Overpass queries. Furthermore, the additional operators, such as `north`, have been implemented alongside Overpass. Additionally, with the `qtype` operator, more fine-grained information can be extracted.

All possible allowed combinations of operators can be defined unambiguously in a Context-Free Grammar (CFG). A condensed form of the CFG can be found in Appendix A. The CFG can be used to ascertain the validity of any parse produced by a semantic parser. Before the validity of a parse is checked, all OSM keys and values are replaced by the placeholders "*keyvariable*" and "*valuevariable*", respectively. This way, the CFG remains concise and keys and values that are added at a later date do not require an update to the CFG.

By definition, the MRL is structured so that any parse can be presented as a tree. For an example, see Figure 3.2. Taking a pre-order traversal of the tree, allows us to easily tokenise a parse. Each node is annotated with a special marker ("@"), followed by the number of

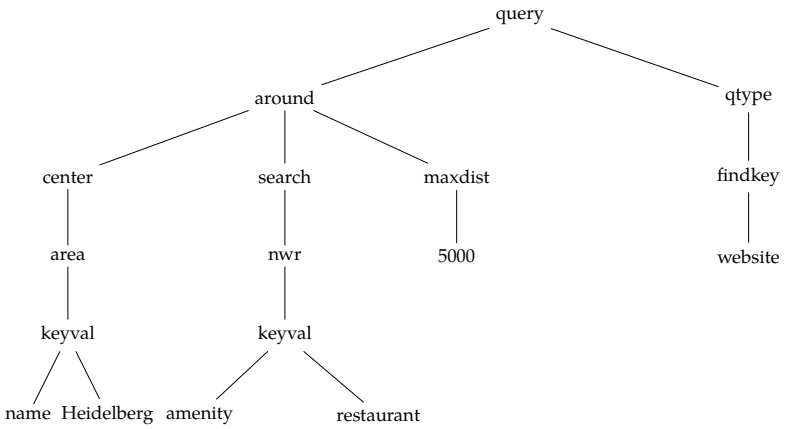


Figure 3.2.: Example of a NLmaps parse represented as a tree for the parse from Listing 3.3.

children (i.e. arguments) it has. The result is a linearised parse where individual tokens are separated by a whitespace. This representation is introduced by Andreas et al. (2013) for the GeoQuery corpus (Zelle and Mooney, 1996). With the conversion to individual tokens, the question-answering task can be treated as a sequence-to-sequence problem. The linearised parse from Listing 3.3 is with 168 non-whitespace tokens of similar length and can be found in Listing 3.4:

```
query@2 around@3 center@2 area@1 keyval@2 name@0 Heidelberg@s nwr@1
    keyval@2 name@0 Kastellweg@s search@1 nwr@1 keyval@2 amenity@0
    restaurant@s maxdist@1 5000@0 qtype@1 findkey@1 website@s
```

Listing 3.4: Example for a NLmaps linearised parse that returns the websites of restaurants in a radius of 5km around the street Kastellweg in Heidelberg.

The requirements we defined at the beginning of the section are all fulfilled. First, the MRL wraps around Overpass and simplifies the Overpass language. Second, objects returned by an Overpass query can be filtered further to extract more fine-grained information. Third, the CFG offers the option to verify whether a parse is valid under the MRL. Fourth, with the pre-order traversal it is possible to tokenise parses so that sequence-to-sequence methods can be applied. To connect the newly defined MRL to the OSM database, we implement corresponding scripts and extend the Overpass project accordingly. It is now possible to pass a parse written in the new MRL to a script which executes the parse against the database and returns an answer. With the MRL fully implemented, we now turn our attention to collecting question-parse pairs.

3.3. Corpus Creation: NLmaps

3.3.1. Question-Parse Pair Generation

The basis for our question-parse corpus, called NLmaps, is the log of saved Overpass queries from the Overpass TURBO website (see Section 3.1.2). Each Overpass query is manually converted to an equivalent parse in the NLmaps MRL. Additionally a fitting `qtype` operator is defined to capture more fine-grained information. In some cases, users of Overpass TURBO simply executed their query for the map cut-out that they had currently on their screen. The information of the map cut-out was not saved and thus another area has to be selected. In such cases, three separate queries are created: one each for the city areas of Heidelberg, Edinburgh and Paris. Based on the resulting final query, an appropriate English question is formulated.

To ensure that the natural language interface can handle important OSM tags, we additionally consulted the list of most common tags, as defined in the OSM Wiki.[16] Based on this list, further question-parse pairs where added manually. In total, 2,380 question-parse pairs were assembled by this author. Together they constitute the NLmaps corpus. A few example questions can be found in Listing 3.5.

```
What is the closest bank with ATMs from the Palace of Holyroodhouse in
    Edinburgh?
Where are the closest bank and the closest pharmacy from the Rue
    Lauriston in Paris?
Give me the name and location of all tourist related activities that can
    be accessed with a wheelchair in Heidelberg!
Where is the closest Indian or Asian restaurant from the cinema Le
    Cinaxe in Paris?
What are the names of cinemas that are within walking distance from High
    Street in Edinburgh?
How many schools in Edinburgh have a bus stop less than 200 meters away?
What is the name of the closest museum or art centre from Notre Dame in
    Paris?
How many historic sites are in the east of Nantes?
Where is the closest restaurant or bar from the Hawes Pier in Edinburgh?
Are there any caves in Osterode and if so how many?
```

Listing 3.5: Example questions from the NLmaps corpus.

3.3.2. Comparison to other Question-Answer Corpora

Corpora for different question-answering tasks have been created in recent years. A standard corpus in semantic parsing is the GeoQuery corpus (Zelle and Mooney, 1996). It contains questions pertaining to the geography of the United States and the topic is thus loosely

[16] http://wiki.openstreetmap.org/wiki/Map_Features, 1st September 2018

	NLmaps	Geoquery	Free917
# sent.	2,380	880	917
tokens	25,906	6,660	6,785
types	1,002	296	2,038
avg. sent. length	10.88	7.57	7.4
avg. NT per sent.	21	16	16
avg. types per sent.	0.42	0.34	2.22
avg. singleton per sent.	0.1	0.1	1.52
FRES	82.18	86.61	83.77

Table 3.2.: Corpus statistics of the question-answering corpora NLmaps, Geoquery and Free917. NT stands for non-terminal and FRES is the Flesch Reading Ease Score (Flesch, 1974).

related to the topic of NLmaps. However, the scope of GeoQuery is limited to a small number of landmarks in the United States and a small set of possible features that can be queried, such as the height of prominent mountains or the population density of states. For many questions it is unrealistic that a human user would pose them in praxis, e.g. "*How many states border the state that borders the most states?*". Its creation was motivated to test a semantic parser's capability rather than to emulate questions human users might ask.

A further corpus is the Free917 corpus (Cai and Yates, 2013), which contains questions that can be answered with the open-domain database Freebase. The corpus contains questions from 81 different domains and were formulated by native speakers. Questions are often simple factoid question and typically contain named entities, for example "*when was the iphone introduced*". We compare our NLmaps corpus to both of these corpora.

Other corpora do not provide question-parse pairs and supply question-answer pairs instead, such as the corpora Webquestions (Berant et al., 2013), WikiQA (Yang et al., 2015), Factoid Questions (Iyyer et al., 2014), *inter alia*. This introduces the added difficult of having to find parses that execute to the correct answers which can cause the quality of the parser to suffer (Wang et al., 2015; Pasupat and Liang, 2015). Other corpora for question-answering tasks are TownInfo (Williams and Young, 2007; Mairesse et al., 2009), Regexp824 (Kushman and Barzilay, 2013), WikiTableQuestions (Pasupat and Liang, 2015), *inter alia*. Corpora published after NLmaps include TriviaQA (Joshi et al., 2017), SearchQA (Dunn et al., 2017b), *inter alia*.

Corpus statistics comparing the corpora NLmaps, Geoquery and Free917, normalised by the number of sentences, can be found in Figure 3.2. With 2,380 question-parse pairs, the NLmaps corpus is twice the size of the other two corpora. NLmaps has a higher type count than Geoquery by a factor of 3, but has only half the amount compared to Free917. This suggests that NLmaps has a higher lexical variety compared to Geoquery but less compared to Free917. This is unsurprising as NLmaps is restricted to the geographical domain whereas the Free917 database covers more domains and contains many named entities.

In terms of syntactic complexity, the NLmaps corpus is more complex than the other two corpora. First, a question from the NLmaps corpus contains on average 3 more words. Second, to

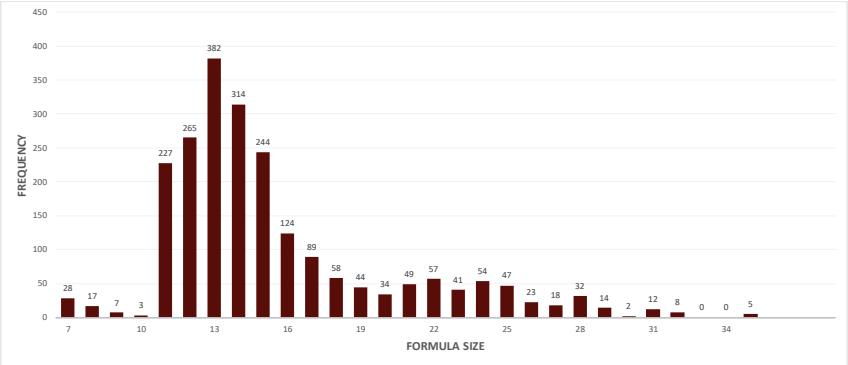


Figure 3.3.: Histogram of the lengths of correct parses in the NLmaps corpus.

validate this observation, we parse the natural language questions with the Stanford Parser (Klein and Manning, 2003) and count the number of non-terminal nodes in a sentence. It requires on average 5 more non-terminals to parse a NLmaps sentence compared to the other two corpora. This confirms what the average sentence length suggested, that NLmaps is syntactically more complex.

With an average of 2.22 new words every sentence, Free917 contains the highest number of types per sentence. This can be explained by the large number of named entities, which also cause to the highest number of singletons per sentence. Finally, we report the Flesch Reading Ease Score (FRES) (Flesch, 1974). It is a score that measures the difficulty of a text and was invented to identify suitably difficult reading material for students learning to read. The lower the score, the more difficult the given text. NLmaps receives the lowest score and can thus be considered more complex than the other two corpora with regards to this metric.

On the MRL side, we report a histogram of the length of the correct parse for NLmaps (see Figure 3.3). The vast majority of questions require a parse of at least length 10. Most parses contain 10 to 20 tokens. Longer parses gradually become more rare, ending with a set of 5 parses of length 36. With an average of 15.5 tokens per parse, parses are longer than the natural language questions. This indicates that the questions are indeed complex and elaborate database operations are required to answer them. Overall, the NLmaps corpus offers a challenging question-answering task for semantic parsers while also considering the end goal of building a natural language interface to OSM.

3.4. Corpus Extension: NLmaps v2

To improve the coverage of the natural language interface, we consider two extensions to the NLmaps corpus. First, we modify the natural language questions of NLmaps to more closely resemble typical search engine queries. Second, we use the special list of terms[17] from Nominatim to automatically generate further question-parse pairs. The list contains

[17] http://wiki.openstreetmap.org/wiki/Nominatim/Special_Phrases/EN, 19th April 2018

English phrases and corresponding OSM tags. With our automatic extension, we can increase both the number of covered OSM tags and introduce a larger variety of named entities. Both extensions are concatenated with the original corpus, leading to NLmaps v2 with 28,609 question-parse pairs. Additionally, we provide a set of question-parse pairs were all named entities are masked by placeholder symbols. This enables the option of handling the problem of named entities separately from the semantic parsing.

3.4.1. Search Engine Style Extension

First tests with a prototype of the natural language user interface showed that human users tend to use short, search engine style questions where all non-essential words are omitted. This is a contrast to the longer, grammatically correct English questions found in the NLmaps corpus. Thus, we modify the original set of training questions to more closely resemble such search engine style queries. Furthermore, phrases indicating the answer type were removed and the answer type on the MRL side is defaulted to `latlong`. Next, the resulting corpus was duplicated and the short phrases "*where*", "*how many*", "*is there*" and "*name*" where prepended to indicate the answer types "`latlong`", "`count`", "`least(topx(1))`" and "`findkey(name)`", respectively. Modified counterparts to the example questions from Listing 3.5, can be seen in Listing 3.6:

```
closest bank with ATM's from Palace of Holyroodhouse in Edinburgh
closest bank and the closest pharmacy from the Rue Lauriston in Paris
tourist related activities wheelchair in Heidelberg
closest Indian or Asian restaurant from the cinema Le Cinaxe in Paris
cinemas within walking distance from High Street in Edinburgh
schools in Edinburgh bus stop less than 200 meters away
closest museum or art centre from Notre Dame in Paris
historic sites in the east of Nantes
closest restaurant or bar from Hawes Pier in Edinburgh
cave in Osterode
```

Listing 3.6: Modified example questions from the NLmaps corpus to resemble search engine style queries.

3.4.2. Automatic Extension

A list[18] of special phrases with English expressions and corresponding OSM tags has been compiled for Nominatim. For example it links "*Cycle Renting*" to "*amenity=bicycle_rental*" or "*Petrol Stations*" to "*amenity=fuel*". The table also includes both singular and plural natural language expressions. Furthermore, it offers each expression in conjunction with "*in*" and "*near*", where the following word is expected to be an appropriate area name. An example of all table entries for the first tag, "*aeroway=aerodrome*", can be found in Table 3.3. In total, the list contains 2,830 entries which reduce to 421 unique tags. We use this list to automatically

[18] `http://wiki.openstreetmap.org/wiki/Nominatim/Special_Phrases/EN`, 1st September 2018

#	Word	Key	Value	Operator	Plural
1	Airport	aeroway	aerodrome	-	N
2	Airports	aeroway	aerodrome	-	Y
3	Airport in	aeroway	aerodrome	in	N
4	Airports in	aeroway	aerodrome	in	Y
5	Airport near	aeroway	aerodrome	near	N
6	Airports near	aeroway	aerodrome	near	Y

Table 3.3.: All entries in the Nominatim list for the tag "*aeroway=aerodrome*".

generate new question-parse pairs. This increases the number of OSM tags available in the natural language interface. Additionally, it introduces more named entities, both in terms of new area names and new points of interests.

Depending on the columns *Operator* and *Plural* in the Nominatim table (see also Table 3.3), different NLmaps MRL parses and natural language skeletons can be automatically filled with the values from the columns titled *Word*, *Key* and *Value*. In the following, we introduce the different possible structures using the airport example from Table 3.3. Each structure contains several placeholders. On both English and MRL side, every structure has the placeholder `$LOCATION` for which any area name can be substituted. Database queries that are not restricted to an area would return a world-wide list of objects which would be very large for most tags. Thus rows #1 and #2 from Table 3.3 are not used for the automatic extension.

With the entries of rows # 3 and #4 of Table 3.3, a basic parse can be built. Next to the `$LOCATION` placeholder for an area name, there is the additional placeholder `$QTYPE`. It expects arguments for the `qtype` operator on the MRL side and an appropriate English phrase on the natural language side. Listing 3.7 shows the skeletons constructed for both question and parse from rows # 3 and #4, respectively:

```
$QTYPE Airport in $LOCATION
query(area(keyval('name','$LOCATION')),
    nwr(keyval('aeroway','aerodrome')),qtype($QTYPE))

$QTYPE Airports in $LOCATION
query(area(keyval('name','$LOCATION')),
    nwr(keyval('aeroway','aerodrome')),qtype($QTYPE))
```

Listing 3.7: Automatically generated English question and MRL parse with placeholders for rows #3 and #4 of Table 3.3.

Employing row #6 from Table 3.3 we can generate two separate question-parse pairs. The first structure assumes that the objects of interest can be found near an area. The second structure is more complex and assumes that the objects of interests should be found around a specific point of interest in an area, leading to the placeholder `$POI`. On both MRL and natural language side, the `$POI` placeholder is to be filled with a named entity. The `$LOCATION` and `$QTYPE` placeholders are present in both cases. For both structures, we require

the `around` operator and a corresponding radius size. The radius size has a placeholder named `$DIST`. The two structures built from row #6 can be found in Listing 3.8:

```
$QTYPE Airports near $LOCATION $DIST
query(around(center(nwr(keyval('name','$LOCATION'))),
    search(nwr(keyval('aeroway','aerodrome'))),
    maxdist($DIST)),qtype($QTYPE))

$QTYPE Airports near $POI in $LOCATION $DIST
query(around(center(area(keyval('name','$LOCATION')),
    nwr(keyval('name','$POI'))),
    search(nwr(keyval('aeroway','aerodrome'))),
    maxdist($DIST)),qtype($QTYPE))
```

Listing 3.8: Automatically generated English questions and corresponding MRL parses with placeholders for row #6 of Table 3.3.

Finally, we use row #5 from Table 3.3 to generate another two question-parse pair structures. Unlike the plural on the natural language side in row #6, the singular in row #5 implies that only the closest object with the correct tag should be returned. For the first structure, the closest object to the `$LOCATION` placeholder should be returned and for the second structure it should be the closest object to the `$POI` placeholder. Analogously to the structures for row #6, the placeholders `$QTYPE` and `$DIST` are also present. The two structure built from row #5 can be found in Listing 3.9:

```
$QTYPE closest Airport from $LOCATION $DIST
query(around(center(nwr(keyval('name','$LOCATION'))),
    search(nwr(keyval('aeroway','aerodrome'))),
    maxdist($DIST),topx(1)),qtype($QTYPE))

$QTYPE closest Airport from $POI in $LOCATION $DIST
query(around(center(area(keyval('name','$LOCATION')),
    nwr(keyval('name','$POI'))),
    search(nwr(keyval('aeroway','aerodrome'))),
    maxdist($DIST),topx(1)),qtype($QTYPE))
```

Listing 3.9: Automatically generated English questions and corresponding MRL parses with placeholders for row #5 of Table 3.3.

Once the entire Nominatim list is converted into this set of structures, the placeholders need to be filled. Area names for the `$LOCATION` placeholder are sampled uniformly from a list of large cities of Germany, France and the United Kingdom. For skeletons containing `$POI` placeholders, city-specific lists are created from which we also sample uniformly. Each city-specific list contains the values from all objects with a "*name*" key for the specific city. The sample is drawn from the city-specific list that corresponds to the previously sampled city.

The `qtype` operator is uniformly sampled from the set {`count`, `latlong`, `least(topx(1))`, `findkey($KEY)`}. In the case of `findkey($KEY)`, the placeholder `$KEY` is filled with

	NLmaps	NLmaps v2
# sent.	2,380	28,609
tokens	25,906	202,088
types	1,002	8,710
avg. sent. length	10.88	7.06
avg. NT per sent.	21	16.63
avg. types per sent.	0.42	0.3
avg. singleton per sent.	0.1	0.2

Table 3.4.: Corpus statistics of the question-answering corpora NLmaps and NLmaps v2. NT is short for non-terminal.

a key that is sampled from the list of keys of the corresponding object. Such a list is automatically created for each object via a database query that returned all key names for the object in question. On the natural language side the following terms took the place of `$QTYPE`, respectively: *How many*, *Where*, *Is there* and `$KEY`.

For the placeholder `$DIST`, we sample uniformly from the set {`WALKING_DIST`, `DIST_INTOWN`, `DIST_OUTTOWN`}. `DAYTRIP_DIST` is omitted from this set because for many common tags the resulting answer set would be much too large for this radius of 80 kilometres. Unfortunately, generating gold answers showed that for many parses with `DIST_OUTTOWN`, the resulting answer set is also too large, leading to an out of memory error during execution. To solve this problem, instances of `DIST_OUTTOWN` are converted to `DIST_INTOWN`. On the natural language side, `DIST_INTOWN` is assumed to be the default type if there is no explicit marker for another distance. If `WALKING_DIST` is chosen, then the phrase "*in walking distance*" appears in the English question.

The sampling process is repeated twice and thus there are two occurrences of each entry with varying elements in the placeholder positions. Next to the set of question-parse pairs with filled in placeholders, we also maintain an identically shuffled set where the placeholders `$POI` and `$LOCATION` are still present. Using that version, it is possible to treat the recognition of named entities and the semantic parsing of a sentence as two separate problems. Such a setup could lead to an overall task improvement and we investigate this in Section 4.3 of the next chapter. The `$POI` and `$LOCATION` masking does not exists for the remainder of the corpus. But the masking can easily be created by automatically extracting the values in the correct positions on the MRL side. Once the correct phrase has been retrieved, the same phrase can be masked on the English side as well.

3.4.3. Comparison to NLmaps

Figure 3.4 shows the corpus statistics of the first and the second version of the NLmaps corpus. With 28,609 question-parse pairs, NLmaps v2 is an order of magnitude larger than the original version. The vocabulary size of the new corpus is nearly 9 times larger. However, due to the nature of the extension, first simplifying the English questions and second automatically creating question-parse pairs from the simpler structures on the Nominatim

list, it is not surprising that the average sentence length, the average non-terminal per sentence and the types per sentence are lower in NLmaps v2 compared to the first version. But because NLmaps v2 contains all instances from the original NLmaps corpus, we know that there are still complex questions in the corpus that are challenging. The additional questions serve to introduce more natural language variability and more OSM tags. The increase of average singletons can be explained by the named entities introduced with the automatic extension.

In Section 4.3.3 of the next chapter we show that training a parser with the training set of NLmaps v2 leads to better performance on the test set of NLmaps compared to a parser trained solely on the training data of NLmaps. This confirms that the automatic nature of the extension is viable.

Conclusion

The OpenStreetMap (OSM) database is a valuable and freely available geographic resource. Currently available tools to search the database that can be operated by non-expert users, are based on string-matching techniques and cannot be used to extract database objects with more complex specifications from the database. Tools that can do this, require expert knowledge of the OSM database and its query language Overpass. We aim to build a natural language interface for non-expert users where natural language questions are transformed into parses that can be executed against the database.

In a first step, we set up an appropriate question-answering task for the interface. We defined a new MRL suitable for the task and which wraps around the already existing Overpass language. The new MRL vastly simplifies the Overpass language, which required the correct use of variables and bracket structures. Additionally, the MRL can operate on returned database objects to return more fine-grained information than the Overpass language can. Lastly, we connected the resulting MRL with the OSM database so that a parse in the newly defined MRL can be executed against the database with a simple script call.

With the MRL defined, we collected question-parse pairs for a corpus, called NLmaps. The collection is based on logs of Overpass queries from the website Overpass TURBO. The queries were converted and extended into NLmaps parses and appropriate English questions were added. Furthermore, we made sure to include the most common OSM tags. We compared the resulting corpus of 2,380 question-parse pairs to two other corpora, Geoquery and Free917, and were able to show that NLmaps corpus offers a more challenging task.

Two separate extensions allowed us to create further question-parse pairs, resulting in the corpus NLmaps v2 which totals 28,609 question-parse pairs. The first extension modified questions of the original corpus to more closely resemble typical search engine queries, where any non-essential words are omitted. The second extension utilises a freely available list that matches common natural language expressions to OSM tags and automatically extends this list into questions and corresponding parses. With the question-answering task defined and appropriate data collected, we now turn in a second step to building semantic parsers which learn a mapping from question to parse.

Chapter 4

Semantic Parsers

For a natural language interface to OSM, we require a semantic parser that can map natural language questions to parses, which in turn can be executed against the database to obtain an answer. The semantic parser can be trained using either version of the NLmaps corpus described in the previous chapter. With a semantic parser, we can formulate a complete pipeline for the question-answering task, leading from a natural language question as input to the answer that can be presented to the user. A visual overview of the complete pipeline can be found in Figure 4.1. This pipeline process can be connected to a graphical user interface, which allows us to present a first, fully functional natural language interface to OSM.

We treat semantic parsing as a sequence-to-sequence problem where the input is a natural language string that is transformed into a corresponding parse. The NLmaps MRL is specifically designed so that a pre-order traversal of its underlying tree structure can transform the parse into a linearised structure with individual tokens. These tokens can be seen as words in the NLmaps MRL, which allows us to view semantic parsing as a machine translation task where we learn to translate from a natural language into a MRL. This approach has been considered previously by Wong and Mooney (2006) and Andreas et al. (2013). In Andreas et al. (2013), the phrase-based SMT framework Moses (Koehn et al., 2007) is modified for the corpora Geoquery (Zelle and Mooney, 1996) and ATIS (Dahl et al., 1995), achieving state-of-the-art results for Geoquery at the time. Similarly, we modify and extend the hierarchical phrase-based SMT framework Cdec (Dyer et al., 2010) for the NLmaps corpus. Our best model can achieve an F1 score of 77.3% on the NLmaps test set.

With a first successful semantic parser available, we next turn our attention to creating a web-based user interface to provide a platform for users to query the OSM database using natural language. We offer several extensions to improve user friendliness. First, text-based answers are linked to corresponding markers on an interactive map. Second, we supply several options for area detection to restrict the search to areas of interest. Third, we offer Nominatim[19] as a back-off system should our semantic parser fail. Fourth, we further extend our semantic parsing model with two additional components and report empirical results. Fifth and lastly, we give users the opportunity to submit feedback once a question has been processed. To the best of our knowledge, this is the first natural language interface to OSM

[19] http://wiki.openstreetmap.org/wiki/Nominatim, 1st September 2018

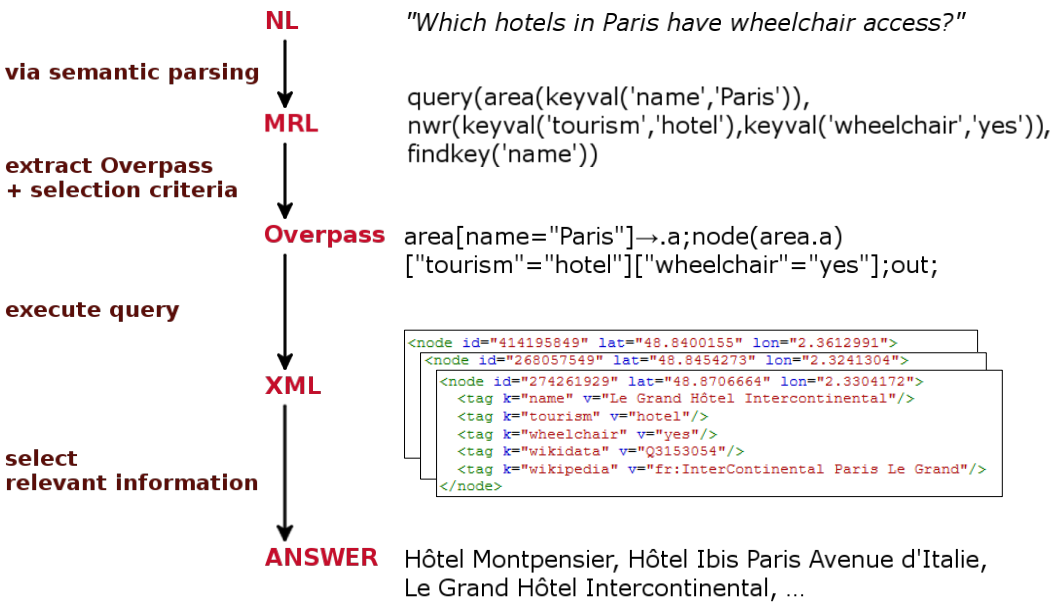


Figure 4.1.: Pipeline for mapping a natural language question to an answer. The natural language question is transformed into a parse in the NLmaps MRL using a semantic parser. The parse is converted to appropriate Overpass queries which are executed against the database. The returned set of database objects, which may be represented in XML format, are then processed to obtain the correct answer.

where no knowledge about OSM tags is required, yet it is possible to run searches that go beyond string matching.

Next, we investigate a different framework for semantic parsing. Neural networks have demonstrated often superior performance to linear models in recent years. They have successfully been employed for semantic parsing, using both question-parse pairs (Jia and Liang, 2016; Dong and Lapata, 2016) and question-answer pairs (Neelakantan et al. (2017); Liang et al. (2017); Mou et al. (2017); Guu et al. (2017), *inter alia*). We show that the RNN-based sequence-to-sequence framework Nematus (Sennrich et al., 2017), even out-of-the-box, can outperform the carefully extended Cdec models, this time both trained and tested on NLmaps v2. Extending the Nematus model by handling named entity recognition separately, we can further improve the performance of our semantic parsing models on the NLmaps v2 corpus. Our final semantic parsing model can achieve a F1 score of over 90% on the test set of NLmaps v2. Finally, we offer an error analysis to better understand which type of errors the different semantic parsing models make.

Finally, we validate the automatic extension, which constitutes parts of the NLmaps v2 corpus, by showing that a model trained on NLmaps v2 training data performs better on the NLmaps v1 test set than a parser trained only with NLmaps v1 training data. In accordance, the experiment also shows that a parser trained on NLmaps v1 training data performs badly on the NLmaps v2 test set because the NLmaps v1 corpus lacks many OSM tags introduced by the automatic extension.

The main contributions of this chapter are as follows:

- Extension of the hierarchical-phrase based machine translation framework Cdec (Dyer et al., 2010) to suit semantic parsing for linearisable MRL parses.
- Empirical results using the modified Cdec framework for the corpora NLmaps and NLmaps v2.
- Design of a natural language web interface for OSM and connecting it to a semantic parsing model.
- Further extensions to the semantic parsing pipeline for a better user experience when using the web interface.
- Employing the RNN-based sequence-to-sequence framework Nematus. (Sennrich et al., 2017) on NLmaps and NLmaps v2.
- Training a Named Entity Recognition (NER) model using Nematus to handle named entities separately from semantic parsing.
- Error analysis of the models trained on NLmaps v2.
- Empirical validation of the automatic corpus extension.

The structure of this chapter is as follows: Modifications to the hierarchical phrase-based machine translation system Cdec (Dyer et al., 2010) and experiments using the NLmaps corpus can be found in Section 4.1. Section 4.2 presents the natural language web interface to OSM with several extensions that aim to improve user experience. Lastly, Section 4.3 presents results on the NLmaps v2 corpus for both Cdec and the neural network framework Nematus (Sennrich et al., 2017). It also provides an error analysis of the NLmaps v2 models and an empirical validation of the automatic corpus extension.

The work presented in this chapter has in part been previously published in peer-reviewed publications. Section 4.1 has been published in Haas and Riezler (2016) and Section 4.2 closely follows Lawrence and Riezler (2016). Parts of Section 4.3 appear in Lawrence and Riezler (2018).

4.1. Linear Model: Phrase-Based Machine Translation

We modify the hierarchical phrase-based SMT framework Cdec (Dyer et al., 2010) (see Section 2.3) and treat semantic parsing as a translation task where we translate from natural language questions to parses in the NLmaps MRL. Prior to this, we apply a few pre-processing steps. First, parses are linearised so that splitting the linearised parse at white spaces leads to individual tokens (see Chapter 3.2), e.g.

```
query(area(keyval('name','Paris')),nwr(keyval('tourism',
'hotel'),keyval('wheelchair', 'yes')),findkey('name'))
```

is transformed into the individual tokens

```
query@3 area@1 keyval@2 name@0 Paris@s nwr@2 keyval@2
tourism@0 hotel@s keyval@2 wheelchair@0 yes@s findkey@1 name@0.
```

Second, the natural language questions are pre-processed. All words are lower-cased and tokenised using a set of regular expressions where a whitespace is inserted before the characters $\{?, !; \}$ in all instances and before $\{.\}$ if the fullstop occurs at the end of a sentence. Lastly, the individual words are stemmed using the NLTK toolkit (Bird et al., 2009), which ensures that inflected word forms all map to the same stem.

4.1.1. Modifications for Semantic Parsing

With both natural language and MRL sides split into individual tokens, we can employ any machine translation framework to learn how to translate natural language questions into parses of the NLmaps MRL. First, we use the alignment tool GIZA++ (Och and Ney, 2003) on the training data to generate word alignments for both translation directions. The alignments from both translation directions can be combined in various heuristic ways (Koehn, 2010, Chapter 4.5.3). Preliminary experiments showed that using the `intersect` heuristic leads to the best results. The `intersect` heuristic only keeps an alignment point if it appears in both translation directions, which ensures that only the most reliable points are kept. Next, a target-side 5-gram language model is trained using SRILM (Stolcke, 2002). The alignments are given to Cdec to generate grammar rules. Cdec's grammar extractor standardly uses 12 dense features and extracts SCFGs for test and development sets. The various features are combined in a log-linear model (see Equation 2.17 in Section 2.3) and they are weighted by a weight vector w, which can be tuned in a discriminative setup. For the setup with the standard 12 dense features, we employ the tuning algorithm Mert (Och, 2003).

Given an input question, Cdec's decoder can return an k-best list of parses ordered from most to least probable parse. However, some linearised parses do not form valid trees and are thus not valid under the NLmaps MRL. Thus, the k-best list is searched for the first, highest ranking parse that is a valid tree. Preliminary experiments with the size of the k-best list showed that a size of $k = 100$ offers a good trade-off between speed and performance. Larger lists rarely produce any further valid parses while increasing the time spent fruitlessly searching.

Once a valid parse is found, it is converted into the original structure of the NLmaps MRL and executed against the OSM database. An answer is considered correct if it is identical to the gold answer. Given the percentage of correct answers in a test set, we can measure the *recall* of a model. The percentage of correct answers out of all non-empty answers in the test set, defines the *precision* of a model. Overall, a model is evaluated by the *F1* score which calculates the harmonic mean between recall and precision, i.e. $2.0 \cdot precision \cdot recall/(precision + recall)$.

The described setup allows us to build a first semantic parser using Cdec. However, we introduce three extensions to this baseline model to further tailor the setup for semantic

parsing on the NLmaps corpus. First, the 12 standard dense features can be extended with sparse features. In addition to the dense features, we also extract rule shapes, identifiers and bigrams from the source and target sentence (Simianer et al., 2012). Using sparse features vastly increases the number of features in the log-linear model and consequently the number of parameters in w that need to be tuned. Mert is not able to handle such a large number of features. Thus, we use the tuning algorithm Mira (Crammer and Singer, 2003) in conjunction with sparse features. Models with this feature are marked with `+sparse`.

Named entities that appear in the natural language question, need to be slotted verbatim into the correct position in the parse. While named entities seen in the training corpus can be learnt as a translation rule, this is not possible for previously unseen named entities. This inhibits the parser's possibility to generalise to unseen named entities. Our second extension addresses this problem. Conveniently, Cdec offers the option to pass through words for which it cannot find a translation rule. However, due to the linearisation, named entities on the MRL side have an additional suffix, consisting of the marker "@" and a number which indicates the amount of children of the current node. Named entities that are passed through from the natural language question would be missing such a suffix. Due to the definition of the NLmaps MRL, we know that named entities can only appear as leaves and as the last argument of their parent node. This means they will always require the suffix "@*s*". Thus, the suffix "@*s*" is appended to any word that is passed through from the natural language side. Models with this feature are marked with `+pass`.

During the traversal of the k-best list, we search for a parse that is a valid tree. But a valid tree does not automatically guarantee a valid parse under the NLmaps MRL. As a third extension, we employ the CFG introduced in Section 3.2 to run this additional check. Models with this feature are marked with `+cfg`.

4.1.2. Empirical Results

Using the first version of NLmaps, we randomly split the corpus into a training set of 1,500 question-parse pairs and a test set of 880 question-parse pairs. The training data is also used as the development data to tune the weights w. In all setups we used `intersection` as the alignment combination heuristic and stemming on the natural language side. Row #1 of Table 4.1 gives the results for the standard Cdec model using 12 dense feature and Mert to tune w. Due to Mert's optimiser instability (Clark et al., 2011), experimental results using Mert are reported averaged over 3 independent runs. With a recall of 62.42%, slightly less than two-thirds of the questions in the test set are answered correctly. The precision of the model indicates that if an answer is returned, we can be about 84% confident that it is the correct one.

Row # 2 of Table 4.1 presents the result for the setup using sparse features and the tuning algorithm Mira. It can outperform the previous setup by about 2 percentage points in F1 score which is significant at $p < 0.05$ using an approximate randomisation significance test (Noreen, 1989). Row # 3 presents the results for the model that uses dense features and the extensions `+cfg` and `+pass`. It can significantly outperform the Cdec base model, but not the model with additional sparse features. However, combining the sparse features with

		Recall	Precision	F1	ΔF1
1	Cdec	62.42	84.69	71.87	
2	Cdec `+sparse`	65.80	84.40	73.95^{1}	+2.08
3	Cdec `+pass +cfg`	65.19	**89.45**	75.41^{1}	+3.54
4	Cdec `+sparse +pass +cfg`	**68.3**	89.04	$\mathbf{77.30}^{1-3}$	+5.43

Table 4.1.: F1 scores on the NLmaps test set for various linear semantic parser settings. Tuning was carried out on the training set. In the case of Mert tuning, the results are averaged over 3 independent runs due to the randomness Mert introduces. Best results are indicated in **bold face**. Statistical significance in terms of F1 of system differences at $p < 0.05$ are indicated by experiment number in superscript.

the other two extensions (see Row # 4) leads to the overall highest F1 score and this model significantly outperforms all others.

Using Cdec with all introduced extensions and the first version of the NLmaps corpus, we are able to create a semantic parser which can answer more than two-thirds of the test questions correctly and offers a confidence of about 89% that a non-empty answer is correct. This is a promising first step for a natural language interface to OSM. Next, we design a user interface that sends user input to the semantic parser and then presents the returned answer.

4.2. Extensions for a Web Interface

For a fully functional natural language interface to OSM, we require a user interface through which users can interact with the semantic parser. We build a web interface, using HTML, CSS and Javascript. The CSS is supplemented by Bootstrap 3.0[20] for the dynamic adjusting of the user interface for various screen sizes, including mobile devices. We employ Python-based CGI programming for scripts that transfer questions to the back-end semantic parsing system and await an answer. A screenshot of the interface with an example question and its corresponding answer, can be found in Figure 4.2.

The interface should be intuitive and convenient to use for everyday users. To this end, we first introduce additional components that will supplement the semantic parser introduced in Section 4.1. Second, we present experiments for two new semantic parsers that were modified specifically for the use in conjunction with the interface. Finally, we describe a form that users can fill out to provide feedback once a question is answered. The resulting interface can be reached via `http://nlmaps.cl.uni-heidelberg.de/` (1st September 2018).

4.2.1. Additional Components

Assisted Area Recognition. The best performing Cdec parsing system from Section 4.1 passes unknown words through, so that named entities, such as town names, can be parsed

[20] `http://getbootstrap.com/`, 1st September 2018

correctly despite never having been seen during training. This feature is of vital importance for the web interface because OSM is a global database and it would be impossible to have all named entities in a training set. However, many names occur more than once on the global map and this can lead to ambiguities. If area names are not disambiguated, then the search for OSM objects will be performed in all areas with the corresponding name. This is undesired in the majority of use cases where users are typically only interested in searching in one area, rather than all areas with a certain name. We offer three different solutions for this problem and users can easily avail themselves of the solution that is most convenient to them.

As a first option, we let Nominatim handle the disambiguation. Nominatim is a promising tool for this problem because it specialises in ranking geographical places based on various features when given an input string (see Section 3.1.1). Once a question is entered, we first identify area names in the question. This is done by a simple but effective string-matching approach. A word is marked as an area if its preceded by one of the following words: "*in*", "*around*" or "*vicinity of*". Area names identified during this step, are sent to Nominatim, which returns a ranked list of OSM objects. For example, Nominatim ranks an area with a city tag higher than an area with a village tag of the same name. Given the ranked list, the highest ranking area is selected as the target area for the question and this area's unique identifier is automatically inserted as a child of the `area` operator in the later parse.

However, a user might not be interested in the highest ranked area. If "*Heidelberg*" is sent to Nominatim, the highest ranked result refers to the city of that name in Germany. This is a conflict if the user is instead interested in a village named "*Heidelberg*" located in Pennsylvania, USA. Thus, as a second option, we offer an alternative input text area where users can specifically enter a more fine-grained area name. If a user enters "*Heidelberg, Pennsylvania*" into this box, then this information is given to Nominatim. With the additionally provided information, Nominatim has no problem returning the desired area as the highest-ranked one.

Finally, we offer a third option which is particularly convenient for the user if they want to search the area of the city that they are currently residing in. In this case, the user can simply click a button labelled "*Use my location*". After giving their explicit permission to be located, the Geolocation API[21] is called to determine the user's GPS location. Sending the GPS location to Nominatim's reverse geocoding component, we can retrieve all OSM areas the user is in. The returned areas are annotated with a number that indicates the type of area and thus the city-level area can be selected.

Answer Presentation. Once a question has been parsed and the most likely parse has been executed against the database, the text-based answer is handed back to the website via a CGI script. This answer is returned in a box below the question input box. In addition to the text-based answer, we also want to provide the user with an interactive map. For this, the NLmaps extension to the Overpass API is further modified. During the database execution for the text-based answer, we additionally save each OSM object that participates in forming the text-based answer. For OSM nodes, we record the associated GPS coordinate. For ways

[21] `http://www.w3schools.com/html/html5_geolocation.asp`, 1st September 2018

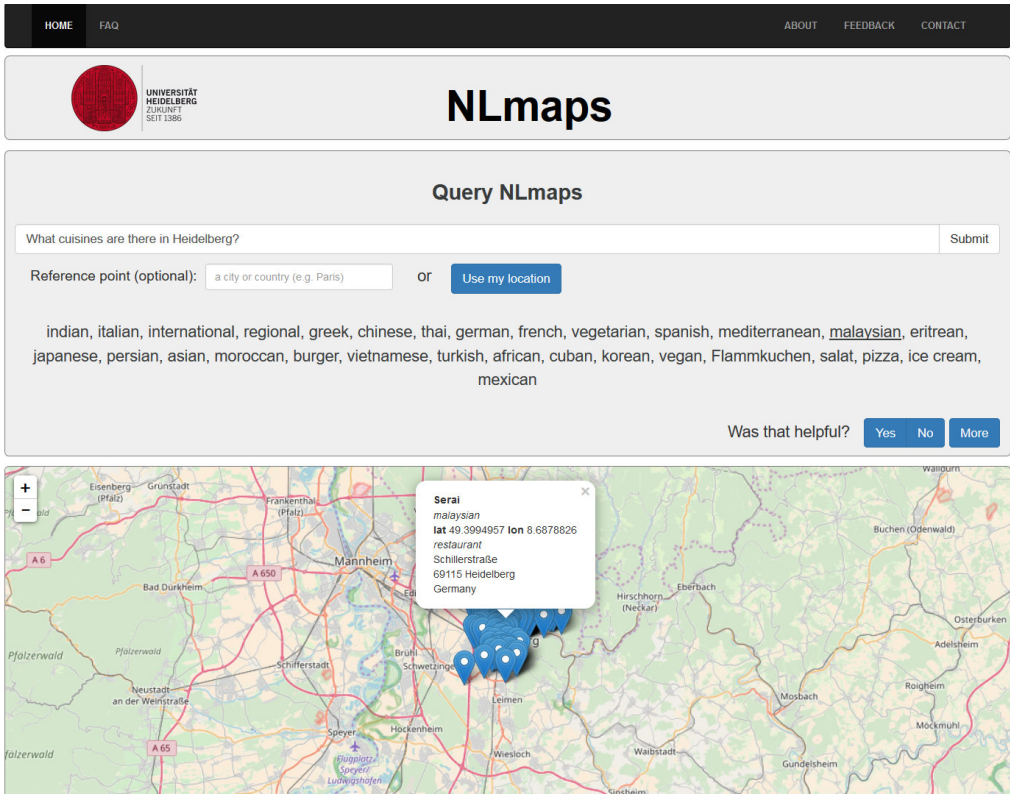


Figure 4.2.: A screenshot of the NLmaps web interface. Once a question has been processed, a text-based answer is presented in a box. If the answer is a list, then the individual elements contain hyperlinks. Clicking a link will open up information boxes on the corresponding map markers.

and relations, the centre coordinate is calculated based on the participating nodes and their coordinates. For each object, a marker is placed on the interactive map at the recorded or calculated GPS coordinate. To display the interactive map, we connect to an OSM tiles server[22] and markers are placed upon the map using Leaflet.[23]

Every marker on the map can be clicked on, opening a pop-up that provides more information about the underlying OSM object. This information is also collected during the single database execution and supplied to the correct marker using the GeoJSON format.[24] First, if the OSM object has a "*name*" key, its value is displayed at the top of the pop-up. Furthermore, we display the GPS coordinate of the marker. The GPS coordinate can also be sent to Nominatim to obtain a reverse decoded street address, which we also provide. Lastly, we present the values of the tags that appear in the executed parse.

For `findkey`-based parses, the elements of the resulting list of answers are presented as hyperlinks. If clicked, the pop-ups of the applicable markers are opened automatically. For example, Figure 4.2 shows a screenshot of the interface for the question "*What cuisines are there in Heidelberg?*". If one clicks on a particular cuisine, such as "*malaysian*", the pop-ups

[22] `http://wiki.openstreetmap.org/wiki/Tiles`, 1st September 2018
[23] `http://leafletjs.com/`, 1st September 2018
[24] `http://geojson.org`, 1st September 2018

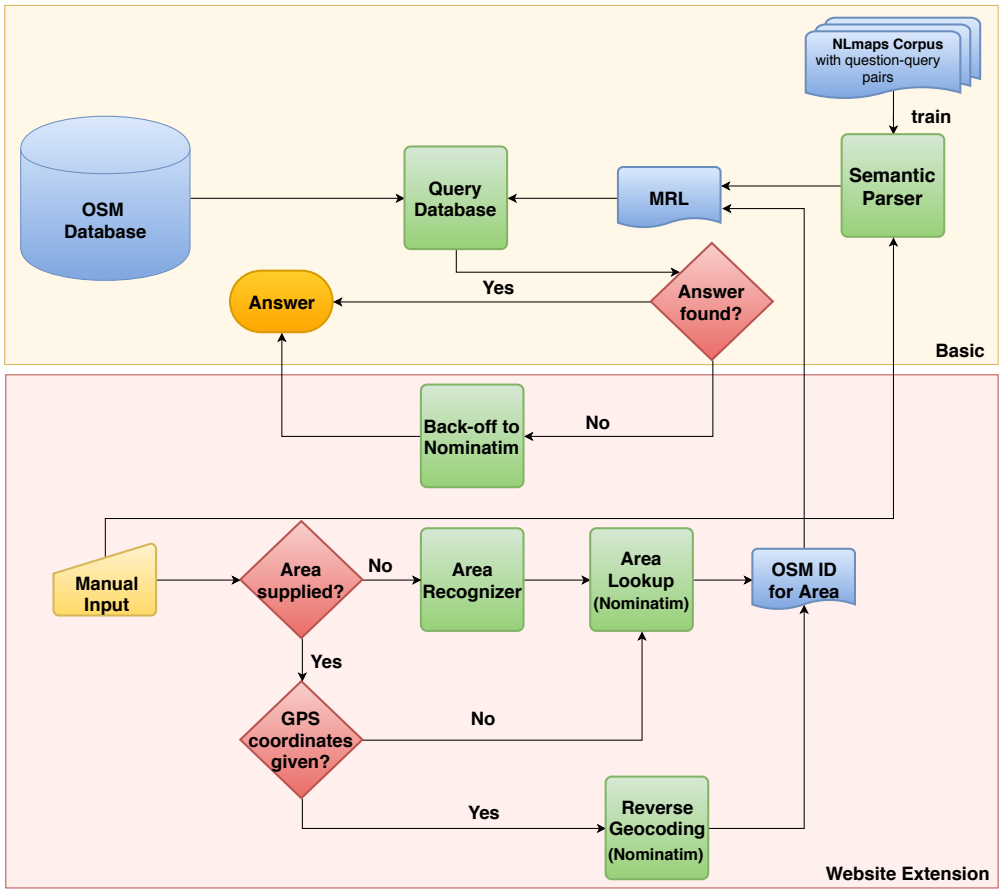


Figure 4.3.: Program flow after the user entered a question in the input field for the natural language interface to OSM.

of all OSM objects with the tag "*cuisine=malaysian*" are opened. This information is also repeated alongside the other nuggets of information present on a pop-up.

Back-off System. If the semantic parser cannot find an answer, we would still like to present the user with some useful information. Thus, should the answer be an empty string, we pass the question on to Nominatim and present the result returned by Nominatim. This can be helpful for very short questions, such as "*Golden Gate Bridge*", as such questions don't appear in the NLmaps corpus, precisely because Nominatim is capable of handling these.

An overview of the complete process that is started once a manual input has been received, can be found in Figure 4.3.

4.2.2. Empirical Results

For the natural language interface, we use the the setup `cdec +pass +cfg` from Section 4.1. The setup using sparse features led to better results, but due to the increase of features,

NLmaps		Precision	Recall	F1	ΔF1
1	Cdec +pass +cfg	**89.90**	64.02	74.78	
2	Cdec +pass +cfg +IDs	88.67	**66.17**	**75.78**	+ 1.00
3	Cdec +pass +cfg +IDs +SE	89.56	65.67	75.71	+ 0.93
SE		Precision	Recall	F1	
1	Cdec +pass +cfg	71.56	39.27	50.71	
2	Cdec +pass +cfg +IDs	63.32	40.4	49.29	+ 1.42
3	Cdec +pass +cfg +IDs +SE	**90.86**	**71.64**	**80.11**	+29.40

Table 4.2.: F1 scores on the NLmaps and SE test sets for various web interface extensions. Results are averaged over three independent runs because of the randomisation introduced by Mert. Best results are indicated in **bold face**.

parsing slows down significantly. As speed is a crucial concern for the interface, we do not use sparse features here. The results[25] on the NLmaps test set for the setup `cdec +pass +cfg` can be found in Row #1 of Table 4.2. We add two extensions to this model specifically tailored for the web interface.

For the first extension, we apply the area recognition described in Section 4.2.1. The string-matching approach is used to automatically identify locations in the question and Nominatim is used to retrieve the highest ranked OSM object for the identified location. For example, the question "*Where are restaurants in Heidelberg?*" is changed to "*Where are restaurants in 3600285864*" where "*3600285864*" is the unique identifier for the city called Heidelberg in Germany. This approach works more effectively than relying on Cdec's pass through feature for unknown words. Using this extension is indicated by `+IDs` in Table 4.2. Adding it to the previous setup leads to a slight increase of 1 percentage point in F1 score.

During an early trial phase, we noticed that users tended to omit non-crucial words in their question, akin to search engine queries. This leads to shorter, often ungrammatical questions which differ from the questions found in the NLmaps corpus. Modifying the test set of NLmaps to resemble such search-engine-style questions leads to the SE test set (also see Section 3.4.1). Results on this test set are reported in the second half of Table 4.2. Both `cdec +pass +cfg` and the `+ID` extension in row #1 and #2, respectively, show a significant drop in performance when the model is trained on the training data of the original NLmaps corpus, but tested on the SE test set. To close this gap in performance, we also modify the training set of the NLmaps corpus leading to the search engine style extension described in Section 3.4.1. Adding the training data from this extension to the original training data results in the model marked with `+SE` in Row #3 of Table 4.2. With this extension, the parser can successfully increase its performance on the SE test set. It is this model that was deployed for the web interface when it went live in 2016.

[25]The results differ slightly from Table 4.1 as the model has been re-run and the underlying OSM database was updated.

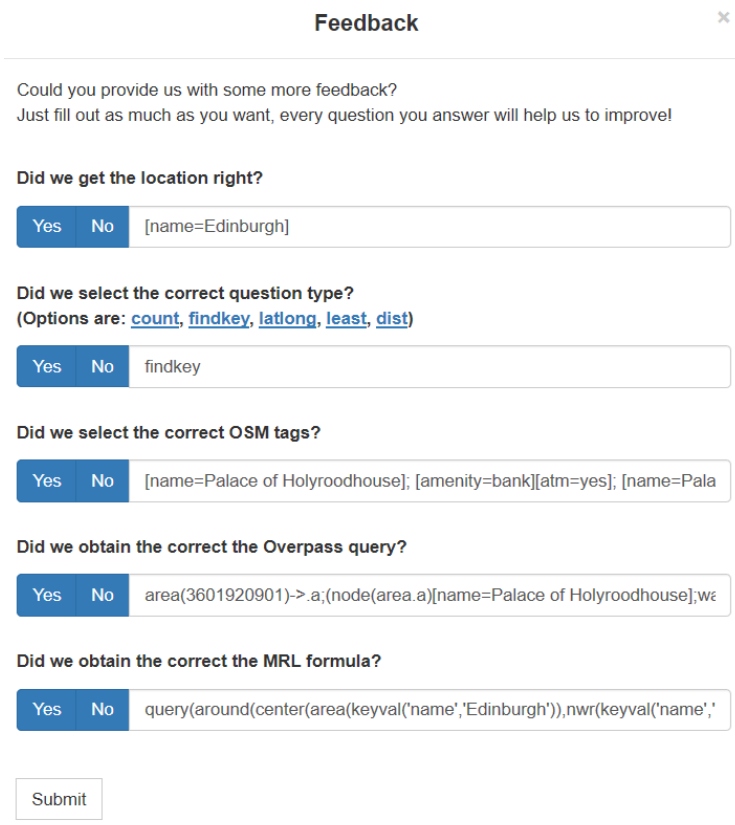


Figure 4.4.: A screenshot of the feedback form a user can fill in once a question has been processed and an answer presented.

4.2.3. Feedback Form

After a user entered a question and received an answer, we provide two options for users to give feedback. Below the text-based answer box, we ask the user if the answer was helpful with regards to their question (*"Was that helpful?"*) and the user can provide a binary answer by either clicking a *"Yes"* or a *"No"* button.

There is an additional button labeled *"More"* which opens up a more detailed feedback form that a user can optionally fill in. The questions in this form become progressively more complex, i.e. they require more expert knowledge. The first two questions (*"Did we get the location right?"* and *"Did we select the correct question type?"*) can be answered by any user. For the next question (*"Did we select the correct OSM tags?"*), the user would need knowledge about the OSM database. The questions after that (*"Did we obtain the correct Overpass query?"* and *"Did we obtain the correct the MRL formula?"*) would require familiarity with the Overpass API and the NLmaps MRL, respectively. A user can either supply the correct answer or provide binary feedback on whether or not the answer suggested by the system is correct or not. If a user is able to supply the correct parse, the resulting question-parse pair could be added to the corpus. But most users would not be able to give this type of feedback. All

other feedback options can be seen as weaker supervision signals of varying granularity and could be used to improve the parser.

The online web interface offers users the option of using natural language to parse the OSM database. The success of this crucially depends on the performance of the underlying semantic parser. Next, we investigate if NLmaps v2 can produce a better semantic parser. Additionally, we explore if the neural sequence-to-sequence framework Nematus can achieve better results than the linear model Cdec.

4.3. Neural Model: Sequence-to-Sequence Learning

Nematus (Sennrich et al., 2017) is a state-of-the-art, neural-network based sequence-to-sequence framework employing an encoder-decoder setup with attention (Cho et al. (2014), Sutskever et al. (2014), Bahdanau et al. (2015)) as described in Section 2.4. As the natural language input to the system we use the same pre-processing steps as described previously at the beginning of Section 4.1. The target outputs are again linearised parses. In our experiments we employ 1,024 hidden units and a word embedding of size 1,000. The maximum output length is 200 and the number of tokens in the training set determine both input and output vocabulary size. The learning rate optimiser is Adadelta (Zeiler, 2012) with gradients being clipped to 1.0 should they exceed this value. The batch size is set to 1 and the training data is shuffled after each epoch.

At regular intervals, a development set is processed, the resulting parses are executed against the OSM database and the F1 score with regards to the gold answers is measured. The best model is chosen according to the highest F1 score on the development set. Translations for development and test set are obtained by employing beam search with a beam of size 12.

4.3.1. Empirical Results

For the experiments using Nematus, we employ NLmaps v2 which subsumes NLmaps. All test instances of NLmaps are part of the test set of NLmaps v2. Keeping the same set ratios as NLmaps, we split NLmaps v2 into a training set of 16,172 instances, with an additional 2,000 instances reserved for the development set, and a test set of 10,594 instances. To be able to draw comparisons between Nematus and Cdec models, we train two new models using the best Cdec setup (`+pass +cfg`, shortened to `+pc` in Table 4.3), once with and once without sparse features, on the NLmaps v2 corpus.

Row #1 in Table 4.3 presents the results using Mert for tuning, whereas row #2 additionally uses sparse (`+sparse`) feature and thus tunes using Mira. Analogous to the results for the original NLmaps, the latter setup performs better. Row #3 presents the results using the neural network settings described above. Even without any further modifications, the Nematus model significantly outperforms both Cdec models by over 3 percentage points in F1 score.

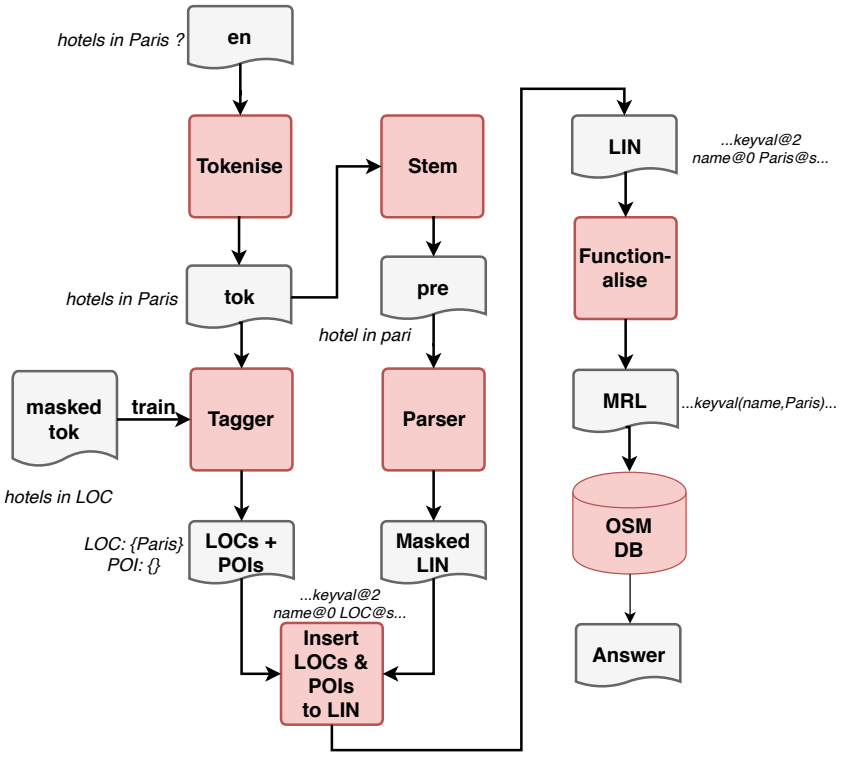


Figure 4.5.: Flowchart of the approach to handle named entities separately from the semantic parsing task by training an additional NER tagger. An example is illustrated in *italic* font.

The last column in Table 4.3 gives an insight into an additional advantage of using Nematus over Cdec. It reports the number of instances where no valid MRL parse can be found. Recall that both Cdec models use the NLmaps CFG to look for valid NLmaps parses in the k-best list, whereas the Nematus models simply output the most likely translation after a beam search with beam size 12. Despite this apparent disadvantage for Nematus, Nematus has a vastly larger rate of valid parses. For the Cdec models, in about 15% of the cases, no valid NLmaps parse can be found in the list of the 100 most likely parses. On the other hand, less than 1% of parses are invalid for the Nematus models where only the most likely output was considered.

A neural network that does not use sub-word units, cannot produce words that were not seen in the training set. This is a problem for the generalisation of named entities, such as town names or names of points of interest. For the task of building a natural language interface to OSM, generalisation to any town name in the world is paramount. Using a sub-word unit technique, such as Byte-Pair-Encoding (BPE) (Sennrich et al., 2016), could be a solution. However, this approach is not ideal because nonsense words can be created. Copying named entities directly from the question into the correct position in the parse would be more effective.

To effectively copy named entities, we propose to utilise the version of NLmaps v2 where locations and points of interests are masked with the placeholders `$LOCATION` (here shortened

		Recall	Precision	F1	Δ F1	No parse
1	Cdec +pc	69.04	84.99	76.19 ±0.00		15.10%
2	Cdec +pc +sparse	70.24	85.15	76.98 ±0.00	+ 0.79	13.55%
3	Nematus	77.97	82.91	80.36 ±0.11	+ 4.17	0.30%
4	Nematus +ner	**88.20**	**92.07**	**90.09**±0.04	+13.90	0.32%

Table 4.3.: F1 scores on the NLmaps v2 test set for various linear and neural semantic parser settings. "No parse" gives the percentage of instances where the model could not find a valid NLmaps parse. For Cdec, tuning was carried out on the development set and in the case of Mert tuning (row #1), the results are averaged over three independent runs due to the randomness Mert introduces. Both Nematus models are also averaged over three runs. Best results are indicated in **bold face**. All models are statistically significant from each other at $p < 0.05$.

to `$LOC`) and `$POI`, respectively (see last paragraph of Section 3.4.2). Using the set of tokenised questions, such as "*hotels in Paris*" and their masked counterpart, "*hotels in* `$LOC`", we train another neural network to identify named entities. The masked sequences are converted to sequences composed of three types, $\{O, L, P\}$, where `$LOC` is mapped to L, `$POI` to P and all other tokens to O. Based on this, we train an NER neural network and we use the development set to select the model with the best sequence level accuracy, which is 99.13%. The model is then used to automatically classify and extract all locations and points of interest on the test set.

Alongside the NER model, we train a semantic parser where stemmed questions, e.g. "*hotel in pari*", serve as input. The parser learns to map them to masked parses such as

query@3 area@1 keyval@2 name@0 `$LOC@s` *nwr@0 tourism@0 hotel@s qtype@1 latlong@0.*

The placeholders `$LOC` and `$POI` in the linearised parse are then replaced with the locations and points of interest identified by the NER model. Consecutive L or P tags are grouped together and treated as a multi-word location or point of interest. Non-consecutive L or P tags are treated as separate locations or points of interest and are pasted into parses by order of occurrence. This procedure cannot introduce any errors because parses are insensitive to the order of locations or points of interest. Reversing the linearisation of the parse, results in a fully formed parse that can be executed against the database, i.e.

query(*area*(*keyval*('*name*','*Paris*')), *nwr*('*tourism*','*hotel*'),*qtype*(*latlong*)).

The results for this approach are presented in row #4 of Table 4.3. Solving the problem of unknown named entities, our parser can now achieve a F1 score of over 90.00%, significantly beating all other models. This model serves as the back-end parser for the graphical web interface as of November 2018.

4.3.2. Error Analysis

For an error analysis, we compare all parses that lead to an incorrect answer to the correct parses and automatically detect the following error types:

- OSM tag: An error of this type means that either the first, second or both arguments of `keyval` are wrong
- Qtype: This error indicates that one or more arguments of `qtype` are wrong.
- Wrong Distance: A Wrong Distance error indicates that the radius size was chosen wrongly (e.g. `DIST_INTOWN` instead of `WALKING_DIST`).
- Skeleton: A Skeleton error implies that one of the following operators were either missing or superfluous in the parse compared to the gold parse: `around`, `area`, `and`, `or` or a cardinal direction. If Skeleton errors are identified, the previously mentioned errors are not checked.
- Invalid Parse: In the case of this error, the model did not return a valid MRL parse. By definition, the other errors cannot be checked if this error occurs.

Additionally, we split the group OSM tag into the following sub-categories:

- NE: For an NE error, the first argument of `keyval` is correctly "*name*", but the corresponding value is wrong, i.e. the model did not pass on the named entity from the question to the parse correctly.
- Not NE: All other OSM tag errors.

	Cdec +pc	+sparse	Nematus	+ner
OSM tag	1,760	1,820 ↑3.43%	3,414 ↑94.00%	1,161 ↓34.04%
NE	971	1,018 ↑4.80%	2,652 ↑172.99%	278 ↓71.38%
Not NE	788	802 ↑1.73%	762 ↓3.34%	883 ↑11.97%
Qtype	148	135 ↓8.78%	196 ↑32.21%	237 ↑60.36%
Wrong Dist.	840	922 ↑9.72%	21 ↓97.50%	42 ↓95.04%
Skeleton	168	142 ↓15.64%	80 ↓52.48%	103 ↓38.61%
Invalid Parse	1,600	1,435 ↓10.31%	131 ↓91.79%	36 ↓97.75%
Total	4,516	4,454 ↓1.38%	3,842 ↓14.94%	1,579 ↓65.04%

Table 4.4.: Overview of which type of errors the NLmaps v2 parsing models make and the percental in- or decrease with regards to the first model.

	Cdec +pc	+sparse	Nematus	+ner
count	9.69%	10.6%	3.6%	3.95%
findkey	35.26%	39.07%	70.05%	73.02%
latlong	48.04%	41.72%	18.99%	16.49%
least	7.01%	8.61%	7.36%	6.54%

Table 4.5.: Percental overview of which type of qtype errors the NLmaps v2 parsing models make.

In Table 4.4, we present the number of errors in each category for the various models. For models with randomised components, we report the average number for each error type over the three independent runs. Starting with the second model, we also show the percental

in- or decrease of the error type in comparison to the first model. For the first Cdec model, we see that the largest number of errors occurs from not being able to find a valid MRL in the k-best list. This is closely followed by errors stemming from OSM tags whereas the majority is due to wrongly passed through named entities. The Cdec model with sparse features shows slight decreases for some error types but increases for others, leading to a slight overall reduction of errors by 1.38%.

Using the neural network framework Nematus, leads to a more dramatic shift in error types. Because the neural network approach effortlessly learns what constitutes a correct NLmaps parse, we observe a large decrease in the error type Invalid Parse. Similarly, we observe a decrease in the error types Wrong Distance and Skeleton. However, the vanilla Nematus model does not have the ability to handle unknown named entities. Thus, we record a large increase in NE errors.

The number of Qtype errors also increases and we present a percental overview of which Qtype arguments are parsed wrongly for the different models in Table 4.5. While there is a percental decrease in `count` and `latlong` errors, the rate of `findkey` error is higher for the Nematus models compared to the Cdec models. We conjecture that the more restrictive grammar rules of Cdec are an advantage in this instance because they ensure that the argument of `findkey` also appears in the input side. Neural networks on the other hand often tend to focus more on fluency (Tu et al., 2016), which implies that the model is more concerned in finding a token that conforms to the NLmaps grammar, rather than a token that is directly motivated by a token in the input question. This is also evidenced by how well the Nematus models learn the structure of the NLmaps MRL, which leads to nearly no Invalid Parse errors. Despite this, in total the Nematus model can record a decrease in errors of 14.95% compared to the first Cdec model.

The Nematus +ner model is able to combine the best of both worlds. Similar to the first Nematus model, the model does not make many errors of the types Wrong Distance and Invalid Parse. At the same time, it elegantly handles unknown named entities, leading to a drastically lower count of NE errors than the previous models. Overall, this model can reduce the errors by 65.04% compared to the first Cdec model.

		Train		
		v1	v2	Δ
Test	v1	73.56±0.61	75.49±0.01	+ 1.93
	v2	28.31±0.25	80.36±0.11	+52.05

Table 4.6.: F1 scores on the NLmaps and NLmaps v2 test sets for neural parsers trained on NLmaps and NLmaps v2 training sets, respectively. Results are averaged over three independent runs. Results are statistically significant at $p < 0.05$.

4.3.3. Empirical Validation of the Automatic Corpus Extension

To empirically validate the usefulness of the automatically created data, we compare two parsers trained with Nematus. The first model is trained using the original NLmaps training

data and the second using the training data of NLmaps v2 which contains the original training data and additional synthetic data. Both systems are tested on the original NLmaps test data and on the test set of NLmaps v2. Results may be found in Table 4.6. On the original test set, adding the automatically generated instances allows the parser to significantly improve by 3.22 points in F1 score. The parser trained on the original training data performs badly on the new test set because it is ignorant of many OSM tags that were introduced with the extension. This is not the case if we employ NLmaps v2 training data.

Conclusion

For a natural language interface to OSM, we require a semantic parser that can map natural language questions to parses, which can be executed against the database. We presented several semantic parsers using two different frameworks. The first framework, Cdec, is based on hierarchical phrase-based machine translation and employs a log-linear model. Using the NLmaps corpus, we trained a first semantic parser using the standard settings of Cdec. This baseline parser was improved upon with several extensions. We added sparse features to the model, enabled the correct passing through for named entities and included a check that verifies whether a particular parse is a valid NLmaps MRL parse or not. This led to a parsing model with an F1 score of 77.3% on the NLmaps corpus. Moving to NLmaps v2, leads to larger training and test sets which include the respective sets of NLmaps. For the same setup, the semantic parser reaches a comparable score of 76.19% F1 on this data set.

The second framework, Nematus, trains a RNN-based sequence-to-sequence neural network. Even without any extensions, a model trained with Nematus can outperform the best Cdec model. However, this Nematus baseline model cannot handle previously unseen named entities. Using the NLmaps v2 data, which contains masks for points of interests and locations (see last paragraph of Section 3.4.2), we trained an additional NER model to recognise locations and points of interests in a natural language question. This crucial change drastically improved the performance of the semantic parser, leading to an F1 score of 90.09%.

We also introduced a web user interface via which users can interact with the semantic parser. Several components are added to provide a positive user experience. Answers returned by the parser are presented textually as well as interactively on a world map. Once an answer is presented, the user has several options to provide feedback on the correctness of the answer. Additionally, we offer several options for area detection to the user. Finally, Nominatim is queried as a back-off system if the semantic parser is unable to return an answer.

Part I Conclusion

Our goal was to build a natural language interface to OpenStreetMap (OSM) because currently available search tools for non-expert users are based on simple string matching methods. As a consequence a wealth of information remains inaccessible to every day human users who do not know the particularities of OSM and its query languages. As a first step towards our goal, we introduced a new, specifically tailored Machine Readable Language (MRL) in Chapter 3, which wraps around the Overpass query language designed for expert users to query the OSM database. Additionally we insured that our MRL can filter more fine-grained information than Overpass. Second, we assembled large corpus of 28,609 question-parse pairs, called NLmaps v2. Third, in Chapter 4, we introduced various semantic parsers which can map natural language questions to parses in our defined MRL. Because we implemented our MRL alongside Overpass, a parse can be executed against the OSM database with a simple script call to obtain an answer. Our best semantic parser achieves an answer-level F1 score of over 90%. Fourth, we presented a graphical user interface, which offers various conveniences to users and which can connect to a semantic parser in the background. Ultimately, these four stepping stones, allowed us to assemble a fully functional natural language interface to OSM, which allows non-expert users to query the OSM database for complex concepts that go beyond string matching. The interface has been live since 2016 and can be reached via `http://nlmaps.cl.uni-heidelberg.de/` (1st September 2018).

So far, we used question-parse pairs to train our semantic parsers. However, question-parse pairs are expensive to obtain because the specifically tailored MRL is known to only a handful of expert users. Because of this, we want to explore alternative, weaker supervision signals that can be used to improve a semantic parser. To this end, we explore two different approaches to learn from feedback given to model outputs. In each approach we will return to improve a semantic parser built upon NLmaps v2. Additionally, each approach is also applied to a task where a machine translation system is to be improved, showcasing that the approaches can be applied successfully to differing sequence-to-sequence tasks for NLP.

Part II.

Response-Based On-Policy Learning

Chapter 5

Multilingual Question-Answering: Grounding Machine Translation in Task Feedback

For some tasks it is more beneficial to employ indirect, weaker supervision than obtaining direct gold targets to train a statistical model. Our first approach to learn from feedback, response-based on-policy learning (see also Section 1.2), can operate in such a scenario by grounding the model in a downstream task for which gold targets are available. Given an input, the model produces one or several outputs that are passed on as input to the downstream task. The output of the downstream task can be compared to the available gold targets. On the basis of this comparison, feedback is sent to the model with which it can be improved.

We present two scenarios for response-based on-policy learning, one in this chapter and one in the next chapter. For our first scenario, we are interested in improving the first model in a pipeline of two statistical models. If these models are trained in isolation, the first model might produce erroneous outputs from which the later model cannot recover. We employ response-based on-policy learning to improve the first model by grounding it in the final, downstream task. The success or failure on that task is used as a feedback signal for learning. This allows us to update the model distribution to be closer to the reward distribution of the downstream task that we want to perform well on.

In particular, we are interested in building a multilingual semantic parser. We assume a semantic parser has been trained to transform English questions into machine-readable parses. To be able to parse German questions, the questions are first translated into English by a machine translation model and the translation is given to the semantic parser.[26] Executing the resulting parse against the database returns an answer. Comparing this answer to the provided gold answer, allows us to judge whether or not the pipeline led to success or failure. For the machine translation system, a translation is considered correct if the meaning can be transferred to the semantic parser so that the produced parse leads to the correct answer. The feedback of whether or not the correct answer was reached, can

[26] We assume that a German semantic parser would be inferior to an English semantic parser, e.g. because a lot more training data is available in English.

be used to update the machine translation system. Grounding the machine translation system in the action of the intended downstream task, allows the system to learn how to work better in conjunction with the semantic parser and this in turn leads to a higher overall performance on the downstream task. A graphical representation of this setup can be found in Figure 5.1.

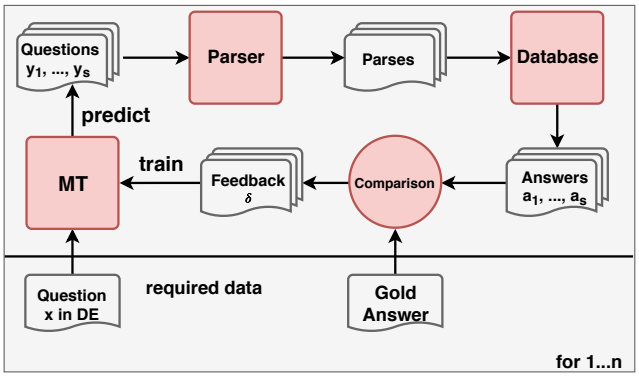


Figure 5.1.: Using extrinsic feedback from the downstream task to improve a machine translation system for a multilingual QA pipeline.

From the viewpoint of the machine translation system, response-based on-policy learning offers several advantages compared to learning with gold reference translations. First, a gold reference might not be reachable. Here, we can try out various translations until one is found that leads to positive task feedback and this translation can be considered a gold translation that is guaranteed by definition to be reachable. Second, the need for expensive, human-generated gold translations is alleviated because allowing the system to test various translations can lead to multiple correct translations being found. Having multiple correct translations to learn from, might allow the machine translation system to more easily distinguish between good and bad translations and it can learn about acceptable synonyms and structural variations. Third, the grounding in the downstream task results in translations of a different quality compared to a human-generated gold translation that was produced without the downstream task in mind. For example, for multilingual semantic parsing the fluency of the translation is secondary as long as the semantic parser can transform the translation into a correct parse. Finally, such response-based learning can be considered intuitive because it shares analogous concepts to human learning.

In all instances, successful preservation of meaning in a translation is defined by positive feedback from the downstream task. For each input, we aim to find an output, called a **hope** output, that we want to encourage, and another output, called a **fear** output, that we want to discourage. Using both hope and fear outputs, a ramp loss objective can be defined to improve the machine translation system.

Collobert et al. (2006) first introduced a ramp loss objective as a non-convex alternative to the hinge loss for binary classification. The concept was transferred to structured prediction by Do et al. (2008), who additionally show that the ramp loss offers a tighter upper bound of the true loss compared to the hinge loss as a convex alternative. McAllester and Keshet

(2011) show that the ramp loss produces models that approach the lowest possible task loss for a given feature set using a linear model with an appropriate regulariser and in the limit of infinite data. They also note that the ramp loss can be optimised using stochastic gradient descent and likened this update to the perceptron update (Collins, 2002).

A ramp loss objective has previously been used to tune a machine translation system (Gimpel and Smith, 2012), but hope and fear outputs where identified employing references instead of grounding the system in a downstream task. Grounding a machine translation system in a downstream task has been previously investigated in the context of cross-lingual information retrieval (Nikoulina et al., 2012) and in the context of adjusting the system to human preference for computer-assisted translation (Saluja et al. (2012); Barrachina et al. (2008); Koehn and Haddow (2009), *inter alia*).

For semantic parsing, research has been conducted to employ question-answer pairs rather than question-parse pairs (Clarke et al., 2010; Goldwasser and Roth, 2013; Kwiatkowski et al., 2013; Berant et al., 2013). In this scenario, the parser, rather than the machine translation system, is grounded in the downstream task where the result of the parse execution is compared to the intended gold answer. We combine the idea of using gold answers as a learning signal with the approach of Gimpel and Smith (2012): we tune a machine translation system which is grounded in the downstream task of obtaining the correct answer for a semantic parsing problem. Based on the success or failure to obtain the correct answer, we define hope and fear translations which are used to improve the machine translation system.

Concretely, for our task, we employ a general-purpose SMT system and tune it to translate German NLmaps questions into English. To this end, the NLmaps corpus has been translated into German by the author. We propose two discriminative algorithms, Ramp[27] and Response-based Online Learning (Rebol), which employ the feedback obtained from grounding the proposed translations in the final task to update the parameters of the SMT system. For both algorithms, each German question requires a corresponding gold answer. Rebol additionally utilises English reference translations, which are a valuable additional learning signal. We can demonstrate that the Ramp algorithm improves the machine translation system so that the overall final task performance rises by about 8 percentage points in F1 score. The Rebol algorithm achieves an even higher increase of 18 points in F1 score, but requires two gold targets for each input, which might be too expensive to obtain in praxis.

In our setup, we assume that gold answers are available. This stands in contrast to the gold parses which are available for the NLmaps corpus. However, for many domains gold answers are easier and cheaper to obtain than gold parses and thus we do not use the gold parses available in the NLmaps corpus in this setup.

The main contributions of this chapter are as follows:

- Re-implementation of the algorithms Rebol and Ramp in Python.
- Translation of the first version of NLmaps into German.
- Experiments with Rebol and Ramp on NLmaps and subsequent error analysis.

[27] Note that this algorithm is called Exec in Riezler et al. (2014) and Haas and Riezler (2016).

The work presented in this chapter has previously been published in Riezler et al. (2014), Haas (2014), Haas and Riezler (2015) and Haas and Riezler (2016). The algorithms were formulated by Stefan Riezler and a first implementation was coded in Ruby by Patrick Simianer. I employed the algorithm in its Ruby implementation successfully on the Geoquery corpus (Wong and Mooney, 2006) and on the Free917 corpus (Cai and Yates, 2013). Here, we apply Rebol and Ramp, now implemented by myself in Python, to the NLmaps corpus. These NLmaps results were previously published in Haas and Riezler (2016).

The structure of this chapter is as follows: Section 5.1 introduces the ramp loss objective and the algorithms Rebol (Subsection 5.1.2) and Ramp (Subsection 5.1.1). Section 5.2 details the experiments performed on the NLmaps corpus and includes an error analysis.

5.1. Objectives

For response-based on-policy learning we adopt the following general notation. We assume the existence of a model $\pi_w(y|x)$ parametrised by w that, given an input $x \in \mathbf{X}$, defines a probability distribution over all possible outputs $y \in \mathbf{Y}(x)$. Furthermore, we assume that it is possible to score every possible output y by employing an external metric $\delta(y)$, which returns a reward that quantifies how suitable it is to produce y given x. In the following, we define various objectives that can be used to update the parameters w of $\pi_w(y|x)$ by performing stochastic gradient descent for a loss function $\mathcal{L}$ using the update rule

$$\Delta w = -\eta \nabla_w \mathcal{L}, \tag{5.1}$$

where η is a suitably set learning rate. Finally, we denote the most likely output under the current model $\pi_w(y|x)$ as

$$\hat{y} = \arg\max_{y \in \mathbf{K}(x)} \pi_w(y|x), \tag{5.2}$$

where $\mathbf{K}(x)$ is a k-best list obtained via beam search and which is searched instead of $\mathbf{Y}(x)$ because $\mathbf{Y}(x)$ is too large.

The basis for our objectives, is the following general definition of the ramp loss:

$$\mathcal{L}_{\text{Ramp}} = -\left(\frac{1}{m} \sum_{t=1}^{m} \pi_w(y_t^+|x_t) - \frac{1}{m} \sum_{t=1}^{m} \pi_w(y_t^-|x_t) \right), \tag{5.3}$$

where y^+ is a **hope** output that we want to promote and y^- is a **fear** output that we want to demote. In this chapter, we assume $m = 1$, i.e. updates are made after each seen input. Intuitively, y^- should be an output with high probability but low reward from the external metric. Analogously, y^+ should be an output with a high probability and with a high reward from the external metric. Various definitions of y^+ and y^- lead to different objectives, which we explore later on.

Given a gold target $\bar{y}$, we can recover the loss of the structured perceptron (Collins, 2002) from Equation 5.3 by setting $y^+ = \bar{y}$ and $y^- = \hat{y}$ (McAllester and Keshet, 2011), where an update to w is only performed if $\bar{y} \neq \hat{y}$. However, the structured perceptron update itself is difficult to apply for SMT because the model might not be able to produce $\bar{y}$ (Och and Ney, 2002; Liang et al., 2006). In SMT the underlying model $\pi_w(y|x)$ is a log-linear model (see also Section 2.3, in particular Equation 2.17) that computes a probability distribution over the output space $\mathbf{Y}(x)$ given an input x:

$$\pi_w(y|x) = \frac{e^{w\phi(x,y)}}{\sum_{y' \in \mathbf{Y}(x)} e^{w\phi(x,y')}} = e^{w\phi(x,y)}/\mathcal{Z}, \tag{5.4}$$

where $\phi(x, y)$ is a feature vector for the pair (x, y). Depending on the employed features of the model, it might not be possible to produce the feature vector for the gold target $\bar{y}$, e.g. if the necessary translation rule does not exist. In such a situation we say that $\bar{y}$ is not reachable. Without this feature vector, the perceptron update cannot be performed. Instead, an external metric can be used to identify the most similar output to the gold target $\bar{y}$. This output's feature vector can act as a surrogate to the missing feature vector of $\bar{y}$.

In praxis, it can be expensive to compute the normalisation constant $\mathcal{Z}$ in Equation 5.4 and thus instead of $\pi_w(y|x)$ only the model scores, which are proportional to the probabilities, are used, i.e.

$$s_w(y|x) = w\ \phi(x, y). \tag{5.5}$$

If the model scores $s_w(y|x)$ are used in the ramp loss objective (Equation 5.3), then the update rule takes the following form:

$$\Delta w = \eta(\phi(x, y^+) - \phi(x, y^-)). \tag{5.6}$$

Gimpel and Smith (2012) are the first to employ a ramp loss to tune the parameters of an SMT system, leading to the Rampion algorithm. Their definitions of a hope translation y^+ and a fear translation y^- rely on computing the per-sentence BLEU (Nakov et al., 2012). The per-sentence BLEU score quantifies how close a translation y is to the reference translation $\bar{y}$. Gimpel and Smith (2012) transform this metric into a cost function, i.e. $c(\bar{y}, y) = 1 - \text{BLEU}(\bar{y}, y)$. This cost function is employed to the find hope and fear translation in the k-best list $\mathbf{K}(x)$. The translation with highest model score and simultaneously lowest cost becomes the hope output,

$$\begin{aligned} y^+ &= \arg\max_{y \in \mathbf{K}(x)} \left(s_w(y|x) - c(\bar{y}, y)\right) \\ &= \arg\max_{y \in \mathbf{K}(x)} \left(s_w(y|x) - (1 - \text{BLEU}(\bar{y}, y))\right). \end{aligned} \tag{5.7}$$

The fear translation is characterised by a high cost while also obtaining a high model score,

$$\begin{aligned} y^- &= \arg\max_{y \in \mathbf{K}(x)} \left(s_w(y|x) + c(\bar{y}, y) \right) \\ &= \arg\max_{y \in \mathbf{K}(x)} \left(s_w(y|x) + (1 - \text{BLEU}(\bar{y}, y)) \right). \end{aligned} \tag{5.8}$$

With y^+ and y^- defined, we can formulate the Rampion algorithm. The corresponding pseudo-code can be found in Algorithm 1. Iterating over the training set, the current model with parameters w receives an input string x for which it outputs the most likely translation $\hat{y}$. If this translation is identical to the gold translation $\bar{y}$, it becomes the hope translation y^+. Otherwise, it becomes the fear translation y^-. The missing translation is chosen according to Equations 5.7 and 5.8, respectively. A stochastic gradient descent update according to Equation 5.6 is then performed to the parameters w of the model.

Goldwasser and Roth (2013) employ a similar objective to improve their semantic parsing task. An English natural language instruction is transformed into a parse in a Machine Readable Language (MRL). The parse is executed and the result a is compared the gold result $\bar{a}$.[28] Based on this the following external reward metric can be defined:

$$\delta(y) = \begin{cases} 1 & \text{if } a = \bar{a} \\ 0 & \text{else.} \end{cases} \tag{5.9}$$

Using this feedback function, it is possible to improve the semantic parser using only the comparison to the gold result as a supervision signal. In this scenario the gold parse is never directly used, only indirectly via the feedback function. Given an input x, the most likely parse $\hat{y}$ under the current model parameters w is computed. If $\delta(\hat{y}) = 1$, then the parse is added to a set of valid parses for the current input x. If $\delta(\hat{y}) = 0$, then an update to w is performed. The concrete update depends on whether or not a parse with positive feedback has been previously found for x. If no previous parse with positive feedback exists, then simply all features of $\hat{y}$ are punished, i.e. $y^- = \hat{y}$ and $y^+ = \{\}$. If a positive parse has been previously found, then this parse becomes y^+. A corresponding update to w is then performed.

For our scenario of building a multilingual semantic parser, we use use the external reward metric defined in Equation 5.9. Once a translation is proposed, it is given to the semantic parser to produce a parse that can be executed against a database to receive an answer a. This answer a can then be compared to the gold answer $\bar{a}$. Via the feedback obtained from the metric of Equation 5.9, we ground the machine translation system and improve its parameters. By adjusting the machine translation system with this feedback, it can work better in conjunction with the semantic parser and consequently a higher final task performance can be achieved.

We propose two different ramp loss objectives, which mainly differ in the required gold targets. For successful grounding, each German question needs to have a corresponding gold

[28] Note that we use the term result rather than answer here because in the task of Goldwasser and Roth (2013) executed parses are not necessarily answers.

answer associated with it. This is the only annotation assumption the first algorithm, called Ramp, makes. The second algorithm, called Rebol, additionally assumes the existence of a corresponding English gold reference translation for each German question. Requiring two annotations for each German question is more expensive and might not be attainable in praxis. But if available, using both annotations in conjunction can provide valuable additional information for the machine translation system.

Algorithm 1 Pseudo-code for the Rampion algorithm.

Input: Training data of size n with inputs x, gold translations $\bar{y}$, cost function $c(\bar{y}, y)$, learning rate η
Ref $= \{\}$
repeat
  for $t = 1, \ldots, n$ **do**
    Receive input sequence x_t and gold translation $\bar{y}_t$
    Predict output sequence $\hat{y}_t$
    if $\hat{y}_t = \bar{y}_t$ **then**
      $y_t^+ = \hat{y}_t$
      Search for y_t^-
    else
      $y_t^- = \hat{y}_t$
      Search for y_t^+
    end if
    $w = w + \eta(\phi(x_t, y_t^+) - \phi(x_t, y_t^-))$
  end for
until Convergence

5.1.1. Ramp

The first algorithm, Ramp, assumes training data where each German question is paired with a corresponding gold answer. A translation is considered to successfully transfer the correct meaning if the parse produced by the semantic parser executes to the correct answer. Thus, the hope translation is defined as the most likely translation in the k-best list $\mathbf{K}(x)$ that obtains positive task feedback, i.e.

$$y^+ = \underset{y \in \mathbf{K}(x):\delta(y)=1}{\arg\max} \; s_w(x, y), \tag{5.10}$$

where $\delta(y)$ is the feedback function defined in Equation 5.9. The fear translation is analogously the most likely translation in the k-best list $\mathbf{K}(x)$ that obtains negative task feedback, i.e.

$$y^- = \underset{y \in \mathbf{K}(x):\delta(y)=0}{\arg\max} \; s_w(x, y). \tag{5.11}$$

Algorithm 2 presents the pseudo-code for Ramp. Iterating over the training set, the current model with parameters w receives an input string x for which it outputs the most likely translation $\hat{y}$. If this translation receives positive task feedback, i.e. $\delta(y) = 1$, then the translation becomes the hope translation y^+. It is also stored as a hope translation for the corresponding input x for future use. The k-best list is then traversed to find the highest scoring translation that does not obtain positive task feedback, i.e. $\delta(y) = 0$. Should the most likely translation $\hat{y}$ receive negative task feedback, it becomes the fear translation y^-. If there is a hope translation recorded for the input x from a previous epoch, then this hope translation is re-used. Otherwise, the k-best list is traversed to find the most likely translation that achieves positive task feedback. A stochastic gradient descent update is then performed to the parameters w of the model. If either y^- or y^+ cannot be computed, the example is skipped without an update being performed.

Algorithm 2 Pseudo-code for the Ramp algorithm.

Input: Training data of size n with inputs x and gold answers $\bar{a}$, the latter of which is employed in the task feedback function $\delta(y)$, learning rate η
Ref $= \{\}$
repeat
  for $t = 1, \ldots, n$ **do**
    Receive input sequence x_t and gold answer $\bar{a}_t$
    Predict output sequence $\hat{y}_t$
    Receive task feedback $\delta(\hat{y}_t) \in \{1, 0\}$
    if $\delta(\hat{y}_t) = 1$ **then**
      $y_t^+ = \hat{y}_t$
      Ref$[x_t] = \hat{y}_t$
      Search for y_t^-
      if y_t^- not found **then**
        Skip to next input sequence
      end if
    else
      $y_t^- = \hat{y}_t$
      if Ref$[x_t]$ **then**
        $y_t^+ =$ Ref$[x_t]$
      else
        Search for y_t^+
        if y_t^+ not found **then**
          Skip to next input sequence
        end if
      end if
    end if
    $w = w + \eta(\phi(x_t, y_t^+) - \phi(x_t, y_t^-))$
  end for
until Convergence

5.1.2. Rebol

Algorithm 3 Pseudo-code for the Rebol algorithm.

Input: Training data of size n with inputs x, gold answers $\bar{a}$ and gold translations $\bar{y}$, reward function $\delta(y)$ which employs $\bar{a}$, cost function $c(\bar{y}, y)$, learning rate η
Ref $= \{\}$
for $t = 1, \ldots, n$ **do**
  Receive input sequence x_t and gold translation $\bar{y}_t$
  Ref$[x_t] = [\bar{y}_t]$
end for
repeat
  for $t = 1, \ldots, n$ **do**
    Receive input sequence x_t and gold answer $\bar{a}_t$
    Predict output sequence $\hat{y}_t$
    Receive task feedback $\delta(\hat{y}_t) \in \{1, 0\}$
    Receive references Ref$[x_t]$
    if $\delta(\hat{y}_t) = 1$ **then**
      $y_t^+ = \hat{y}_t$
      Ref$[x_t] += \hat{y}_t$
      Search for y_t^-
      if y_t^- not found **then**
        Skip to next input sequence
      end if
    else
      $y_t^- = \hat{y}_t$
      if any reference r in Ref$[x_t]$ is reachable **then**
        $y_t^+ = r$
      else
        Search for y_t^+
        if y_t^+ not found **then**
          Skip to next input sequence
        end if
      end if
    end if
    $w = w + \eta(\phi(x_t, y_t^+) - \phi(x_t, y_t^-))$
  end for
until Convergence

The second algorithm, Rebol, assumes that for each German question both a corresponding gold answer and gold translation are available. With gold translations available, following Gimpel and Smith (2012), we can incorporate per-sentence BLEU (Nakov et al., 2012) in the objective as a cost function. Thus, Rebol defines the hope translation as the translation with highest score, but lowest possible loss and positive task feedback, i.e.

$$y^+ = \underset{y \in \mathbf{K}(x): \delta(y)=1}{\arg\max} \left(s_w(x, y) - (1 - \text{BLEU}(\bar{y}, y))\right). \tag{5.12}$$

Analogously, the fear translation achieves a high score, but also a high loss and receives negative task feedback, i.e.

$$y^- = \underset{y \in \mathbf{K}(x):\delta(y)=0}{\arg\max} \left(s_w(x,y) + (1 - \text{BLEU}(\bar{y}, y))\right). \tag{5.13}$$

Algorithm 3 presents the pseudo-code for Rebol. It closely follows the pseudo-code of Ramp, with the notable difference that each input string x immediately receives a valid reference translation in the form of the available gold translation. When new hope translations are found, they are appended to the list of reference translations for the corresponding input x. All per-sentence BLEU scores are then calculated on the basis of the entire set of available reference translations.

5.2. Empirical Results

We apply Ramp and Rebol to the NLmaps corpus. We translate all questions of NLmaps into German. These German questions serve as the input to an SMT system (see Section 2.3) trained with Cdec (Dyer et al., 2010). The English question is then transformed into a parse using the semantic parsing setup introduced in Section 4.1. Concretely, we use the model of row # 3 of Table 4.1, which employs the modifications `+cfg` and `+pass`. Additionally using sparse features (Simianer et al., 2012) would give a better semantic parser, but sparse features slow down the parsing process and we thus omit them for the sake of speed.

For the machine translation system on the other hand, we use both Cdec's standard features and sparse features. To obtain word alignments and a 5-gram language model we use the Common Crawl[29] (Smith et al., 2013) corpus. The initial weights for the different features are manually chosen. This model serves as the baseline and as the starting point for learning with the ramp loss objectives. Because the Rebol algorithm additionally assumes the existence of gold translations, we also compare it against the Rampion algorithm (Gimpel and Smith, 2012), which uses only the gold translations but not the gold answers to identify hope and fear translations.

We set the size of the k-best list that is searched for hope and fear translations to 100. In preliminary experiments, this value proved to offer a good trade-off between accuracy and speed. A larger k-best list rarely includes a previously unseen hope translation while significantly increasing the time spent fruitlessly searching.

The main evaluation measure is the F1 score as measured on the downstream task by comparing answers produced by the pipeline to gold answers. The F1 score is calculated as the harmonic mean of recall and precision. Recall is the number of correct answers divided by the number of answers in the underlying set and precision divides the number of correct answers by the number of answers that are non-empty strings. As a secondary metric, we report corpus-level BLEU scores (Papineni et al., 2002). Results may be found in Table 5.1. Significances between system differences are measured using an approximate randomisation test (Noreen, 1989).

[29] http://www.statmt.org/wmt13/training-parallel-commoncrawl.tgz, 1st September 2018

Ramp can improve the baseline system by 7.86 percentage points in F1 score by grounding it in the pipeline's final task. Even though it is not targeted directly, the BLEU score also increases by about 2.5 points. Both results are statistically significant compared to the baseline. The result successfully demonstrates that overall task performance can be increased if the first system in a pipeline setup is specifically tailored to perform well on the intended final task by leveraging the feedback given by it.

	method	P	R	F1	ΔF1	BLEU	ΔBLEU
1	Baseline	67.8	24.89	36.41		38.3	
2	Ramp	75.2	31.36	44.27^{1}	+ 7.86	40.85^{1}	+ 2.55
3	Rampion	78.21	38.75	$51.82^{1,2}$	+15.41	$51.82^{1,2}$	+13.52
4	Rebol	80.76	41.02	$\mathbf{54.41}^{1,2,3}$	+18.00	$\mathbf{51.88}^{1,2}$	+13.58

Table 5.1.: F1 scores on the NLMaps v2 test set for various response-based on-policy objectives. Best results are indicated in **bold face**. Statistical significance of system differences at $p < 0.05$ are indicated by algorithm number in superscript.

If gold targets is also available for the first system in the pipeline, then the performance increase over the baseline is larger. With gold translations available to improve the machine translation system, Rebol can significantly outperform the baseline system by 18 points in F1 and by 13.5 points in BLEU. Rampion does not have the feedback of the final task, i.e. gold answers, available. As a result, it falls a significant 2.59 points short in F1 score compared to the Rebol system. This re-affirms the importance of tailoring the first system in the pipeline towards the final task goal. As both Rebol and Rampion update their underlying models based on a per-sentence BLEU metric, it is not surprising that they reach near identical improvements in BLEU.

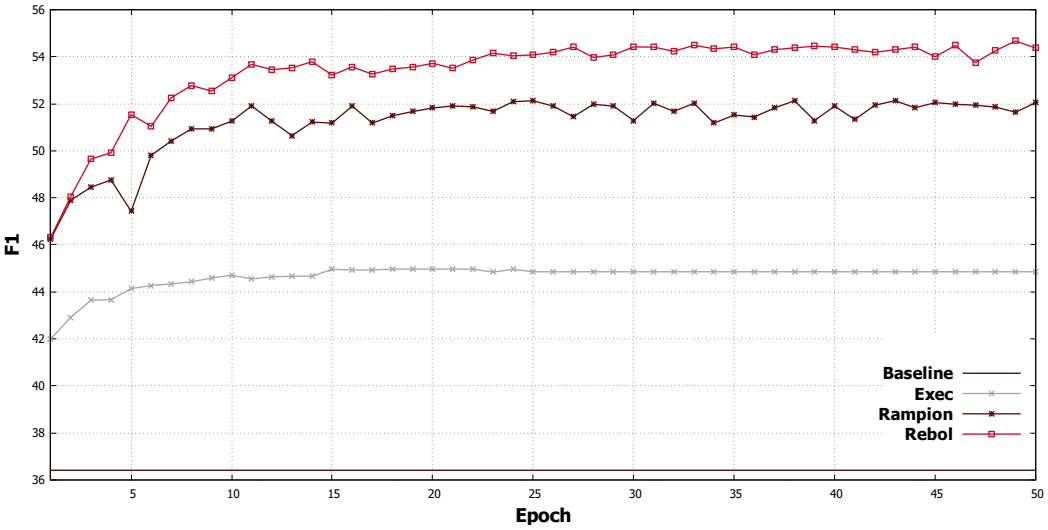


Figure 5.2.: F1 performance on the test set after each epoch for various response-based on-policy objectives.

We conclude that best performance is obtained if gold targets for both tasks are available. However, this assumption is likely too expensive to realise in praxis. Should only gold targets for the downstream task exist, a significant improvement can still be obtained over the baseline. But in either scenario, with or without gold translations available, the gold targets are a valuable learning signal. They can significantly improve the communication between models in a pipeline setup, which leads to an overall higher final task performance.

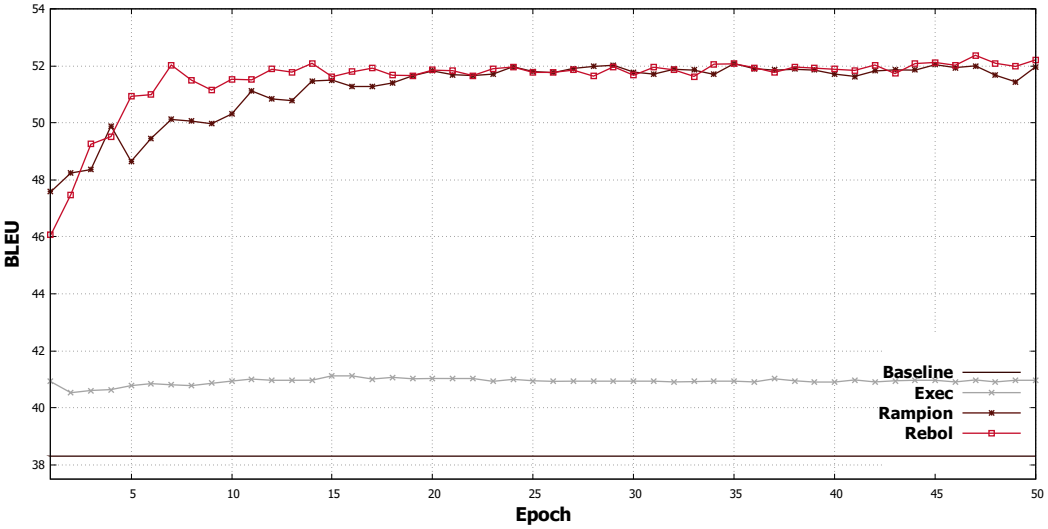


Figure 5.3.: BLEU performance on the test set after each epoch for various response-based on-policy objectives.

Finally, we present the learning curve over epochs as measured on the test set in Figure 5.2 for F1 and in Figure 5.3 for BLEU. Ramp converges quickly after about 15 epochs while the curves for Rampion and Rebol are slightly more erratic. At all times, Rebol stays numerically above Rampion in terms of F1 whereas the BLEU scores are very similar between the two.

error type	Perfect EN	Baseline	Ramp	Rampion	Rebol
OSM tag	42	$125_{\uparrow 198\%}$	$133_{\uparrow 6\%}$	$129_{\uparrow 3\%}$	$103_{\downarrow 18\%}$
Qtype	35	$59_{\uparrow 69\%}$	$51_{\downarrow 14\%}$	$61_{\uparrow 3\%}$	$55_{\downarrow 7\%}$
Wrong Dist.	5	$0_{\downarrow 100\%}$	$1_{\uparrow n.a.}$	$2_{\uparrow n.a.}$	$2_{\uparrow n.a.}$
Skeleton	36	$33_{\downarrow 8\%}$	$40_{\uparrow 21\%}$	$39_{\uparrow 18\%}$	$33_{\downarrow 0\%}$
Invalid Parse	209	$525_{\uparrow 151\%}$	$464_{\downarrow 12\%}$	$398_{\downarrow 24\%}$	$397_{\downarrow 24\%}$
Total	327	$742_{\uparrow 127\%}$	$689_{\downarrow 7\%}$	$629_{\downarrow 15\%}$	$590_{\downarrow 20\%}$

Table 5.2.: Error Analysis: The column titled "*Perfect EN*" shows which type of errors the semantic parsing model makes on the basis of the correct English questions. The column titled "Baseline" indicates which errors are made by the semantic parsing model if the German questions are translated with the baseline system. The percental change is with regards to the first column. The remaining columns give the error type count for the translation systems trained with Ramp, Rampion and Rebol, respectively. The percental change is with regards to the "Baseline" column.

5.2.1. Error Analysis

The semantic parser is not perfect. Thus, even correct English questions do not always lead to the correct answer. Using the English questions produced by the machine translation system, causes a further drop in performance. If the parsing model is given the set of correct English questions, it obtains an F1 score of 75.41%. Using the baseline's translations, this performance drops down to 36.41%. We give an analysis of what type of errors (as defined in Section 4.3.2) the semantic parsing model makes. Comparing the mistakes made on the perfect English translation to the mistakes made using the translations from the baseline system, allows us to identify which errors occur due to inferior translations. The results can be found in the first two columns of Table 5.2.

The largest numerical increase of errors occurs for the category Invalid Parse; it more than doubles. This suggests that translations are often missing crucial terms or a coherent structure that the semantic parser requires to formulate a correct NLmaps parse. The largest percental increase is observed for OSM tag errors. This indicates that the translation system does not choose the correct words, which the semantic parsing model has learnt to associate with OSM tags. Similarly, the Qtype errors also increase, albeit not as dramatically. Unlike the correct English questions, the imperfect translations seem to cause no Wrong Distance errors. On closer inspection, this implausible result can be explained: Instances that led to a parse with a wrong distance using the correct English question, become invalid parses when using the translated English equivalent. The slight reduction in Skeleton errors can be explained analogously.

Next, we analyse which type of errors the various algorithms, Ramp, Rampion and Rebol, are able to fix compared to the baseline. The results can be found in the last three columns of Table 5.2, where the percental changes are measured with regards to the baseline. Overall, Ramp exhibits an error reduction rate of 7%. The largest numerical reduction occurs for Invalid Parse errors which suggests that the answer feedback is able to guide the machine translation towards producing translations that conform to what the semantic parser expects. Ramp can also reduce the number of Qtype errors, but for the other three error categories we observe a slight increase. It is likely that some parses that were previously invalid are now valid but contain other errors. Consequently, the reduction of invalid parses is the reason of Ramp's success.

The reduction of Invalid Parse errors is even greater for both Rampion and Ramp and similar value. OSM tag errors are only reduced for the Rebol system, which suggests that it is important to leverage both gold translations and gold answers to reduce this type of error. Rebol, like Ramp, exhibits a reduction of Qtype errors, whereas Rampion makes more errors than the baseline system. This suggests that for this type of error, the gold answers are important. Overall, Rebol can reduce the number of errors made by 20% compared to the baseline.

Finally, we contrast some translations produced by the different systems. Table 5.3 gives examples of translations produced by Ramp that got positive feedback, whereas the translations produced by the baseline system did not get positive feedback. An additional column also lists the original English question. In line 1, the translations differ in the lexical word

	original EN	baseline	Ramp
1	What are the abandoned theme parks called?	What the deserted parks?	What the abandoned parks?
2	What is the Wikipedia page of the town hall in Heidelberg?	What is the Wikipedia page in Heidelberg?	What is the Wikipedia page of the town hall in Heidelberg?
3	Can I play tennis anywhere in Paris?	Can I play somewhere in Paris?	Can I play tennis somewhere in Paris?
4	Where in the south of Heidelberg are drinking water locations?	Where there is water in the south of Heidelberg?	Where in the south of Heidelberg are there drinking water?
5	At how many places can I rent a bike in Paris?	At many places in Paris can I rent a bike?	In how many places in Paris can I rent a bike?

Table 5.3.: Example translations produced by baseline and Ramp where the former receives negative feedback and the latter positive, along with the original English question.

choice of "*abandoned*" versus "*deserted*". The latter term never occurs in the training data of the semantic parser, thus it does not know to which OSM tag the term should be mapped to. Getting negative feedback for this question, allows the machine translation to adjust and propose "*abandoned*" instead which leads to positive feedback.

For the examples in line 2 and 3, the baseline neglects to translate a crucial word (" *town hall*" and "*tennis*", respectively). With these expressions missing, the semantic parser lacks the necessary words that could trigger the correct OSM tag. In line 4, the baseline makes a similar mistake. Here, the expression "*Trinkwasser*" (meaning "*drinking water*") is only partially translated with the word "*water*" while the term "*drinking*" is missing. Again Ramp is able to learn the correct expression via the answer-level feedback. For the example of line 5, the baseline translation seems closer to the original question at first glance, but it's missing the crucial word "*how*" to indicate the correct question type. Ramp finds an alternative translation that contains the word and can obtain positive feedback.

Table 5.4 presents examples produced by Rebol that obtain positive feedback whereas the Rampion counterparts do not. Akin to line 2 and 3 from Table 5.3, in line 1 the Rampion translation is missing a crucial word in the translation. Without the word "*well*" in the question, the correct OSM tag, "*man_made=water_well*", cannot be determined. Using answer-level feedback, Rebol can learn that the word "*well*" is crucial to obtain the correct OSM tag. Similarly, in line 2 the word "*book*" is missing from the Rampion translation to complete the expression "*book stores*".

Line 3 presents an example of a wrong lexical word choice for Rampion: the words "*port*" and "*harbour*" imply different OSM tags ("*landuse=port*" and "*landuse=harbour*", respectively). This distinction can only be learnt with the answer-level feedback that Rebol receives.

	original EN	Rampion	Rebol
1	Are there any water wells in Heidelberg?	Are there any water in Heidelberg?	Are there water wells in Heidelberg?
2	How many book stores does Paris have?	How many shops are there in Paris?	How many book shops are there in Paris?
3	Where can harbours be found in Edinburgh?	Where in Edinburgh are ports?	Where in Edinburgh are harbours?
4	Would you give me the location of a tennis court in Heidelberg please?	Would you call me the location of a tennis court in Heidelberg?	Would you give me the location of a tennis court in Heidelberg call?
5	Where are camp sites in Paris?	Where are campsites in Paris?	Where are camping sites in Paris?

Table 5.4.: Example translations produced by Rampion and Rebol where the former receives negative feedback and the latter positive, along with the original English question.

In line 4, the answer-level feedback helps to fix an ambiguity occurring in the German source word. Depending on context, "*nennen*" can be translated as "*give*" or "*call*". Rampion cannot determine the correct context and chooses the wrong translation ("*call*"), whereas Rebol can learn the correct translation ("*give*").

Finally, line 5 presents an example where Rampion's translation should also parse because it is a valid alternative translation. However, "*campsites*" is unknown to the semantic parsing model which has only seen "*camp sites*" previously. Rebol finds a translation containing "*camping sites*" which, once stemmed, maps to the same form as "*camp sites*". Here, the fault clearly lies with the semantic parser. However, this example demonstrates why it is important to adjust earlier models in a pipeline setup to perform well in conjunction with later models.

Conclusion

In our first scenario for response-based on-policy learning, we improved the first model in a two-model pipeline setup by grounding it in the final, downstream task. In particular, we showed this on the task of multilingual semantic parsing. German questions are translated into English by a machine translation system and the English translations are sent to a semantic parser that converts the English question into a parse. The parse can be executed against a database and the resulting answer is compared to the gold answer that the German question is accompanied by. This comparison can be used as a feedback signal to improve the machine translation system, which in turn leads to a better performance of the entire pipeline on the task.

We introduced two algorithms which employ ramp loss objectives. The first algorithm, Ramp, assumes that each German question has a corresponding gold answer. This allows us to ground the machine translation system in its final task. The machine translation system

learns to translate the German questions in such a way that the semantic parser can understand them and produces a parse that leads to the correct answer. The second algorithm, Rebol, additionally assumes that for each German question a corresponding English gold translation is available. Combining both sources of information results in a stronger learning signal and Rebol can outperform Ramp. However, requiring two gold annotations for every input could be expensive to obtain in praxis.

Both algorithms were employed on the NLmaps corpus for which the English questions were previously translated into German by the author. Ramp is able to tune the machine translation system so that the overall task improves by a significant 7.86 percentage points in F1 score over the baseline. This indicates that it is crucial to adjust a statistical model in a pipeline setup for the downstream task. Having two learning signals available, Rebol can further improve, achieving an increase of 18 points in F1 score over the baseline.

Rebol is also measured against a competitive algorithm, Rampion, which assumes that only gold translations, but no gold answers are available. Rampion falls a significant 2.59 points short on F1 score compared to Rebol. This showcases the importance of grounding the earlier model in the final task. Additionally, we showed in an error analysis which errors the various algorithms are able to fix by adjusting the translation that is given to the semantic parser. Finally, we finished with a few qualitative examples that concretely contrast successful translation from Ramp or Rebol compared to the baseline and Rampion, respectively.

In these experiments, the semantic parser was not modified. In our second scenario for response-based on-policy learning, we want to improve a semantic parser instead. We explore if comparing a suggested parse's answer to the gold answer can also be used as a feedback signal to directly improve the semantic parser. Simultaneously, we move to non-linear neural network models and investigate whether the ramp loss objective of Ramp, as well as other ramp loss objectives, can also be applied in this setting.

Chapter 6

Question-Answering: Grounding Semantic Parsing in Answer Feedback

Grounding a statistical model in a downstream task with corresponding downstream gold targets can provide a supervision signal that is easier to obtain than gold targets for direct supervision. This is most prominently the case for many domains in semantic parsing for question-answering (Clarke et al. (2010); Berant et al. (2013); Pasupat and Liang (2015); Rajpurkar et al. (2016); *inter alia*). Often the parses are highly complex and in a Machine Readable Language (MRL) that only a few experts have a knowledge of. In contrast to this, it is easier to ask crowd-source workers to provide the correct gold answer. With gold answers available, learning can proceed as follows: A semantic parser produces parses, each parse can be executed to obtain a corresponding answer and this answer can then be compared to the collected gold answer. On the basis of this comparison, feedback can be sent to the semantic parser and it serves as the learning signal to guide the parser towards producing correct parses. This grounding of a semantic parser in the downstream answer is our second scenario for which we employ response-based on-policy learning (see also Section 1.2). The setup is represented graphically in Figure 6.1.

Neural networks are the state-of-the-art models for sequence-to-sequence tasks. In the previous chapter, we employed response-based on-policy learning to improve linear models. Here, we apply our approach to neural semantic parsing. Neural networks are traditionally trained using the MLE objective (see Equation 2.28 in Chapter 2). However, with gold answers rather than gold parses available, the MLE objective is not directly applicable. Instead, we incorporate the feedback from the answer comparison into various objectives as an external metric.

For neural semantic parsing with question-answer pairs, such metric-augmented objectives have been explored in the context of Minimum Risk Training (MRT) (Liang et al., 2017; Guu et al., 2017),[30] as objectives inspired by REINFORCE (Liang et al., 2017; Mou et al.,

[30] Note that Liang et al. (2017) refer to their objective as an instantiation of REINFORCE, however they build an average over several outputs for one input and thus the objective more accurately falls under the heading of MRT.

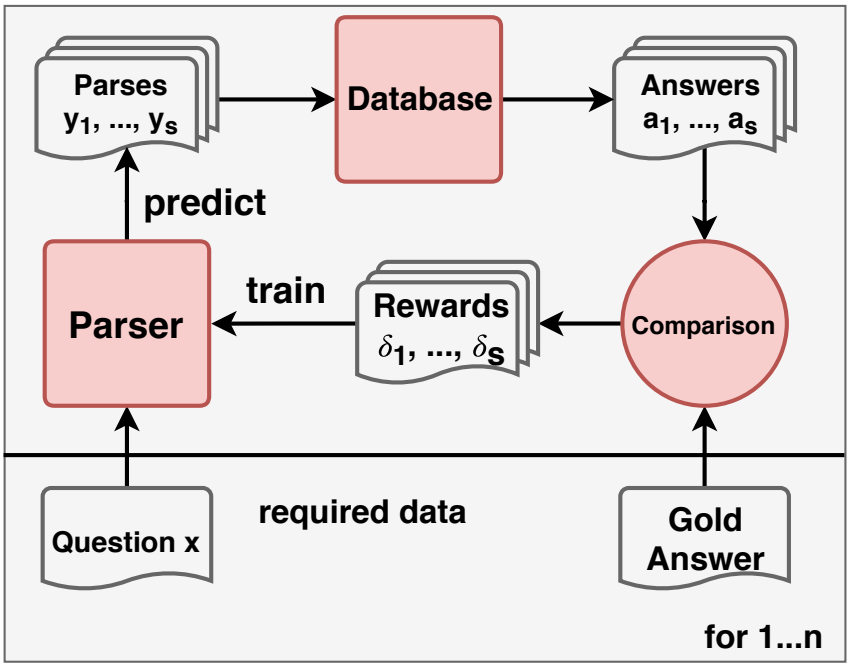


Figure 6.1.: Setup for training a semantic parser where gold answers are available but gold parses are missing.

2017; Guu et al., 2017) and other objectives based on classical structured prediction (Iyyer et al., 2017; Misra et al., 2018). Misra et al. (2018) are the first to compare several objectives for neural semantic parsing and they find that objectives employing structured prediction losses perform best. Similar objectives that incorporate different external metrics, have also been explored for neural models on other tasks, such as machine translation (Ranzato et al., 2016; Shen et al., 2016; Edunov et al., 2018; Wu et al., 2018), summarisation (Edunov et al., 2018; Arumae, 2018), reading comprehension (Choi et al., 2017; Yang et al., 2017) or image captioning (Rennie et al., 2017).

With gold answers rather than gold parses available, we operate in a weak supervision setting where it might be difficult to find parses that execute to the correct answer. To more effectively guide the semantic parser, we propose to not just encourage parses that lead to the correct answer, but to also discourage parses that lead to a wrong answer. This allows the parser to more effectively learn what distinguishes a good parse from a bad one. We refer to objectives that incorporate this idea as *bipolar* objectives. The bipolar idea occurs naturally in ramp loss objectives (Collobert et al., 2006), which we have already encountered in the previous chapter. In a ramp loss objective, a **hope** and a **fear** output are identified and promoted and demoted, respectively.

We lift the ramp loss objective from the previous chapter, Ramp, to neural models, as well as two further ramp loss objectives introduced in Gimpel and Smith (2012), namely Ramp1 and Ramp2. We compare these objectives to MRT, which, in a second step, we also modify to incorporate the bipolar principle. Conceptually, the ramp loss objectives are similar to the best performing objective of Misra et al. (2018), but their negative output is chosen based on

margin violation with regards to a pseudo-gold parse, whereas in our ramp loss objectives negative parses are identified independently.

Furthermore, we modify the Ramp objective to operate on a token level, leading to the objective Ramp+T. By operating at the token level, this objective can more effectively target which parts in a fear parse are wrong. With a comparison to the hope parse, it can acknowledge the parts that are correct in the fear parse and perform a more accurate update in which only incorrect tokens are discouraged. Additionally, this objective more accurately imitates the behaviour of the ramp loss when applied to linear models, where features are only updated if they differ in the hope and fear output. Finally, this objective allows us to directly manipulate tokens as the MLE objective does, but which is not available in scenarios with no direct supervision.

Concretely, we train a semantic parser using the NLmaps v2 corpus and the neural sequence-to-sequence framework Nematus (Sennrich et al., 2017). We assume a small amount of question-parse pairs are available to train an initial neural semantic parser. Further data is only available in the form of question-answer pairs, with which we want to further improve the semantic parser.[31] Our experiments show that the bipolar ramp loss objective Ramp+T significantly outperforms all other objectives and improves the baseline semantic parser by over 12 percentage points in F1 score. Furthermore, we give a detailed analysis of why the sequence-level Ramp objective is superior to MRT.

The main contributions of this chapter are as follows:

- Definition of a token-level ramp loss objective.
- Implementation of the ramp loss objectives in Nematus.
- Application of implemented objectives to semantic parsing on the NLmaps v2 corpus and subsequent error analysis.
- Identification of the bipolar principle which is crucial for successful objectives. This principle was discovered together with Laura Jehl who ran corresponding experiments on machine translation tasks.

The work presented in this chapter has been previously compiled in Jehl et al. (2019), which is currently under submission. My co-author, Laura Jehl, has applied the same objectives as presented in this chapter to machine translation tasks. The result, that a bipolar objective is crucial for success, is also confirmed by my colleague on weakly supervised domain adaptation experiments for machine translation.

The structure of this Chapter is as follows: First, we present all relevant objectives in Section 6.1, where Section 6.1.1 covers MRT, Section 6.1.2 the ramp loss objectives and Section 6.1.3 the token-level ramp loss objective. Second, we present empirical results in Section 6.2, which we conclude with an error analysis.

[31] Due to the nature of the domain, for NLmaps it is harder to obtain gold answers than gold parses. This is often not the case for other domains, hence we assume that only a small amount of question-parse pairs are available to investigate the type of scenario where question-answer pairs are easier to obtain.

6.1. Objectives

For convenience, we repeat here the general notation introduced in the previous chapter for response-based on-policy learning: We assume the existence of a model $\pi_w(y|x)$ parametrised by w that, given an input $x \in \mathbf{X}$, defines a probability distribution over all possible outputs $y \in \mathbf{Y}(x)$. Furthermore, we assume that it is possible to score every possible output y by employing an external metric $\delta(y)$, which returns a reward that quantifies how suitable it is to produce y given x. In the following, we define various objectives that can be used to update the parameters w of $\pi_w(y|x)$ by performing gradient descent for a loss function $\mathcal{L}$ using the update rule $\Delta w = -\eta \nabla_w \mathcal{L}$. In contrast to the previous chapter, we assume here that updates are made on the basis of minibatches of size m. Finally, we denote the most likely output under the current model $\pi_w(y|x)$ as

$$\hat{y} = \arg\max_{y \in \mathbf{K}(x)} \pi_w(y|x), \tag{6.1}$$

where $\mathbf{K}(x)$ is a k-best list that is searched using beam search because $\mathbf{Y}(x)$ is too large.

6.1.1. Minimum Risk Training (MRT)

We compare our ramp loss objectives to MRT (Smith and Eisner, 2006; Shen et al., 2016). Given an input x_t, r outputs are sampled from the model distribution and updates are performed based on the following objective:

$$\mathcal{L}_{\text{MRT}} = -\frac{1}{m} \sum_{t=1}^{m} \frac{1}{r} \sum_{s=1}^{r} \pi_w(y_{t,s}|x_t) \left(\delta(y_{t,s}) - b(x_t)\right), \tag{6.2}$$

where $\delta(y_{t,s})$ is the reward returned for $y_{t,s}$ by an external metric and $b(x_t)$ is a baseline computed by sampling r' outputs y'_t from the model distribution,

$$b(x_t) = \frac{1}{r'} \sum_{s'=1}^{r'} \delta(y'_{t,s'}). \tag{6.3}$$

The pseudo-code for this objective is presented in Algorithm 4.

6.1.2. Sequence-Level Ramp Loss

The ramp loss objectives in this chapter can be formulated in the generalised form introduced in the previous chapter (see Equation 5.3), here repeated for convenience:

$$\mathcal{L}_{\text{Ramp}} = -\left(\frac{1}{m} \sum_{t=1}^{m} \pi_w(y_t^+|x_t) - \frac{1}{m} \sum_{t=1}^{m} \pi_w(y_t^-|x_t)\right), \tag{6.4}$$

Algorithm 4 Pseudo-code for the MRT objective.

Input: Training data with inputs x and gold answers $\bar{a}$, the latter of which is employed in the task feedback function $\delta(y)$, minibatch size m, number of samples r, number of samples r' for the baseline , learning rate η, initial policy π_w
repeat
  $y_{avg} = \text{None}$
  for $t = 1, \ldots, m$ **do**
    Receive input sequence x_t and gold answer $\bar{a}_t$
    Sample r' output sequences y'_t
    Compute $b(x_t) = \frac{1}{r'} \sum_{s'=1}^{r'} \delta(y'_{t,s'})$
    Obtain r output sequences y_t and their probabilities $\pi_w(y_t|x_t)$
    if $m = 1$ **then**
      $y_{avg} = \frac{1}{r} \sum_{s=1}^{r} \pi_w(y_{t,s}|x_t) \left(\delta(y_{t,s}) - b(x_t)\right)$
    else
      $y_{avg} += \frac{1}{r} \sum_{s=1}^{r} \pi_w(y_{t,s}|x_t) \left(\delta(y_{t,s}) - b(x_t)\right)$
    end if
  end for
  $w = w + \eta \, \nabla_w \frac{1}{m} \sum_{t=1}^{m} y_{avg}$
until Stopping Criterion Reached

Name	y^+	y^-
Ramp	$\arg\max_{y \in \mathbf{P}(x)} \pi_w(y\|x)$	$\arg\max_{y \in \mathbf{N}(x)} \pi_w(y\|x)$
Ramp1	$\hat{y}$	$\arg\max_{y \in \mathbf{N}(x)} \pi_w(y\|x)$
Ramp2	$\arg\max_{y \in \mathbf{P}(x)} \pi_w(y\|x)$	$\hat{y}$

Table 6.1.: Formulation of three ramp loss objectives for semantic parsing. We abbreviate $\mathbf{P}(x) = \mathbf{K}(x) : \delta(y) = 1$ and $\mathbf{N}(x) = \mathbf{K}(x) : \delta(y) = 0$.

where y^+ is a **hope** output that we want to promote and y^- is a **fear** output that we want to demote. Intuitively, y^- should be an output with high probability but low reward from the external metric. Analogously, y^+ should be an output with a high probability and with a high reward from the external metric.

The exact definitions of y^- and y^+ depend on the underlying task. In semantic parsing for question answering, natural language questions are mapped to machine readable parses. Such a parse y can be executed against a database, which returns an answer a that can be compared to the gold answer $\bar{a}$ and the following metric can be defined:

$$\delta(y) = \begin{cases} 1 & \text{if } a = \bar{a} \\ 0 & \text{else.} \end{cases} \tag{6.5}$$

Adopting the definition of Ramp from Chapter 5, y^+ is defined as the most probable output in the k-best list $\mathbf{K}(x)$ that leads to the correct answer, i.e. where $\delta(y) = 1$. In contrast, y^- is defined as the most probable output in $\mathbf{K}(x)$ that does not lead to the correct answer,

Algorithm 5 Pseudo-code for the RAMP objective with mini-batches.

Input: Training data with inputs x and gold answers $\bar{a}$, the latter of which is employed in the task feedback function $\delta(y)$, minibatch size m, learning rate η, initial policy π_w
repeat
  $hope = [\,]$
  $fear = [\,]$
  for $t = 1, \ldots, m$ **do**
    Receive input sequence x_t and gold answer $\bar{a}_t$
    Obtain output sequence $\hat{y}_t$ and its probability $\pi_w(\hat{y}_t|x_t)$
    Receive task feedback $\delta(\hat{y}_t) \in \{1, 0\}$
    if $\delta(\hat{y}_t) = 1$ **then**
      Search for y_t^- and get its probability $\pi_w(y_t^-|x_t)$
      if y_t^- is found **then**
        $hope$ += $\pi_w(\hat{y}_t|x_t)$
        $fear$ += $\pi_w(y_t^-|x_t)$
      end if
    else
      Search for y_t^+ and get its probability $\pi_w(y_t^+|x_t)$
      if y_t^+ is found **then**
        $hope$ += $\pi_w(y_t^+|x_t)$
        $fear$ += $\pi_w(\hat{y}_t|x_t)$
      end if
    end if
    $y_{avg}^+ = \frac{1}{m} \sum_{t=1}^{m} hope_t$
    $y_{avg}^- = \frac{1}{m} \sum_{t=1}^{m} fear_t$
    $w = w + \eta \, \nabla_w (y_{avg}^+ - y_{avg}^-)$
  end for
until Stopping Criterion Reached

i.e. where $\delta(y) = 0$. If y^+ or y^- are found, the parse is cached as a hope or fear output, respectively, for the corresponding input x. If at a later point y^+ or y^- cannot be found in the current k-best list, then previously cached outputs are accessed instead. Should no cached output exist, the corresponding sample is skipped.

Following Gimpel and Smith (2012), we define two further ramp loss objectives: Setting the hope to be the most likely output, i.e. $y^+ = \hat{y}$, leads to RAMP1 and setting the fear to be the most likely output, i.e. $y^- = \hat{y}$, leads to RAMP2. The overview of the three objectives can be found in Table 6.1 and the pseudo-code is presented in Algorithm 5.

6.1.3. Token-Level Ramp Loss

For models that decompose over tokens, we can also formulate a token-level ramp loss objective. To be able to adjust individual tokens, we move to log probabilities, so that the sequence decomposes as a sum over individual tokens and it is possible to ignore tokens while encouraging or discouraging others. This leads to the Ramp-T objective:

$$\mathcal{L}_{\text{Ramp+T}} = -\left(\frac{1}{m}\sum_{t=1}^{m}\sum_{j=1}^{|y_t^+|}\tau_{t,j}^+ \log \pi_w(y_{t,j}^+|y_{t,<j}^+, x_t) - \frac{1}{m}\sum_{t=1}^{m}\sum_{j=1}^{|y_t^-|}\tau_{t,j}^- \log \pi_w(y_{t,j}^-|y_{t,<j}^-, x_t)\right), \tag{6.6}$$

where $y_{t,<j} = y_{t,1}, y_{t,2} \dots y_{t,j-1}$ and where $\tau_{t,j}^+$ and $\tau_{t,j}^-$ are set to 0, 1 or -1 depending on whether the corresponding token $y_{t,j}^+/y_{t,j}^-$ should be left untouched, encouraged or discouraged, respectively. Concretely, we define:

$$\tau_{t,j}^+ = \begin{cases} 0 & \text{if } y_{t,j}^+ \in y^- \\ 1 & \text{else} \end{cases} \tag{6.7}$$

and

$$\tau_{t,j}^- = \begin{cases} 0 & \text{if } y_{t,j}^- \in y^+ \\ -1 & \text{else.} \end{cases} \tag{6.8}$$

With this definition, tokens that appear in both y^+ and y^- are left untouched, whereas tokens that appear only in the hope output are encouraged and tokens that appear only in the fear output are discouraged. This contrast allows the model to distinguish between a good and a bad output on a more fine-grained level than the sequence-level objectives can.

Note that all ramp loss objectives are naturally bipolar, whereas MRT objectives are only bipolar if the metric assigns negative scores for bad outputs.

6.2. Empirical Results

Our experiments are conducted on the NLmaps v2 corpus (Lawrence and Riezler, 2018) and we use the sequence-to-sequence neural network package Nematus (Sennrich et al., 2017), which follows the setup outlined in Section 2.4. For each question, the NLmaps v2 corpus provides both gold parses and gold answers. We take a random subset of 2,000 question-parse pairs to train an initial model π_w with an MLE objective (see Equation 2.28), where parses are split into individual tokens by taking a pre-order traversal of the original tree structure (see Listing 3.4 in Section 3.2). A further 1,843 and 2,000 instances of the corpus are retained for development and test set, respectively. For the remaining 22,766 data points we assume that no gold parses exist and only gold answers are available. With the gold

		m	% F1	Δ
1	Baseline		57.45	
2	MRT	1	63.60±0.02	+ 6.15
3	Ramp1	80	60.50±0.01	+ 3.05
4	Ramp2	80	64.22±0.00	+ 6.77
5	Ramp	80	69.03±0.04	+11.58
6	Ramp+T	80	**69.87**±0.02	+12.42

Table 6.2.: F1 scores on the NLMaps v2 test set for various response-based on-policy objectives. Results are averaged over two independent runs. m is the minibatch size. The best result is indicated in **bold face**. All models are statistically significant from each other at $p < 0.01$, except the pair $(2, 4)$.

answers as a guide, the initial model π_w is further improved using the objectives outlined in Section 6.1.

The model has 1,024 hidden units and word embeddings of size 1,000. The optimal learning rate for SGD was chosen in preliminary experiments on the development set and is set to 0.1. Gradients are clipped if they exceed a value of 1.0 and the sentence length is capped at 200. In the case of the MRT objectives, we set $r = r' = 10$. For the Ramp objectives the size of the k-best list $\mathbf{K}$ is 10. For objectives with minibatches, the size of a minibatch is $m = 80$ and validation on the development set is performed after every 100 updates. For objectives where updates are performed after each seen input, the validation is run after every 8,000 updates. The highest evaluation score on the development set determines the stopping point and results are reported on the test set. Each experiments is run for 30 validations or 30 days, whichever occurs first.[32]

For validation and at test time, the most likely parse is obtained after a beam search with a beam of size 12, and this parse is executed against the database to retrieve its corresponding answer. We define recall as the percentage of completely correct answers divided by the set size and precision as the percentage of correct answers out of the set of answers with non-empty strings. The harmonic mean of recall and precision constitutes the F1 score which is the evaluation measure we report in our experiments. Statistical significance between models is measured using an approximate randomization test (Noreen, 1989).

Results using the various ramp loss objectives as well as MRT can be found in Table 6.2. MRT can outperform the baseline by about 6 percentage points in F1 score. Ramp1 performs worse than MRT but can still significantly outperform the baseline by 3.05 points in F1 score. Ramp2 performs better than Ramp1, but cannot significantly outperform MRT. In contrast to this, by carefully selecting both a hope and fear parse, Ramp achieves a significant further 5.43 points in F1 score over MRT.

By incorporating token-level feedback, our novel objective Ramp+T outperforms all other models significantly and beats the baseline by over 12 points in F1 score. Compared to Ramp, Ramp+T can take advantage of the token-level feedback, which allows a model to determine

[32] The 30 day mark was only hit by Ramp2 which had to execute more parses looking for a correct one than the other models, which causes a significant slowdown. This is most likely because it sets the most likely parse, which is often already good, as the fear parse, i.e. $y^- = \hat{y}$.

		m	% F1	Δ
1	Baseline		57.45	
2	MRT	1	63.60±0.02	+ 6.15
3	MRT+NEG	1	65.93±0.16	+ 8.48
4	RAMP M=1	1	66.78±0.21	+ 9.33
5	RAMP	80	69.03±0.04	+11.58

Table 6.3.: F1 scores on the NLMAPS V2 test set for the RAMP and the MRT objectives as well as two further objectives, which help crystallise the difference between the two former objectives. Results are averaged over two independent runs. m is the minibatch size. All models are statistically significant from each other at $p < 0.01$, except the pair $(3, 4)$.

which tokens in the hope output are instrumental to obtain a positive reward but are missing in the fear output. Analogously it is possible to identify which tokens in the fear output lead to an incorrect parse, rather than also punishing the tokens in the fear output which are actually correct.

We want to investigate why RAMP performs better than MRT. To this end, we introduce the objective MRT+NEG : it modifies the feedback for parses with a wrong answer to be -1 rather than 0, which resembles the fear output that is discouraged in the RAMP objective. With this change, the MRT objective incorporates bipolar supervision as it now actively discourages wrong parses. With this modification, MRT+NEG can significantly outperform MRT by 2.33 points in F1 score (see Table 6.3). This showcases the importance of employing bipolar supervision and it constitutes an important finding compared to previous approaches (Liang et al., 2017; Misra et al., 2018), where the feedback is defined to lie in the range of $[0, 1]$.

However, MRT+NEG still falls short of RAMP by 3.1 points in F1 score. There is one further crucial difference between MRT+NEG and RAMP: the MRT objective uses $r = 10$ outputs per input in the update, whereas the RAMP objectives only update with regards to two outputs (the hope and the fear output) per input. Consequently, the MRT objectives take up more valuable RAM memory on a GPU per input. This leads to a trade-off between using more outputs per input or calculating an average over several inputs. To investigate how important it is to calculate an average over several inputs, we set the minibatch size of the RAMP objective to 1. This objective, RAMP M=1, obtains a lower F1 score than RAMP (see Table 6.3), which showcases the importance of calculating an average over several inputs. Its F1 score is in fact on par with the MRT+NEG objective, which suggests that no benefit is gained from calculating an average over outputs for one input if one carefully chooses an appropriate hope and fear output.

Finally, Figure 6.2 reports results on the test set at every validation point for the best two ramp loss objectives, RAMP and RAMP+T, as well as MRT and the MRT+NEG. At all points the ramp loss objectives significantly lie above both MRT objectives. RAMP+T lies significantly above RAMP at nearly all times, particularly at the later validation points.

To summarise, RAMP can attribute its success to two factors: First, it discourages parses that receive a wrong answer rather than ignoring them as MRT does. Second, it calculates an average over inputs rather than an average over the outputs of one input. Finally, further

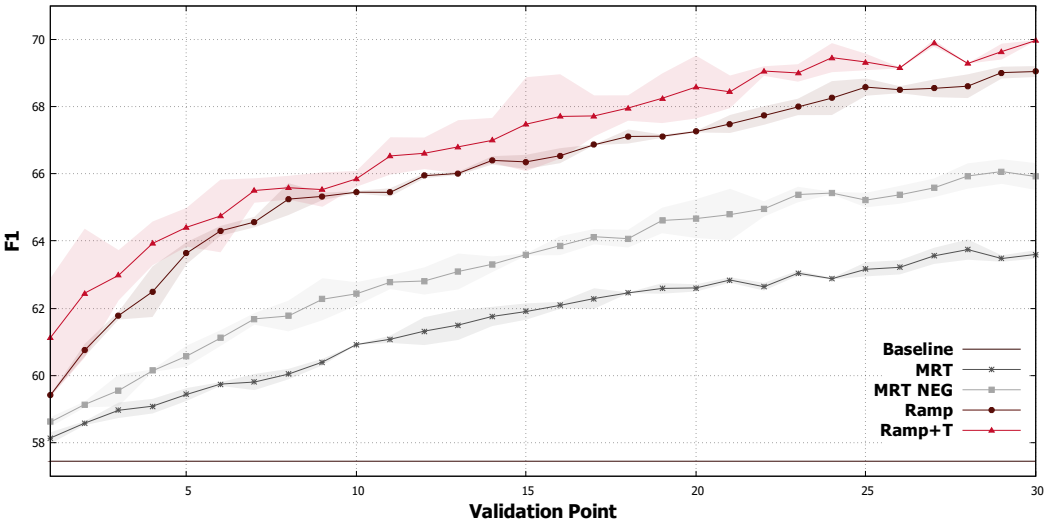


Figure 6.2.: F1 performance on the test set at the validation points for the objectives MRT, MRT+neg, Ramp and Ramp+T.

performance gains can be obtained by employing the token-level objective Ramp+T. These results also concur with weakly supervised domain adaptation experiments for machine translation conducted by my colleague, Laura Jehl.

	Baseline	MRT	Ramp	Ramp+T
OSM tag	1,248	1,071 $_{\downarrow 14.18\%}$	864 $_{\downarrow 30.77\%}$	717 $_{\downarrow 42.55\%}$
Qtype	135	113 $_{\downarrow 16.3\%}$	101 $_{\downarrow 25.19\%}$	93 $_{\downarrow 31.11\%}$
Wrong Dist.	3	4 $_{\uparrow 33.33\%}$	5 $_{\uparrow 66.67\%}$	7 $_{\uparrow 133.33\%}$
Skeleton	70	62 $_{\downarrow 11.43\%}$	181 $_{\uparrow 158.57\%}$	256 $_{\uparrow 265.71\%}$
Invalid Parse	7	7 $_{\rightarrow 0.0\%}$	7 $_{\rightarrow 0.0\%}$	22 $_{\uparrow 214.29\%}$
Total	1,463	1,255 $_{\downarrow 14.22\%}$	1,101 $_{\downarrow 24.74\%}$	1,093 $_{\downarrow 25.29\%}$

Table 6.4.: Overview of which type of errors the baseline makes and the percental in- or decrease with regards to the baseline for the objectives MRT, Ramp and Ramp+T.

6.2.1. Error Analysis

Following the error analysis setup introduced in Section 4.3.2, we present the type of errors made by the baseline model, as well as the models produced by the objectives MRT, Ramp and Ramp+T in Table 6.4.[33]

The MRT model achieves its largest numerical decrease in OSM tag errors and the largest percental decrease occurs for errors of the type Qtype. Overall, this model can reduce the total number of errors by 14.22% compared to the baseline. The Ramp model records a larger decrease in OSM tag and Qtype errors than the MRT model, but also observes a large increase of Skeleton errors. However, the reduction of errors outweighs, leading to a 24.74% decrease in overall errors compared to the baseline. The trend of Ramp is exacerbated for the Ramp+T model: compared to Ramp it has a lower count of OSM tag and Qtype errors, but at the same time an even higher count of Skeleton errors. With 25.29% less errors than the baseline model it overall reaches a slightly higher error reduction rate than Ramp.

For both Ramp and Ramp+T, the increase of Skeleton errors is largely due not producing the operator `around` where it would be required to form a correct parse.[34] In detail, this for example means that for the natural language question "*Gas Stations near Résidence de l'Abbaye de Roseland in Nice*", instead of the correct parse,

```
query(around(center(area(keyval('name','Nice')),
nwr(keyval('name','Résidence de l'Abbaye de Roseland'))),
search(nwr(keyval('amenity','fuel'))),
maxdist(DIST_INTOWN)),qtype(latlong)),
```

Ramp wrongly produces the parse

```
query(area(keyval('name','Nice')),
nwr(keyval('amenity','fuel')), qtype(latlong)).
```

For Ramp, the loss of the `around` operator causes 144 out of 181 Skeleton errors (83.7%) and for Ramp+T it is 212 out of 256 (89.8%). In contrast, for MRT the issue of the missing `around` operator occurs only 15 out of 62 times (27.4%). However, removing the instances of the missing `around` operator, Ramp and Ramp+T retain only 37 and 44 Skeleton errors, respectively, which is lower than the Skeleton errors recorded for the baseline (70) and MRT (62). Despite the frequent loss of the `around` operator, Ramp and Ramp+T still correctly use the `around` operator in the majority of instances. Out of 891 instances in the test set where the `around` operator is required to formulate the correct parse, Ramp correctly produces it 747 times (83.83%) and Ramp+T 680 times (76.31%).

We would like to understand why the ramp loss objectives develop the issue of not producing the operator `around` where it would be required. We conjecture that it is harder to find a hope parse if a correct parse requires the `around` operator because these parses are longer and more complex than parses without the `around` operator.

[33] Errors are reported averaged over both independent runs and values are rounded to the nearest integer.

[34] Note that without the `around` operator, the correct answer cannot be reached, except for rare cases of spurious parses. Spurious parses are parses that do not correctly represent the meaning of the associated question but by coincidence evaluate to the correct answer.

To investigate this, we first analyse the set of gold parses that we do have available for the NLmaps v2 training data, even though we do not employ them in our experiments. This shows that in 43.3% of the cases, the correct parse requires the use of the `around` operator. However, analysing the log file of one Ramp experiment, shows that only 17% of found hope parses contain the `around` operator. This discrepancy indicates that parses with the `around` operator are drastically under-represented in updates for the ramp loss objectives, as instances are skipped if no hope can be found. Consequently, the loss of the `around` operator could be a case of overfitting on parses without the `around` operator for the ramp loss objectives. As the reduction in the error types OSM tag and Qtype outweighs this increase, this effect cannot be caught by choosing the highest evaluation score on the development set. One possible alternative to explore in the future, could be to specifically oversample instances that contain the `around` operator and for which hope parses, i.e. correct parses, have been found previously.

Conclusion

For many domains of semantic parsing for question-answering, it is too expensive to obtain gold parses for direct supervision because the underlying MRL is only known to a few experts. Instead, it is easier to collect gold answers. We used such gold answers in a response-based on-policy learning setup to ground the semantic parser in this downstream, weak supervision. We considered the following scenario: we trained a baseline neural semantic parser on a small subset of the NLmaps v2 using question-parse pairs and assumed that all further training data is only available in the form of question-answer pairs. When only question-answer pairs are available, it is not possible to apply the usual MLE objective to train the neural network. Instead, we employ metric-augmented objectives, where our metric is defined on the basis of whether a parse produced by the model leads to the correct answer or not.

To more effectively help a model to find correct parses, we proposed to not just encourage parses that lead to the correct answer, but to also discourage parses that lead to the wrong answer. We call objectives incorporating this idea bipolar. Specifically, we investigated various ramp loss objectives, which naturally incorporate this bipolar principle. Next to the ramp loss objectives Ramp1 and Ramp2, we also revisited the objective Ramp from the previous chapter. Ramp losses define a hope and a fear parse, which are promoted and demoted, respectively. For Ramp, the hope parse for an input question is the parse that has the highest probability while executing to the correct answer. On the other hand, the fear parse has the highest probability while resulting in a wrong answer. Ramp1 uses the same fear definition, but sets the hope to be the most likely output. In contrast, Ramp2 sets the fear to be the most likely output and uses the same hope definition as Ramp1.

The ramp loss objectives were compared to MRT, which does not incorporate the bipolar principle and instead ignores incorrect parses. We showed that Ramp outperforms all other sequence-level objectives because it can contrast a hope and fear parse. Additionally, its procedure to identify the hope and fear parses is more effective than the procedures of the other two ramp loss objectives, Ramp1 and Ramp2.

We also introduced a token-level variant of Ramp, called Ramp+T. This objective outperforms all other objectives because it can better distinguish between a good and a bad parse by comparing them at the token level. Via a comparison to the hope parse, Ramp+T can ensure that tokens in a fear parse are only punished if they are incorrect, i.e. they do not appear in the hope parse. At the same time it ensures specifically that the tokens in the hope parse, but not in the fear parse, are promoted. With this more fine-grained contrasting of hope and fear parse, this objective significantly outperforms all other objectives.

In an attempt to analyse the usefulness of the bipolar principle further, we modified MRT to punish, instead of ignore, parses that lead to an incorrect answer. The resulting objective, MRT+neg, can significantly outperform the original objective, but still falls short of reaching the performance of sequence-level Ramp. We were able to identify that the remaining performance difference is due to MRT's approach to compute an average over several outputs for one input. Instead, Ramp computes an average over inputs, for each of which only two outputs - a hope and a fear parse - are required. In an experiment, we showed that it's more important to compute such an average over inputs rather than an average over outputs for one input, hence explaining the superiority of Ramp over MRT.

Similar conclusions can be drawn for weakly supervised domain adaptation experiments for machine translation conducted by my colleague, Laura Jehl. In these experiments, Ramp+T is also the best objective and MRT can only improve upon the baseline if the augmented metric is modified to punish, rather than ignore, bad outputs.

Part II Conclusion

Response-based on-policy learning leverages feedback obtained from grounding a model in a downstream task. With this approach, feedback for arbitrarily many model outputs can be obtained and outputs that lead to positive task feedback can be discovered. It is possible to find several successful outputs, which can lead to more robust learning. Furthermore, in the case of linear models, gold targets might not be reachable and response-based on-policy learning allows us to find surrogate gold targets that can be produced by the model.

We successfully employed the approach to two NLP sequence-to-sequence tasks. In a first task, we tuned a linear machine translation model to work well in conjunction with a semantic parser for a multilingual semantic parsing pipeline. Additionally, response-based on-policy learning alleviates the need for direct-supervision gold targets in scenarios where it is easier to obtain downstream gold targets. We explored this setup in second task: we improved a neural semantic parser in a scenario where input questions are paired with gold answers, rather than gold parses, because gold answers are easier to obtain for many domains.

Together, the results from Chapter 7 and Chapter 8 show that response-based on-policy learning is very effective and leads to significant improvements on both tasks. But ultimately response-based on-policy learning still requires the gold targets of the downstream task, which might also be too expensive to obtain. This is for example the case for the OSM domain. Gold parses are expensive to obtain because the underlying MRL is only known

to a handful of experts. However, gold answers for most questions are outright impossible to acquire from humans because the answer sets can be open-ended, fuzzy or too large to enumerate without error or within reasonable time constraints. Thus, we next explore an approach where no gold targets are required for learning.

Part III.

Counterfactual Off-Policy Learning

Chapter 7

Machine Translation: Learning from Deterministic Logs with Bandit Feedback

Machine-learnt statistical models have increasingly become the core of various online applications, such as information retrieval, ad placement, automatic machine translation and question-answering systems. Our second approach to learn from feedback, counterfactual off-policy learning (see also Section 1.3), assumes that such a model is deployed in a production system. User-system interactions are logged and used to improve the deployed model or another model. For every interaction, the context or user input to the model is recorded in conjunction with the output that the model produces. For each such pair, corresponding feedback is elicited implicitly or explicitly from the user and recorded. Unlike gold-labelled data for supervised learning, this type of data is cheap to collect and plentiful. Developing methods to learn from such data is consequently an interesting endeavour for industrial online systems.

Learning from logged data is considerably more challenging compared to learning from supervised data. Because no gold targets are available, feedback for the output has to be collected directly from users. Consequently the feedback is subjective and might be wrong. This can lead to a very noisy data set for learning. Furthermore, the feedback is only available for one system output and no other output can be judged. Finally, as the output is chosen by the deployed model, the resulting log is biased towards the choices of the model.

A setup where feedback is only available for one output is referred to as a **bandit** learning setup (Bubeck and Cesa-Bianchi, 2012). It borrows its terminology from choosing one slot machine, a "one-armed bandit", among others: a slot machine (or output) is chosen before the reward is revealed and it remains unknown what reward would have been obtained if another slot machine (or output) had been chosen. Bandit learning can be seen as a special problem of reinforcement learning (Sutton and Barto, 1998) and has been investigated for structured prediction problems previously by Sokolov et al. (2016) and Kreutzer et al. (2017). In both works, models were updated **on-policy** (see Figure 7.1), where the term **policy** is used as a synonym for model in reinforcement learning literature and which we adopt in the following.

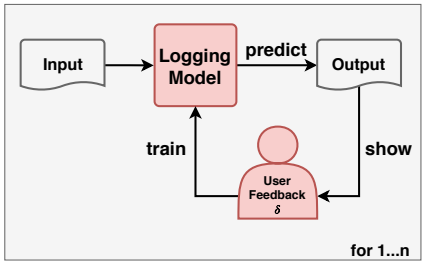


Figure 7.1.: Graphical view of on-policy learning. During on-policy learning, the system presents an output to a user and receives a reward with which the system's parameters are immediately updated.

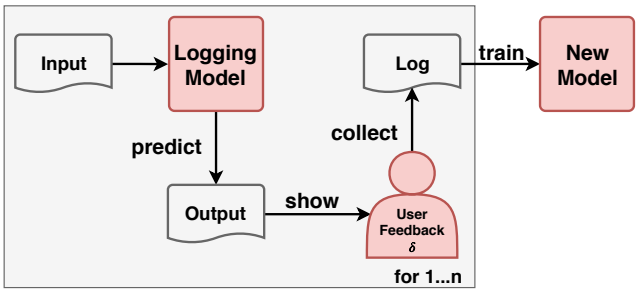


Figure 7.2.: Graphical view of off-policy learning. During off-policy learning, the system presents an output to a user and receives a reward. Together with the input, the information is saved to a log. Later on the log is used to update a system, which can but doesn't have to be the system that produced the log.

In contrast, our work assumes that a sufficient amount of data is collected in a log first and the log is then used in an **off-policy** setting to update the policy (see Figure 7.2). This scenario has for example been explored by (Bottou et al., 2013) on the task of response prediction for displaying advertisement online. It has also been studied in the context of contextual bandits (Langford et al. (2008); Strehl et al. (2010); Dudik et al. (2011); Li et al. (2015), *inter alia*), reinforcement learning (Sutton and Barto (1998); Precup et al. (2000); Jiang and Li (2016); Thomas and Brunskill (2016), *inter alia*), and structured prediction (Swaminathan and Joachims (2015a,b); Joachims et al. (2018), *inter alia*).

We choose to update models offline, i.e. not while they are deployed, for several reasons. First, updating a deployed system could cause the system to deteriorate without notice which would result in showing inferior outputs to the user and cause a monetary loss. This danger is heightened if a user recognises that they can influence the application and intentionally gives malicious feedback. Second, in an offline setting it is possible to test various algorithms and hyperparameter settings. Third, the resulting new policy can be validated against separate test sets to ensure satisfactory performance before deployment. The safety that offline learning provides, is essential for real-world applications where deteriorated models bear a significant risk (Bottou et al., 2013; Swaminathan and Joachims, 2015a; Thomas et al., 2015) and it is one possible approach of safe reinforcement learning (García and Fernández, 2015; Abbeel et al., 2009).

Algorithm 6 Pseudo-code for collecting stochastic logs.

Logging policy μ
$\mathcal{D} = \{\}$
for $t = 0, \ldots, T$ **do**
    Observe x_t
    Sample $y_t \sim \mu(y|x_t)$
    Obtain feedback δ_t
    $\mathcal{D} \cup \{x_t, y_t, \delta_t, \mu(y_t|x_t)\}$

While offline learning is the safer choice, it introduces a complication. The outputs recorded in the log are chosen by the logging policy and are thus biased towards this policy. Using the log to learn a new policy presents a problem because the new policy would have chosen different outputs if it had been in control during logging. For this reason, off-policy learning is also referred to as **counterfactual learning**. The data bias in the log can be corrected using **importance sampling**. For this, we additionally record the logging policy's probability for the chosen output, i.e. the **propensity score**. Using the inverse propensity score (Rosenbaum and Rubin, 1983), it is possible to use the log to obtain an unbiased estimate of the average **reward** a new policy would procure. The resulting Inverse Propensity Scoring (IPS) estimator can be used as the objective in a gradient ascent setup to improve a policy, i.e. for **policy learning**. Alternatively, the estimator can be used for **policy evaluation**, where it estimates the expected average reward of another policy.

The IPS estimator suffers from high variance which can lead to problems in praxis. Two extensions have been suggested previously to deal with this issue. One approach reweights the importance sampling ratio (Precup et al. (2000); Jia and Liang (2016); Thomas and Brunskill (2016)), leading to the Reweighted Inverse Propensity Scoring (IPS+R) estimator. Swaminathan and Joachims (2015b) showed that, from the perspective of Monte Carlo simulation, this can be seen as using a multiplicative control variate (Kong (1992)). But both the IPS and IPS+R estimator can suffer from degenerate behaviour for real-valued positive rewards. We give an analysis and a corresponding proof regarding this behaviour in this chapter.

Another approach to reduce the variance of the IPS estimator is the Doubly Robust (DR) estimator (Dudik et al. (2011); Jia and Liang (2016); Thomas and Brunskill (2016)). For this estimator, IPS is combined with a Direct Method (DM) reward predictor which can predict rewards for any input but suffers from high bias. This can be seen as extending the IPS estimator with an additive control variate (Ross (2013), Chapter 9). In previous work, the linear interpolation parameter between IPS and DM has simply been set to 1. We extend this by empirically calculating the optimal value for this scalar (Ross (2013), Chapter 9), leading to the $\hat{c}$ Doubly Robust ($\hat{c}$ DR) estimator.

In order for any of these counterfactual estimators to work well, the output space needs to be explored sufficiently by the logging policy (Langford et al., 2008; Strehl et al., 2010). This condition can easily be satisfied if a policy samples an output from the underlying distribution, leading to stochastically created logs (see Algorithm 6). However, in many scenarios such sampling can result in inferior outputs, which in turn lead to user dissatisfaction and monetary loss. This danger is particularly high for sequence-to-sequence tasks in NLP be-

Algorithm 7 Pseudo-code for collecting deterministic logs.

Logging policy μ
$\mathcal{D} = \{\}$
for $t = 0, \ldots, T$ **do**
    Observe x_t
    $y_t = \arg\max_{y \in \mathbf{Y}(x_t)} \mu(y|x_t)$
    Obtain feedback δ_t
    $\mathcal{D} \cup \{x_t, y_t, \delta_t\}$

cause all but a small number of outputs would be incorrect. For example, in machine translation, a commercial system can typically only present one translation to the user. To avoid the risk of showing a bad output to a user, sampling is avoided and only the most likely translation under the current policy is presented.

The strategy of showing only the most likely output leads to deterministic logging (see Algorithm 7). In this setup, the propensity score becomes 1 and as a consequence, the importance sampling of the inverse propensity score approach is disabled. Furthermore, for deterministic logging, explicit exploration is missing, which makes it unclear if the output space is sufficiently explored. To this end, we define deterministic estimators which are counterparts to the introduced stochastic estimators. Using these estimators, we design an experiment that aims to discern if deterministic logging is feasible for sequence-to-sequence learning in NLP.

In particular, our goal is the domain adaptation of a machine translation system, which uses the linear SMT framework Cdec (Dyer et al., 2010). We can show empirically on two separate domains and language pairs that the best deterministic estimator is up to par with the best stochastic estimator. With improvements of up to 2 BLEU, we can demonstrate that it is possible to use counterfactual learning for the sequence-to-sequence task of machine translation. In the next chapter, we also apply counterfactual learning to another sequence-to-sequence task, namely semantic parsing for question-answering.

Counterfactual learning has previously been typically applied to problems with small output spaces. Its success on a machine translation task, shows that the approach is also applicable to problems with large output spaces. Even though explicit exploration is missing for deterministic logging, we give a possible explanation why implicit exploration is sufficient for sequence-to-sequence tasks.

The main contributions of this chapter are as follows:

- Application of counterfactual policy evaluation and learning to a problem with a large output space, namely machine translation.
- Definition of deterministic estimators: Deterministic Propensity Matching (DPM), Reweighted Deterministic Propensity Matching (DPM+R), Doubly Controlled (DC) and $\hat{c}$ Doubly Controlled ($\hat{c}$ DC).
- Implementation of all estimators for the framework Cdec.
- Analysis and proofs of degeneracies in the estimators IPS, DPM, IPS+R and DPM+R.

- Empirical evidence that deterministic logging performs as well as stochastic logging in a machine translation domain adaptation task.
- Intuitive sketch of why implicit exploration is sufficient for deterministic logging on sequence-to-sequence tasks.

The work presented in this chapter has been previously published in (Lawrence et al., 2017b) and (Lawrence et al., 2017a). Artem Sokolov defined the DR and $\hat{c}$DR estimators (Section 7.1.4) for stochastic policies. Furthermore, he wrote the script to generate logs for the SMT framework Cdec and created the baseline systems as well as the oracle systems. He allowed me to re-use functions from the mentioned script to implement the counterfactual policy learning algorithms introduced in this chapter for Cdec. Additionally, Stefan Riezler and Pratik Gajane advised me on the best presentation and structure for the proofs presented in Section 7.2.

The structure of this chapter is as follows: Section 7.1 introduces various counterfactual estimators for stochastic and deterministic logging. Some of the estimators exhibit degenerate behaviour. This is analysed formally and intuitively in Section 7.2. Section 7.3 presents empirical results for both policy evaluation and policy learning and we discuss a possible explanation of why deterministic logging is as successful as stochastic logging.

7.1. Counterfactual Estimators

In the counterfactual setup for sequence-to-sequence learning, we assume an input $x \in \mathbf{X}$ that is mapped to an output $y \in \mathbf{Y}(x)$ by a parametrised model, in reinforcement learning terms also called policy. Furthermore, there exists a feedback function $\delta(x, y)$ that assigns a reward in the range of $[0, 1]$ to (x, y) pairs. A logging policy μ chooses an output y given the input x. Obtaining a reward $\delta(x, y)$, a log of triplets can be collected: $\mathcal{D} = \{(x_t, y_t, \delta_t)\}_{t=1}^{n}$, where y_t is an output produced by the policy μ given an input x_t and δ_t is the corresponding feedback.

For stochastic logging (see Algorithm 6), we additionally record the propensity score $\mu(y|x)$, i.e. the probability of y being chosen given x by the logging policy μ. In the case of deterministic logging (see Algorithm 7), the policy μ always chooses the most likely output y and thus the propensity score is $\mu(y|x) = 1$. We are interested in formulating counterfactual estimators that can estimate the average reward for another parametric policy π_w based on the log $\mathcal{D}$. This estimate can be used either for the policy evaluation of π_w or for policy learning to improve π_w.

In this chapter, we introduce four estimators for stochastic policies and their counterpart estimators for deterministic policies. The stochastic estimators can be used for policy evaluation. Policy evaluation using deterministic estimators is not recommended because they cannot counter the bias present in the log. Using counterfactual estimators for policy evaluation is a promising technique to replace costly A/B tests. In A/B tests one typically diverts a random subset of users to a new system B, whereas the remainder of the users continue to use system A. Comparing the behaviour of users of system A to the behaviour of users of

system B, it is then possible to infer whether or not system B should replace system A permanently and for all users. In such a setup only a limited number of alternative systems can be tested and the tests themselves can be time intensive. Using counterfactual estimators to estimate the average reward of a policy can be a time efficient alternative and opens up the possibility of testing many more alternative systems concurrently.

For policy learning, deriving the various counterfactual estimators $\mathcal{V}$ leads to a variety of learning algorithms when applying a gradient ascent update rule[35],

$$\Delta w = \eta \nabla \mathcal{V}(\pi_w). \tag{7.1}$$

Algorithm 8 gives the pseudo-code for counterfactual off-policy learning.

Algorithm 8 Pseudo-code for batch off-policy learning from logs.

```
Input: collected log 𝒟 of size n, learning rate η, initial policy π_w
repeat
    for t = 0, ..., n do
        Receive input-output pair (x_t, y_t) from the log 𝒟
        Calculate π_w(y_t|x_t)
    end for
    Calculate 𝒱(π_w)
    w = w + η∇_w 𝒱(π_w)
until Stopping Criterion Reached
```

7.1.1. Inverse Propensity Scoring (IPS)

Outputs are chosen by the logging policy μ and thus the resulting log $\mathcal{D}$ is biased towards the choices of μ. This bias can be corrected using importance sampling. The Inverse Propensity Scoring (IPS) (Rosenbaum and Rubin, 1983) estimator employs importance sampling to obtain an unbiased estimate of the expected reward for a different parametric policy π_w:

$$\begin{aligned} \mathcal{V}_{IPS}(\pi_w) &= \frac{1}{n} \sum_{t=1}^{n} \delta_t \frac{\pi_w(y_t|x_t)}{\mu(y_t|x_t)} \\ &\approx \mathbb{E}_{p(x)} \mathbb{E}_{\mu(y|x)} [\delta(y) \frac{\pi_w(y|x)}{\mu(y|x)}] \\ &= \mathbb{E}_{p(x)} \mathbb{E}_{\pi_w(y|x)} [\delta(y)]. \end{aligned} \tag{7.2}$$

[35] Because we formulate our counterfactual estimators as *reward* estimators, we employ gradient ascent, rather than gradient descent. By negating the rewards, we would recover equivalent *risk* estimators with which gradient descent could be performed.

7.1.2. Deterministic Propensity Matching (DPM)

Choosing the most likely output under the logging policy μ, leads to deterministic logs with $\mu(y|x) = 1$. Modifying the IPS estimator accordingly, leads to the Deterministic Propensity Matching (DPM) estimator:

$$\mathcal{V}_{DPM}(\pi_w) = \frac{1}{n}\sum_{t=1}^{n} \delta_t \pi_w(y_t|x_t). \tag{7.3}$$

In this setup it is no longer possible to correct the data bias and we are restricted to use $\mathcal{V}_{DPM}$ for empirical reward maximisation, which is equivalent to empirical risk minimisation if we set $\mathcal{L}_{DPM}(\pi_w) = -\mathcal{V}_{DPM}(\pi_w)$.

Both IPS and DPM can suffer from high variance because both will overfit to the outputs present in the log. This can be combated by defining appropriate control variates which lower the variance (Johnson and Zhang (2013); Wang et al. (2013); Defazio et al. (2014); Greensmith et al. (2004); Schulman et al. (2015); Ranganath et al. (2014); *inter alia*). A lower variance leads to faster convergence which has also been linked to improved generalisation (Hardt et al., 2016). We introduce two different control variates in the following.

7.1.3. Multiplicative Control Variate: Reweighting

Assume the existence of a random variable X. The variance of X can be reduced by choosing a suitable, i.e. positively correlating, control variate Y with an expectation $\bar{Y} = \mathbb{E}[Y]$ that is known or easily computed. In the case of a multiplicative control variate, we can formulate

$$\mathbb{E}[X] \approx \mathbb{E}[\frac{X}{Y}]\bar{Y}. \tag{7.4}$$

Note that this method introduces a bias, as $\mathbb{E}[\frac{X}{Y}] \neq \frac{X}{\mathbb{E}[Y]}$ (Swaminathan and Joachims, 2015b). But in praxis, the benefits from the reduced variance typically outweigh the downside of the introduced bias. The variance of $r = \frac{X}{Y}$ can be expressed as (Kong, 1992):

$$\text{Var}(r) \approx \frac{1}{n}(r^2\text{Var}(Y) + \text{Var}(X) - 2r\,\text{Cov}(Y, X)). \tag{7.5}$$

Equation 7.5 indicates that $\mathbb{E}[\frac{X}{Y}]\bar{Y}$ will exhibit lower variance than $\mathbb{E}[X]$ if the covariance between Y and X is sufficiently high.

If $X = \frac{1}{n}\sum_{t=1}^{n} \delta_t \rho_w(y_t|x_t)$, then $\rho_w(y_t|x_t) = \frac{\pi_w(y_t|x_t)}{\mu(y_t|x_t)}$ recovers the IPS estimator. To reduce the variance of the IPS estimator, we choose $Y = \frac{1}{n}\sum_{t=1}^{n} \frac{\pi_w(y_t|x_t)}{\mu(y_t|x_t)}$ with expectation (Swaminathan and Joachims, 2015b)

$$\bar{Y} = \frac{1}{n}\sum_{t=1}^{n}\int_{x_t}\int_{y_t}\frac{\pi_w(y_t|x_t)}{\mu(y_t|x_t)}\mu(y_t|x_t)p(x_t)\mathrm{d}y_t\mathrm{d}x_t = \frac{1}{n}\sum_{t=1}^{n}\int_{x_t}p(x_t)\mathrm{d}x_t = 1, \tag{7.6}$$

where $p(x_t)$ indicates the probability of observing x_t.

With Y chosen, we arrive at the Reweighted Inverse Propensity Scoring (IPS+R) estimator:

$$\begin{aligned}\mathcal{V}_{IPS+R}(\pi_w) &= \frac{\frac{1}{n}\sum_{t=1}^{n}\delta_t\frac{\pi_w(y_t|x_t)}{\mu(y_t|x_t)}}{\frac{1}{n}\sum_{t=1}^{n}\frac{\pi_w(y_t|x_t)}{\mu(y_t|x_t)}} \\ &= \sum_{t=1}^{n}\delta_t\bar{\rho}_w(y_t|x_t).\end{aligned} \tag{7.7}$$

The estimator has been used previously (Precup et al. (2000), Jiang and Li (2016), Thomas and Brunskill (2016)), although the link to control variates was not made explicit until Swaminathan and Joachims (2015b). Unlike IPS, IPS+R is bound by the range of δ because the reweight variable defines a probability distribution over the log and consequently $\bar{\rho}_w(y_t|x_t) \in [0,1]$. This is a contrast to the IPS estimator where estimates can lie outside the range defined by δ. However, an estimate in the range of δ is crucial for policy evaluation.

Setting $\rho_w(y_t|x_t) = \pi_w(y_t|x_t)$, we can define the deterministic counterpart to IPS, namely Reweighted Deterministic Propensity Matching (DPM+R):

$$\begin{aligned}\mathcal{V}_{DPM+R}(\pi_w) &= \frac{\frac{1}{n}\sum_{t=1}^{n}\delta_t\pi_w(y_t|x_t)}{\frac{1}{n}\sum_{t=1}^{n}\pi_w(y_t|x_t)} \\ &= \sum_{t=1}^{n}\delta_t\bar{\rho}_w(y_t|x_t).\end{aligned} \tag{7.8}$$

7.1.4. Additive Control Variate: DM Predictor

Additive control variates are another option to reduce variance. Given a random variable X, we can use an appropriate second random variable Y with expectation $\bar{Y} = \mathbb{E}[Y]$ as an additive control variate to reduce the variance of X:

$$\mathbb{E}[X] = \mathbb{E}[X - Y] + \bar{Y}. \tag{7.9}$$

The variance of $(X - Y)$ can be written as (Hoffman (2015); Ross (2013), Chapter 9.2):

$$Var(X - Y) = Var(X) + Var(Y) - 2Cov(X, Y). \tag{7.10}$$

Using Equation 7.10, we can formulate under which condition $(X - Y)$ has a lower variance than X:

$$\begin{aligned} Var(X - Y) &< Var(X) \\ Var(X) + Var(Y) - 2Cov(X, Y) &< Var(X) \\ Var(Y) &< 2Cov(X, Y) \\ \frac{1}{2}Var(Y) &< Cov(X, Y). \end{aligned} \tag{7.11}$$

Equation 7.11 implies that the additive control variate will lower the variance if $\frac{1}{2}Var(Y) < Cov(X, Y)$, i.e. Y should have a low variance while exhibiting a high covariance with X. A solution to ensuring that the condition of Equation 7.11 is always fulfilled, is to multiply Y and $\bar{Y}$ with a scalar $\hat{c}$ (Hoffman (2015); Ross (2013), Chapter 9.2), leading to:

$$\mathbb{E}[X] = \mathbb{E}[X - \hat{c}Y] + \hat{c}\bar{Y}. \tag{7.12}$$

The variance of $(X - \hat{c}Y)$ is (Ross (2013), Chapter 9.2):

$$Var(X - \hat{c}Y) = Var(X) + \hat{c}^2 Var(Y) - 2\hat{c}Cov(X, Y). \tag{7.13}$$

Similar to before, we can use Equation 7.13 to formulate under which condition $(X - \hat{c}Y)$ has a lower variance than X:

$$\begin{aligned} Var(X - \hat{c}Y) &< Var(X) \\ Var(X) + \hat{c}^2 Var(Y) - 2\hat{c}Cov(X, Y) &< Var(X) \\ \hat{c}^2 Var(Y) &< 2\hat{c}Cov(X, Y) \\ \frac{1}{2}\hat{c}Var(Y) &< Cov(X, Y). \end{aligned} \tag{7.14}$$

We can ensure that the condition from Equation 7.14 is always met by setting $\hat{c}$ optimally (Hoffman (2015); Ross (2013), Chapter 9.2). Taking the derivative of Equation 7.13 with regards to $\hat{c}$ gives

$$\begin{aligned} \nabla_{\hat{c}} \big(\; & Var(X) + \hat{c}^2 Var(Y) - 2\hat{c}Cov(X, Y)\big) \\ &= 2\hat{c}Var(Y) - 2Cov(X, Y). \end{aligned} \tag{7.15}$$

Setting Equation 7.15 to 0 and solving for $\hat{c}$ leads to

$$\hat{c} = \frac{Cov(X,Y)}{Var(Y)}. \tag{7.16}$$

Substituting Equation 7.16 in Equation 7.14 gives

$$\begin{aligned} \frac{1}{2}\frac{Cov(X,Y)}{Var(Y)}Var(Y) &< Cov(X,Y) \\ \frac{1}{2} &< 1 \end{aligned} \tag{7.17}$$

Equation 7.17 is always true, thus using the optimal value for the scalar $\hat{c}$ will guarantee a variance reduction. The amount of variance reduction is directly related to the correlation. This can be seen by dividing the variance $Var(X - Y)$ by the original variance $Var(X)$ (Hoffman, 2015):

$$\begin{aligned} \frac{Var(X-Y)}{Var(X)} &= \left[Var(X) + \hat{c}^2 Var(Y) - 2cCov(X,Y)\right]/Var(X) \\ &= 1 + \frac{\hat{c}^2 Var(Y)}{Var(X)} - \frac{2cCov(X,Y)}{Var(X)} \\ &= 1 + \frac{Cov^2(X,Y)Var(Y)}{Var(X)Var^2(Y)} - \frac{2Cov^2(X,Y)}{Var(X)Var(Y)} \qquad \text{Eq. 7.16} \\ &= 1 - \frac{Cov^2(X,Y)}{Var(X)Var(Y)} \\ &= 1 - corr^2(X,Y), \text{where } corr(X,Y) = \frac{Cov(X,Y)}{\sqrt{Var(X)}\sqrt{Var(Y)}}. \end{aligned} \tag{7.18}$$

Equation 7.18 shows that the variance reduction effect increases with a higher correlation between X and Y.

Let $X = \delta_t$ and $Y = \hat{\delta}_t$, where $\hat{\delta}_t$ is a Direct Method (DM) predictor trained using the log $\mathcal{D}$. Given pairs (x, y) as input, a model is fitted to predict δ as closely as possible. Once trained, the model can also deliver predictions for pairs (x, y) not present in the log $\mathcal{D}$. Thus, the expectation of Y can be approximated empirically as:

$$\bar{Y} = \sum_{y \in \mathbf{Y}(x_t)} \hat{\delta}(x_t, y)\ \pi_w(y|x_t). \tag{7.19}$$

The advantage of being able to predict a reward for all pairs (x, y) is off-set by a high bias from which DM predictors typically suffer. Thus, Dudik et al. (2011) combine the IPS estimator with a DM predictor to trade-off the high variance of IPS against the high bias of a DM predictor, so that together a more robust estimate can be reached. From the perspective of control variates, combining IPS and a DM predictor in an additive control variate setup reduces variance if there is a high enough positive correlation between X and Y and the variance of Y is correspondingly low enough (see the condition in Equation 7.11). With

$\nabla \mathcal{V}_{IPS/DPM} = \frac{1}{n} \sum_{t=1}^{n} \delta_t \rho_w(y_t\|x_t) \nabla \log \pi_w(y_t\|x_t).$
$\nabla \mathcal{V}_{IPS+R/DPM+R} = \frac{1}{n} \sum_{t=1}^{n} [\delta_t \bar{\rho}_w(y_t\|x_t)(\nabla \log \pi_w(y_t\|x_t) - \sum_{u=1}^{n} \bar{\rho}_w(y_u\|x_u) \nabla \log \pi_w(y_u\|x_u))].$
$\nabla \mathcal{V}_{\hat{c}\,\text{DC}/\hat{c}\,\text{DR}} = \frac{1}{n} \sum_{t=1}^{n} [(\delta_t - \hat{c}\hat{\delta}) \bar{\rho}_w(y_t\|x_t)(\nabla \log \pi_w(y_t\|x_t) - \sum_{u=1}^{n} \bar{\rho}_w(y_u\|x_u) \nabla \log \pi_w(y_u\|x_u)) + \hat{c} \sum_{y \in \mathbf{Y}(x_t)} \hat{\delta}(x_t, y) \pi_w(y\|x_t) \nabla \log \pi_w(y\|x_t)].$

Table 7.1.: Gradients of various counterfactual estimators.

$\rho_w(y_t|x_t) = \frac{\pi_w(y|x)}{\mu(y|x)}$ we define the Doubly Robust (DR) estimator for stochastic logging and with $\rho_w(y_t|x_t) = \pi_w(y|x)$ we define the Doubly Controlled (DC) estimator for deterministic logging:

$$\mathcal{V}_{DR/DC}(\pi_w) = \frac{1}{n} \sum_{t=1}^{n} \Big[(\delta_t - \hat{\delta}_t)\ \bar{\rho}_w(y_t|x_t) + \sum_{y \in \mathbf{Y}(x_t)} \hat{\delta}(x_t, y)\ \rho_w(y|x_t) \Big]. \tag{7.20}$$

In this estimator, $\hat{c}$ equals 1 and the condition $Var(X - Y) < Var(X)$ from Equation 7.11 may or may not be fulfilled. We modify these estimators by setting $\hat{c} = \frac{Cov(X,Y)}{Var(Y)}$. This ensures that $Var(X - \hat{c}Y) < Var(X)$ from Equation 7.14 is fulfilled if $Var(Y)$ and $Cov(X, Y)$ are known. However, in praxis $Var(Y)$ and $Cov(X, Y)$ have to be estimated empirically, which introduces a bias. Again for stochastic logs, we set $\rho_w(y_t|x_t) = \frac{\pi_w(y|x)}{\mu(y|x)}$ and for deterministic logs, we set $\rho_w(y_t|x_t) = \pi_w(y|x)$. This leads to the $\hat{c}$ DR and $\hat{c}$ DC estimators for stochastic and deterministic logging, respectively:

$$\mathcal{V}_{\hat{c}\,\text{DC}/\hat{c}\,\text{DR}}(\pi_w) = \frac{1}{n} \sum_{t=1}^{n} \Big[(\delta_t - \hat{c}\hat{\delta}_t)\ \bar{\rho}_w(y_t|x_t) + \hat{c} \sum_{y \in \mathbf{Y}(x_t)} \hat{\delta}(x_t, y)\ \rho_w(y|x_t) \Big]. \tag{7.21}$$

Gradients for the various estimators can be found in Table 7.1 and were be obtained using the score function gradient estimator (Fu, 2006) (also see Section 2.2). For a more detailed derivation see Appendix B.

7.2. Degeneracy Analysis

7.2.1. IPS & DPM

Next to suffering from high variance, the IPS and DPM estimators have an inherent issue that can lead to degenerate behaviour during policy learning. The estimators can be maximised by simply setting all logged outputs to probability 1, i.e. $\pi_w(y_t|x_t) = 1, \forall(y_t, x_t, \delta_t) \in \mathcal{D}$. Obviously, this is undesired as the probability for low reward outputs should not be raised.

Theorem 1. $\max_\pi \mathcal{V}_{IPS}$ *and* $\max_\pi \mathcal{V}_{DPM}$ *if* $\forall(y_t, x_t, \delta_t) \in \mathcal{D} : \pi(y_t|x_t) = 1 \wedge \delta_t > 0$.

Proof. For abbreviation, we set $\pi_t = \pi_w(y_t|x_t)$ and $\mu_t = \mu(y_t|x_t)$. We need to show that the value of $\mathcal{V}_{IPS}$ with $\pi_t = 1$ for $\forall(y_t, x_t, \delta_t) \in \mathcal{D} = \{(x_t, y_t, \delta_t)\}_{t=1}^n$ is greater than the value of $\mathcal{V}_{IPS}$ with $\exists(x_t, y_t, \delta_t)$ with $\pi_t \in [0, 1)$. W.l.og. assume that (x_n, y_n, δ_n) is the tuple with $\pi_t \in [0, 1)$.

$$\begin{aligned} \sum_{t=1}^{n} \frac{\delta_t}{\mu_t} &> \sum_{t=1}^{n-1} \frac{\delta_t}{\mu_t} + \frac{\delta_n \pi_n}{\mu_n}, \\ \frac{\delta_n}{\mu_n} &> \frac{\delta_n \pi_n}{\mu_n}, \\ 1 &> \pi_n, \end{aligned} \tag{7.22}$$

which is true for $\pi_n \in [0, 1)$. Because DPM is a special case of IPS with $\mu_t = 1$ for $\forall(y_t, x_t) \in \mathcal{D}$, the proof also holds for DPM. □

As a consequence, IPS and DPM cannot discriminate between outputs and are only concerned with overall probabilities. If a small constant c is added to every probability $\pi_w(y|x)$ for the entries in the log $\mathcal{D}$,[36] the overall value of the estimator increases. The resulting new policy is considered better than the original policy even though the new policy cannot distinguish any better between outputs that have good rewards and outputs that have bad rewards.

In Figure 7.3, a toy example visualises the problem. Assume there are two samples in the log. The first sample has a propensity score of $\mu_1 = 0.1$ and a reward $\delta_1 = 0.9$ and the second sample has $\mu_2 = 0.001$ and $\delta_2 = 0.3$. For simplicity, lets assume that new policies can only assign the following probabilities for the two samples: $\pi_1 = \pi_2 = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$. This gives rise to 100 possible combinations. Their IPS estimates can be seen in Figure 7.3. The differing values on the diagonal illustrate the problem of a policy gaining a higher estimate by adding the constant value $c = 0.1$ to all probabilities in the log. Additionally, because of the division by the propensity score, none of the estimates are within the range of the reward defined by the log (here $[0.3, 0.9]$). The policy with the best estimate is obtained when $\pi_1 = \pi_2 = 1.0$. This is not a desired solution because the second sample has a low reward of 0.3 and this sample's probability should not be increased.

[36] We assume that the value c is sufficiently small so that no probabilities result in a value higher than 1.

	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	---	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000	0.9000
0.1	0.3000	0.3059	0.3118	0.3175	0.3231	0.3286	0.3340	0.3393	0.3444	0.3495	0.3545
0.2	0.3000	0.3030	0.3059	0.3089	0.3118	0.3146	0.3175	0.3203	0.3231	0.3258	0.3286
0.3	0.3000	0.3020	0.3040	0.3059	0.3079	0.3098	0.3118	0.3137	0.3156	0.3175	0.3194
0.4	0.3000	0.3015	0.3030	0.3045	0.3059	0.3074	0.3089	0.3103	0.3118	0.3132	0.3146
0.5	0.3000	0.3012	0.3024	0.3036	0.3048	0.3059	0.3071	0.3083	0.3094	0.3106	0.3118
0.6	0.3000	0.3010	0.3020	0.3030	0.3040	0.3050	0.3059	0.3069	0.3079	0.3089	0.3098
0.7	0.3000	0.3009	0.3017	0.3026	0.3034	0.3043	0.3051	0.3059	0.3068	0.3076	0.3085
0.8	0.3000	0.3007	0.3015	0.3022	0.3030	0.3037	0.3045	0.3052	0.3059	0.3067	0.3074
0.9	0.3000	0.3007	0.3013	0.3020	0.3027	0.3033	0.3040	0.3046	0.3053	0.3059	0.3066
1	0.3000	0.3006	0.3012	0.3018	0.3024	0.3030	0.3036	0.3042	0.3048	0.3054	0.3059

Figure 7.4.: IPS+R estimates for a small toy log containing two samples with $\mu_1 = 0.1, \delta_1 = 0.9$ and $\mu_2 = 0.001, \delta_2 = 0.3$. Assume that the new policy can only assign the probabilities $\pi_1 = \pi_2 = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$. Presented are all possible 100 combinations and the corresponding IPS+R estimate. Red backgrounds mark the highest value, grey backgrounds mark the diagonal.

	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0	0.45	0.9	1.35	1.8	2.25	2.7	3.15	3.6	4.05	4.5
0.1	15	15.45	15.9	16.35	16.8	17.25	17.7	18.15	18.6	19.05	19.5
0.2	30	30.45	30.9	31.35	31.8	32.25	32.7	33.15	33.6	34.05	34.5
0.3	45	45.45	45.9	46.35	46.8	47.25	47.7	48.15	48.6	49.05	49.5
0.4	60	60.45	60.9	61.35	61.8	62.25	62.7	63.15	63.6	64.05	64.5
0.5	75	75.45	75.9	76.35	76.8	77.25	77.7	78.15	78.6	79.05	79.5
0.6	90	90.45	90.9	91.35	91.8	92.25	92.7	93.15	93.6	94.05	94.5
0.7	105	105.45	105.9	106.35	106.8	107.25	107.7	108.15	108.6	109.05	109.5
0.8	120	120.45	120.9	121.35	121.8	122.25	122.7	123.15	123.6	124.05	124.5
0.9	135	135.45	135.9	136.35	136.8	137.25	137.7	138.15	138.6	139.05	139.5
1	150	150.45	150.9	151.35	151.8	152.25	152.7	153.15	153.6	154.05	154.5

Figure 7.3.: IPS estimates for a small toy log containing two samples with $\mu_1 = 0.1, \delta_1 = 0.9$ and $\mu_2 = 0.001, \delta_2 = 0.3$. Assume that the new policy can only assign the probabilities $\pi_1 = \pi_2 = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$. Presented are all possible 100 combinations and the corresponding IPS estimate. Red background marks the highest value, grey backgrounds mark the diagonal.

One solution to the problem is to choose an appropriate learning rate and perform early stopping on a held out data set. The learning process is stopped before the undesired state can be reached. A more sophisticated solution is to use the IPS+R and DPM+R estimators, for IPS and DPM, respectively. The reweighting defines a probability distribution over the log $\mathcal{D}$. Now, increasing the probability of a low reward output takes away probability mass from the higher reward output. This decreases the value of the estimator, and will thus be avoided in learning. Increasing all probabilities by a small constant c, leads to the same estimates under IPS+R and DPM+R. This is visualised in Figure 7.4 using the same toy example introduced previously, but calculating the IPS+R estimates. The diagonal illustrates that the estimate remains constant as all probabilities are increased by $c = 0.1$, i.e. it is not possible any longer to obtain an higher estimate by simply increasing all probabilities. At

the same time, all estimates are now bound by the range of the reward as defined by the log.

A closely related issue for IPS has been noted by Swaminathan and Joachims (2015b). They demonstrate that adding a small constant c to all rewards δ in the log does not lead to the same increase by c in the IPS estimate. In other words, the IPS estimator is not equivariant with regards to the geometric translation of rewards. This issue is also fixed by using reweighting as a multiplicative control variate.

7.2.2. Reweighted Estimators

While IPS+R and DPM+R solve the degeneracy of IPS and DPM, they suffer from a degeneracy themselves. To show this, we define the set $\mathcal{D}^{max}$ that contains all tuples that receive the highest reward δ_{max} observed in the log, and we assume $\delta_{max} > 0$, leading to a cardinality of $\mathcal{D}^{max}$ of at least one.

Definition 1. *Let* $\mathcal{D}^{max} = \max_\delta \mathcal{D}$, *then* $\mathcal{D} = \mathcal{D}^{max} \cup \mathcal{D} \backslash \mathcal{D}^{max}$.

We will show that the estimators can be maximised by simply setting the probability of at least one tuple in $\mathcal{D}^{max}$ to a value higher than 0, while leaving all other tuples in $\mathcal{D}^{max}$ at their original probabilities in range $[0, 1]$, and setting the probability of tuples in the set $\mathcal{D} \backslash \mathcal{D}^{max}$ to 0. Clearly, this is undesired as outputs with a reward close to δ_{max} should not receive a probability of 0. Furthermore, this learning goal is easy to achieve since a degenerate estimator only needs to be concerned about lowering the probability of tuples in $\mathcal{D} \backslash \mathcal{D}^{max}$ as long as there is one tuple of $\mathcal{D}^{max}$ with a probability above 0. This is visualised in Figure 7.4: The maximum value is achieved if the lower reward sample has a probability of 0 and the higher reward sample has any probability higher than 0. We want to prove the following theorem:

Theorem 2. $\max_\pi \mathcal{V}_{IPS+R} \wedge \max_\pi \mathcal{V}_{DPM+R}$ *if* $\exists (x_t, y_t, \delta_{max}) \in \mathcal{D}^{max} : \pi_t \in (0, 1]$ $\wedge \forall (y_t, x_t, \delta_t) \in \mathcal{D} \backslash \mathcal{D}^{max} : \pi_t = 0$.

We introduce a definition of data indices belonging to the sets $\mathcal{D}^{max}$ and its complement in $\mathcal{D}$:

Definition 2. *Let*

$$\mathcal{V}_{IPS+R}(\pi_w) = \frac{\sum_{t=1}^{n} \delta_t \frac{\pi_t}{\mu_t}}{\sum_{t=1}^{n} \frac{\pi_t}{\mu_t}} = \frac{\delta_{max} \sum_{t=1}^{s-1} \frac{\pi_t}{\mu_t} + \sum_{t=s}^{n} \delta_t \frac{\pi_t}{\mu_t}}{\sum_{t=1}^{n} \frac{\pi_t}{\mu_t}},$$

where w.l.o.g. indices $(1 \dots (s-1))$ *refer to tuples in* $\mathcal{D}^{max}$ *and indices* $(s \dots n)$ *refer to indices in* $\mathcal{D} \backslash \mathcal{D}^{max}$. *Thus,* $\mathcal{D}^{max} = \{(x_t, y_t, \delta_{max})\}_{t=1}^{s-1}$ *and* $\mathcal{D} \backslash \mathcal{D}^{max} = \{(x_t, y_t, \delta_t)\}_{t=s}^{n}$.

Proof. We need to show that the value of $\mathcal{V}_{IPS+R}$ where $\pi_t = 0$ for $\forall (y_t, x_t, \delta_t) \in \mathcal{D}^{max}$ is lower than the value of $\mathcal{V}_{IPS+R}$ where $\exists (x_t, y_t, \delta_{max}) \in \mathcal{D}^{max}$ with $\pi_t \in (0, 1]$. Then

$$\frac{\sum_{t=s}^{n} \delta_t \frac{\pi_t}{\mu_t}}{\sum_{t=s}^{n} \frac{\pi_t}{\mu_t}} < \frac{\delta_{max} \sum_{t=1}^{s-1} \frac{\pi_t}{\mu_t} + \sum_{t=s}^{n} \delta_t \frac{\pi_t}{\mu_t}}{\sum_{t=1}^{n} \frac{\pi_t}{\mu_t}}, \tag{7.23}$$

$$0 < \frac{\delta_{max} \sum_{t=1}^{s-1} \frac{\pi_t}{\mu_t}}{\sum_{t=1}^{n} \frac{\pi_t}{\mu_t}},$$

where the last line is true for $\delta_{max} > 0$ as long as $\exists(x_t, y_t, \delta_{max}) \in \mathcal{D}^{max}$ with $\pi_t > 0$ as $\mu_t \in (0, 1]$ by definition.

Furthermore, we need to show that the value of $\mathcal{V}_{IPS+R}$ where $\exists(y_t, x_t, \delta_t) \in \mathcal{D} \backslash \mathcal{D}^{max}$ with $\pi_t \in (0, 1]$ is lower than the value of $\mathcal{V}_{IPS+R}$ with $\pi_t = 0$ for $\forall(y_t, x_t, \delta_t) \in \mathcal{D} \backslash \mathcal{D}^{max}$.

From the above, it is clear that $\exists(x_t, y_t, \delta_{max}) \in \mathcal{D}^{max}$ with $\pi_t \in (0, 1]$, thus $\frac{\delta_{max} \sum_{t=1}^{s-1} \frac{\pi_t}{\mu_t}}{\sum_{t=1}^{s-1} \frac{\pi_t}{\mu_t}}$ is defined. W.l.o.g. assume that $(y_s, x_s, \delta_s) \in \mathcal{D} \backslash \mathcal{D}^{max}$ is the tuple with $\pi_s \in (0, 1]$. Then

$$\frac{\delta_{max} \sum_{t=1}^{s-1} \frac{\pi_t}{\mu_t} + \delta_s \frac{\pi_s}{\mu_s} + \sum_{t=s+1}^{n} \delta_t \frac{0}{\mu_t}}{\sum_{t=1}^{s} \frac{\pi_t}{\mu_t} + \sum_{t=s+1}^{n} \frac{0}{\mu_t}} < \frac{\delta_{max} \sum_{t=1}^{s-1} \frac{\pi_t}{\mu_t} + \sum_{t=s}^{n} \delta_t \frac{0}{\mu_t}}{\sum_{t=1}^{s-1} \frac{\pi_t}{\mu_t} + \sum_{t=s}^{n} \frac{0}{\mu_t}} = \delta_{max}, \tag{7.24}$$

$$\delta_{max} \sum_{t=1}^{s-1} \frac{\pi_t}{\mu_t} + \delta_s \frac{\pi_s}{\mu_s} < \delta_{max} \sum_{t=1}^{s} \frac{\pi_t}{\mu_t},$$

$$\delta_{max} \sum_{t=1}^{s-1} \frac{\pi_t}{\mu_t} + \delta_s \frac{\pi_s}{\mu_s} < \delta_{max} \sum_{t=1}^{s-1} \frac{\pi_t}{\mu_t} + \delta_{max} \frac{\pi_s}{\mu_s},$$

$$\delta_s \frac{\pi_s}{\mu_s} < \delta_{max} \frac{\pi_s}{\mu_s},$$

$$\delta_s < \delta_{max}.$$

The last line of Equation 7.24 is true as $\delta_s < \delta_{max}$ by Definition 1 . As DPM+R is a special case of IPS+R with $\mu_t = 1$ for $\forall(y_t, x_t) \in \mathcal{D} = \{(x_t, y_t, \delta_t)\}_{t=1}^{n}$, the proof also holds for DPM+R. □

While employing stochastic gradient ascent, IPS+R and DPM+R can be prevented from reaching their degenerate state by performing early stopping on a development set. However, one cannot control what happens to the probability mass that is freed when lowering the probability of a logged output. The freed probability mass could be allocated to outputs that receive a lower reward than the logged output which would create a system that is worse than the logging system.

The estimators DR, DC, $\hat{c}$DR and $\hat{c}$DC successfully solve this problem. The DM predictor takes the whole output space into account and thus assigns rewards to any output. The objective will now be increased if the probability of outputs with high estimated reward is increased, and decreased for outputs with low estimated reward. For this to happen, high reward outputs other than the ones with maximal reward will be considered, even if the outputs have not been seen in the log. This will shift probability mass to unseen data with high estimated reward, which is a desired property in learning.

7.3. Empirical Results

We test the counterfactual estimators for both policy evaluation and policy learning on a domain adaptation setup for machine translation. In machine translation the output space is defined by the vocabulary size of the target language $\mathbf{V}_y$ and the maximum possible sentence length s, leading to $\mathbf{V}_y^s$ possible combinations. For example, a target vocabulary of $90,000$ and a maximum sentence length of 200 leads to an output space of size $90,000^{200}$. Machine translation is thus an ideal problem to demonstrate that counterfactual policy evaluation and learning is possible for problems with a large output space.

For domain adaptation, an out-of-domain system is trained and used to translate in-domain data. We distinguish between deterministic and stochastic logging. In the case of deterministic logging, the policy chooses the most likely translation and a propensity score of 1 is recorded. For stochastic logging, the policy samples from the underlying probability distribution for the current input. With the help of the true references, we simulate corresponding feedback for the chosen translations using sentence-level BLEU (Nakov et al., 2012). In such a simulated setup, a direct comparison between stochastic and deterministic logging is possible and will allow us to draw a conclusion whether deterministic logging provides sufficient exploration to keep up with stochastic logging. This would be a great advantage for deployed machine translation services which want to always present the most likely translation of the current system to the user. However, for policy evaluation only stochastically logged data is suitable and thus we assume that for policy evaluation a small amount of data can be logged using a stochastic scheme.

Previously, counterfactual estimators have only been used in conjunction with linear models. Thus, we will also employ a linear model in these experiments. As the SMT framework we use Cdec (Dyer et al., 2010) which employs a log-linear model (see Equation 2.17 in Section 2.3),

$$\pi_w(y|x) = \frac{e^{\alpha w \phi(x,y)}}{\sum_{y \in \mathbf{Y}(x)} e^{\alpha w \phi(x,y)}}, \tag{7.25}$$

where ϕ is a vector of different features of the model and α is a hyperparameter to be set. For high α, the distribution becomes increasingly peaked. As $\alpha \to \infty$, the distribution approaches the deterministic case where $y = \arg\max_{y \in \mathbf{Y}(x)}$. The derivative of this model is

$$\nabla \log \pi_w(y|x) = \alpha\big(\phi(x,y) - \sum_{y \in \mathbf{Y}(x)} \phi(x,y)\pi_w(y|x)\big), \tag{7.26}$$

and it can easily be plugged into the gradients from Table 7.1. The weight vector w can be updated using the gradient ascent update rule $\Delta w = \eta \nabla \mathcal{V}(\pi_w)$.

We define two separate domain adaptation tasks which differ in language pair and target domain. The first task uses the German-to-English Europarl corpus (Koehn, 2005) with the goal to adapt a system trained on it to the domain of transcribed TED talks using the TED

	TED DE-EN	News FR-EN
train	122k	30k
development	3k	1k
test	3k	2k

Table 7.2.: Number of sentences for in-domain data splits of the train, development, and test data.

parallel corpus (Tiedemann, 2012). The second task uses the French-to-English Europarl data with the goal of domain adaptation to news articles with the News Commentary corpus (Koehn and Schroeder, 2007). We split off two parts from the TED corpus to be used as development and test data for the learning experiments. As development data for the News corpus we use the splits provided at the WMT shared task[37], namely `nc-devtest2007` as development data and `nc-test2007` as test data. An overview of the splits and their sizes can be seen in Table 7.2.

As a baseline, an out-of-domain system is built using Cdec (Dyer et al., 2010) with dense features (10 standard features and 2 for the language model). After tokenizing and lowercasing the training data, the data were word aligned using Cdec's fast_align. A 4-gram language model is built on the target languages for the out-of-domain data using KenLM (Heafield et al., 2013). For News, we additionally assume access to in-domain target language text and train another in-domain language model on that data, increasing the number of features to 14 for News. We peak the distribution of the log-linear model by setting $\alpha = 5$ (see Equation 7.25), which we found to work well on development data. By setting $\alpha > 1$, we can encourage to learn a model that is suited to producing one-best translations, which is what the model will be asked to produce when deployed. All models are evaluated using the corpus-level BLEU metric (Papineni et al., 2002).

7.3.1. Direct Method (DM) Predictor

For the estimators DR, $\hat{c}$ DR, DC and $\hat{c}$ DC, we need a DM predictor. We build a regression model using a random forest of decision trees. The model is trained using scikit-learn (Pedregosa et al., 2011) and uses 10 trees.

The input for the scikit-learn model is a vector of decoder features $\phi(x, y)$ from Cdec for the given source sentence x and logged translation y. The target value is the true reward δ recorded in the log. To measure the performance of the random forest predictor, we employ 5-fold cross validation on $\mathcal{D}$. Each 5-fold cross validation is performed 10 times due to random initialisations and the average of the underlying micro and macro average is reported. We present the micro average between estimated and true reward, which quantifies the expectation of how much a random sample will differ from the true target:

[37] `http://www.statmt.org/wmt07/`, 1st September 2018

$$\text{Micro} := \frac{1}{n} \sum |\delta(x_i, y_i) - \hat{\delta}(x_i, y_i)|. \tag{7.27}$$

Additionally, we report the macro difference which calculates the difference of the average estimated and the average true reward on the entire held-out fold:

$$\text{Macro} := |\frac{1}{n} \sum \delta(x_i, y_i) - \frac{1}{n} \sum \hat{\delta}(x_i, y_i)|. \tag{7.28}$$

The results for both deterministic and stochastic logging can be found in Table 7.3. The DM models that the various estimators employ, are trained on the entire training set.

	Deterministic		Stochastic	
	Micro	Macro	Micro	Macro
TED	15.02	0.66	14.18	0.65
News	10.86	0.22	10.69	0.22

Table 7.3.: 5-fold cross validation: Macro and micro difference for deterministic and stochastic logging for a random forest regression model as the DM predictor.

7.3.2. Policy Evaluation

Policy evaluation can only be performed using stochastically logged data. If we were to use deterministically logged data, we could not correct the bias of the logging policy and this is essential for policy evaluation. Thus, we assume that for a small percentage of users, the system outputs are chosen stochastically. Based on this data, we perform policy evaluation using the estimators IPS+R, DR and $\hat{c}$ DR. IPS is omitted because it does not return values in the permissible range of rewards. A log $\mathcal{D}_{eval}$ of 10,000 instances is created using the baseline out-of-domain translation system μ. The goal is to estimate the performance of an in-domain system π_w.

To obtain the in-domain system, the weights of the baseline system are tuned with Mert (Och, 2003) using supervised in-domain data. Due to the random component in Mert, the final weights are the average of three runs. To measure the true performance of the resulting in-domain system, it also translates the source sentences of the log $\mathcal{D}_{eval}$ and returns the most likely translation. This set of translations can be scored against the references to obtain the true BLEU score. Using $\mathcal{D}_{eval}$, we estimate the expected reward with estimators IPS+R, DR and $\hat{c}$ DR. Comparing the estimated BLEU score to the true BLEU score, allows us to determine the best estimator for policy evaluation. The differences between estimated and true BLEU score are presented in Table 7.4. Due to the randomness in the stochastic sampling scheme, 5 separates logs were created and we report average and standard deviation.

For the News corpus, IPS+R underestimates the system by nearly 8 BLEU and DR overestimates the system by nearly 7 BLEU. Setting $\hat{c}$ optimally, balances out the estimate which

		IPS+R	DR	$\hat{c}$ DR
TED	average estimate	+4.00	+7.98	+6.07
	standard deviation	0.64	3.83	2.06
News	average estimate	-7.78	+6.63	+0.95
	standard deviation	0.97	4.13	2.33

Table 7.4.: Policy Evaluation: Macro averaged difference between estimated and ground truth BLEU on 10k stochastically logged data. Results are averaged over five independent runs.

now overestimates the system by only about 1 BLEU. For the TED corpus, the IPS+R estimator already overestimates the system by 4 BLEU. This overestimation is exacerbated further by the DR estimator, which stems from the fact that the random forest DM predictor tends to overestimate. Again $\hat{c}$ DR can improve upon DR but it does not do so in a sufficient way.

7.3.3. Policy Learning

For policy learning, we use the entire log $\mathcal{D}$. The initial policy π_w that we aim to improve is the out-of-domain baseline and logging policy μ. For deterministic logging, we test DPM+R, DC and $\hat{c}$ DC and for stochastic logging we use IPS+R, DR and $\hat{c}$ DR. All estimators are used to improve the policy π_w by hypergraph re-decoding. Due to the large output space, we normalise the 1,000 most likely outputs and search this k-list for the translation recorded in the log. Hyperparameters are minibatch sizes of either $10k$ or $30k$ samples and the learning rate η is chosen on the development set from the set $\{1e-4, 1e-5, 1e-6\}$ or, alternatively, Adadelta (Zeiler, 2012), which sets the learning rates on a per-feature basis.

We also considered a simpler, alternative option to learning from the logged data by using a Bandit-to-Supervised (B2S) approach. In this setup, either 2,000 random or the 2,000 highest reward instances are chosen from the log.[38] This data is then treated as a supervised data set and the parameters of the logging policy are adjusted using Mert. The weights of all Mert experiments are averaged over three runs due to its randomised component. Finally, the in-domain model trained on supervised in-domain data from the policy evaluation can be seen as an oracle model in this context. It conveys the best possible improvement over the baseline model that is attainable. Statistical significance between the out-of-domain system and all other systems is measured on the test set using an approximate randomisation test (Noreen, 1989; Riezler and Maxwell, 2005). Results can be found in Table 7.5.

For the B2S baseline, we observe some small gains in some cases and no improvements at all in others. This is the case for both deterministic and stochastic logging, although the B2S approach seems to work slightly better for the stochastic log. For the deterministic counterfactual estimators, DPM+R can improve significantly upon the baseline on both corpora. DC can improve upon this result further, doubling the improvements compared to DPM+R in the case of the TED corpus. Setting $\hat{c}$ optimally in the $\hat{c}$ DC estimator, leads to the largest

[38] Note that it is not possible to use a data set as large as the entire log $\mathcal{D}$ in conjunction with Mert, which is why we use a subset of 2,000 instances from $\mathcal{D}$.

	deterministic	TED		News	
		dev	test	dev	test
1	out-of-domain	22.39	22.76	24.64	25.27
2	B2S top 2k	+ 0.00	+ 0.00	+ 0.00	+ 0.50
3	B2S random 2k	+ 0.00	+ 0.00	+ 0.00	+ 0.50
5	DPM+R	+ 0.59	+ 0.67*	+ 0.62	+ 0.94*
6	DC	+ 1.50	+ 1.41*	+ 0.99	+ 1.05*
7	$\hat{c}$ DC	**+ 1.89**	**+ 2.02***	**+ 1.02**	**+ 1.13***
8	in-domain	25.43	25.58*	27.62	28.08*
	stochastic	TED		News	
		dev	test	dev	test
1	out-of-domain	22.39	22.76	24.64	25.27
2	B2S top 2k	+ 0.28	+ 0.20*	+ 0.31	+ 0.72*
3	B2S random 2k	+ 0.02	+ 0.03	+ 0.00	+ 0.49*
5	IPS+R	+ 0.57	+ 0.58*	**+ 0.71**	+ 0.81
6	DR	+ 1.92	+ 2.04*	+ 1.00	**+ 1.18***
7	$\hat{c}$ DR	**+ 1.95**	**+ 2.09***	**+ 0.71**	+ 0.95*
8	in-domain	25.43	25.58*	27.62	28.08*

Table 7.5.: Policy Learning: BLEU increases over the out-of-domain baseline on dev and test sets for various counterfactual objectives. Out-of-domain is the baseline system and in-domain is the oracle system tuned on in-domain data with references. Best results are indicated in **bold face**. Results marked with * are significant at $p \leq 0.002$ to the out-of-domain system.

improvement of about 2 BLEU on the TED corpus. The difference between DC and $\hat{c}$ DC for TED is significant at $p = 0.0017$.

The gain from DC over $\hat{c}$ DC is less strong on the News corpus. This can be explained by looking at the values of $\hat{c}$. In the case of the TED corpus, $\hat{c}$ reaches an average value of 1.35 whereas on the News corpus it has a maximum value of 1.14, which is very close to setting $\hat{c} = 1.0$ and would be equivalent to using DC. It is thus not surprising that $\hat{c}$ DC cannot significantly outperform DC. The in-domain, oracle systems can increase about 3 BLEU point over the out-of-domain system for both corpora. With an increase of about 2 and 1 BLEU for TED and News, respectively, the best counterfactual estimator, $\hat{c}$ DC, can close this gap impressively despite the considerable difference of available information compared to the supervised data available to the oracle system.

For stochastic logging, we observe very similar results. Significance tests between deterministic estimators and their stochastic counterparts yield only a significant difference between DC and DR on the TED corpus. However, the DR result does not significantly outperform the best deterministic estimator $\hat{c}$ DC. For all other deterministic-stochastic pairs, the p values lie above 0.1. With this, our experiments attest that deterministic logging is just as effective as stochastic logging for machine translation.

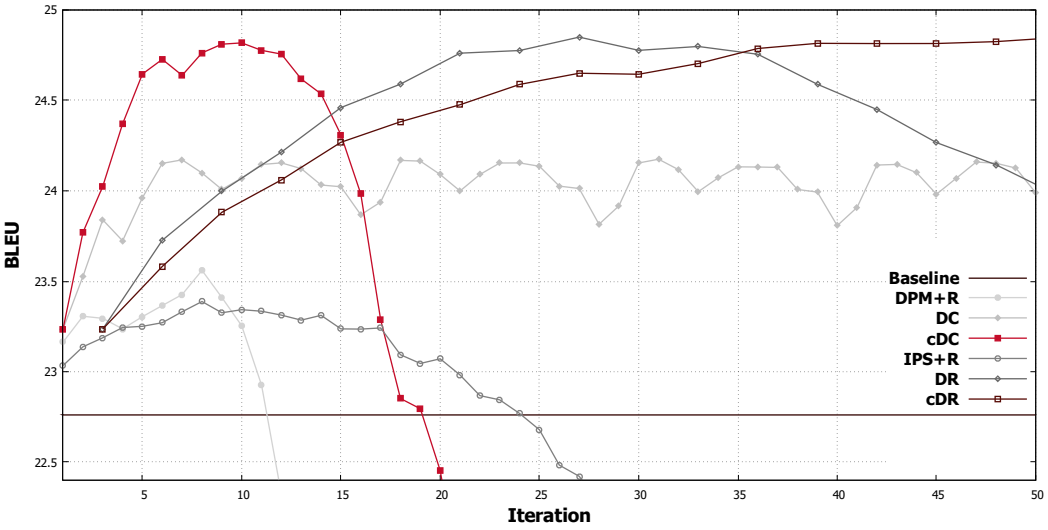


Figure 7.5.: BLEU performance on the TED corpus test set after every iteration for various counterfactual objectives.[39]

This result is also re-affirmed by the learning curves on the test set which can be found in Figures 7.5 and 7.6 for the TED and News corpora, respectively. Deterministically logged objectives are displayed using filled out symbols while their stochastic counterparts use the same symbol, but only its outline. Except for $\hat{c}$DC versus $\hat{c}$DR on the News corpus and DC versus DR on the TED corpus, the deterministic estimators can reach the same or higher peaks than their stochastic counterpart. Additionally, the deterministic estimators reach their peak before their stochastic counterpart does. Overall, the performance of the deterministic estimator is a great result for production systems where deterministic logging is paramount. In the following, we give an intuition why deterministic logging explores the output space sufficiently so that stochastic logging is not necessary.

7.3.4. Implicit Exploration

To be able to learn, reinforcement learning algorithms must explore the action space sufficiently. If an algorithm always chooses the most likely action (exploitation), then it can never discover actions that might lead to a higher reward (exploration). Consider a policy that selects adverts to be shown on a website. Simply displaying the most likely advert is not sufficient (Chapelle and Li, 2011). Without exploration, this policy would always display the same advert. The policy needs to sample to explore different adverts and to find adverts that lead to a higher click-through rate.

The situation changes if a policy is conditioned on some input. If the policy for advert selection is conditioned on some user context vector, then exploration is induced by the context. Chapelle and Li (2011) show in corresponding experiments that this exploration induced

[39] Note that the first evaluation for DR and $\hat{c}$DR occur later and less frequently because in contrast the other objectives, the batch size for these estimators is $30k$, which was chosen on the development.

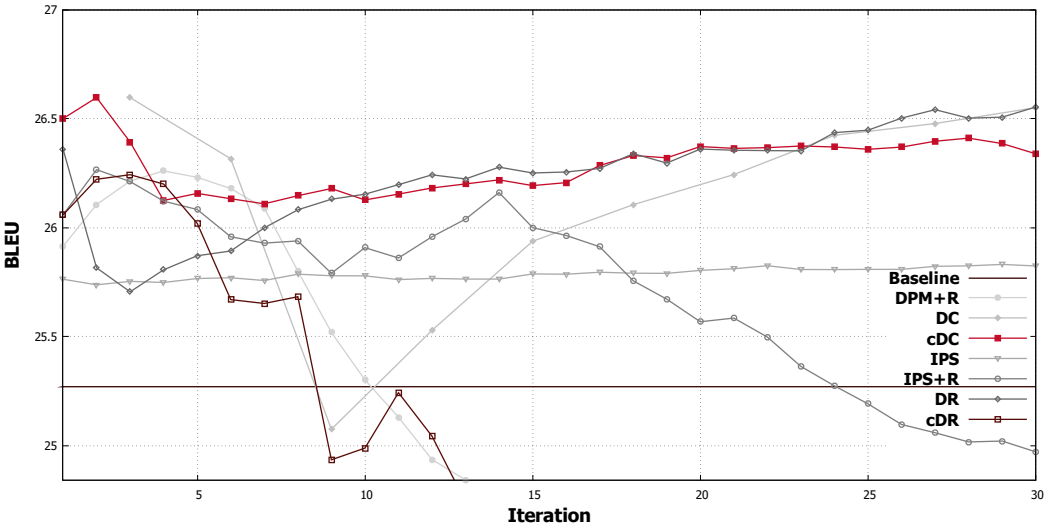


Figure 7.6.: BLEU performance on the News corpus test set after every iteration for various counterfactual objectives.[40]

by context allows their deterministic setup to outperform a random baseline, but not the stochastic approach. This result is consistent with Langford et al. (2008), who state that a deterministic policy is feasible, as long as all actions are explored sufficiently. Similarly, Bastani et al. (2017) prove for a two-armed contextual bandit that a greedy policy can achieve optimal asymptotic regret if the input context is diverse enough.

The context diversity that is needed for deterministic policies to succeed, is naturally given for sequence-to-sequence task in NLP. The natural language sentences that serve as inputs, can be decomposed into individual tokens. As a consequence, identical input tokens can result in different output tokens because the choice of the output token is also conditioned on the other tokens appearing in an input sequence. Across the entire log, this inherently produces a natural exploration caused by the input tokens.

The exploration is further amplified because the output structure can also be decomposed, which stands in contrast to outputs of other tasks, e.g. adverts cannot be further decomposed. Because the same output token can appear in different output sentences, this token will obtain a series of varying sentence-level rewards across the collected log. As a result, it is possible to learn in which context this token is appropriate and in which it isn't. The diversity at both input and output token level, induces enough exploration for deterministic logging to perform as well as the stochastic scheme for sequence-to-sequence tasks in NLP. We confirmed this intuition empirically in Section 7.3.3.

[40] Note that the first evaluation for DC occurs later and less frequently because in contrast the other objectives, the batch size for this estimator is $30k$, which was chosen on the development.

Conclusion

In a counterfactual off-policy learning setup, a deployed system logs user input, system output and an associated reward. The resulting log can be used to either evaluate a different model or to improve a model. To this end, counterfactual estimators have been formulated in previous literature. This offline approach is safer than modifying a system while it is deployed because it could deteriorate without notice and cause monetary loss. Furthermore, in the case of policy evaluation, more policies can be tested than in a typical A/B test setup. For policy learning, it is possible to test various hyperparameters and settings and verify the final policy on separate test sets before deployment.

However, all previous counterfactual estimators assume that the log was created by a stochastic process that samples from the policy's underlying distribution. This is unfortunately not a feasible strategy for deployed sequence-to-sequence systems such as machine translation. In such scenarios, showing an output other than the most likely under the current system is dangerous and could lead to user dissatisfaction and monetary loss. This results in deterministic logs, which stand in contradiction with the theoretical assumption that logs need to be created stochastically to have sufficient exploration of the output space. To investigate this issue, we presented new estimators for a deterministic setup. Using simulation experiments allowed us to directly compare these deterministic estimators with their stochastic counterparts.

Experiments were conducted on a domain adaptation scenario for machine translation with two separate language pairs and target domains. Policies were improved in a hypergraph redecoding setup for a linear SMT model using either deterministic or stochastic logs with corresponding counterfactual estimators. All estimators were able to learn new policies which significantly outperform the out-of-domain baseline that created the logs. The best deterministic and the best stochastic estimators showed no significant difference in the resulting policies. This seeming contradiction can be explained with implicit exploration that naturally occurs in sequence-to-sequence tasks for NLP and is a great positive result for industrial production systems. We were also able to show that counterfactual learning is possible for complex structured prediction problems with large output spaces, such as machine translation.

For both the stochastic and the deterministic case, we showed intuitively and with a mathematical proof that the IPS and DPM estimators exhibit degenerate behaviour. This degeneracy can be resolved by defining a probability distribution over the underlying log using a reweighting multiplicative control variate. However, the resulting new estimators, IPS+R and DPM+R, suffer from a different degenerate behaviour, for which we also give both an intuitive explanation and a mathematical proof. This degenerate behaviour can be solved by introducing a DM predictor as an additive control variate which results in the estimators DR and DC. These latter estimators can be further optimised by setting the linear interpolation scalar $\hat{c}$ optimally. It is these final estimators, $\hat{c}$ DR and $\hat{c}$ DC, that perform best for stochastic and deterministic logging, respectively.

Our experiments show promising results for using counterfactual learning in production setups. However, there are still two issues. First, state-of-the-art models for sequence-to-

sequence tasks are non-linear neural networks, but counterfactual learning has so far only been applied to linear models. Second, the feedback in these experiments is merely simulated. A production system would not have gold targets available to simulate feedback with. The feedback needs to be elicited directly from human users instead. We aim to solve both issues in the following chapter. We also move from machine translation to the question-answering task defined by the NLmaps corpus.

Chapter 8

Question-Answering: Learning from Human Bandit Feedback

Obtaining reliable feedback from human users is a crucial aspect for counterfactual learning (see also Section 1.3) in a real-world scenario. We introduce a setup where such human feedback can be collected efficiently to improve a semantic parser for the Question-Answering (QA) task defined by the NLmaps v2 corpus. We assume a small amount of supervised data is available to train an initial parser and this parser is then used to parse further questions. Given a question, the initial parser returns a parse which can be executed against a database to obtain an answer. The answer is presented to the user and we would like to elicit feedback from the user which can then be used to improve the parser in a counterfactual off-policy learning setup.

However, it is often not easily possible for the user to verify the correctness of an answer. For example, the correct answer to the question "*How many hotels are there in Paris?*" would be 951 hotels and it would be too cumbersome to verify this number by manually counting. Similarly, we cannot obtain feedback for the parse because a non-expert user does not know the underlying MRL. Instead, we present a novel approach: the parse is automatically converted into a set of human-understandable statements. Each statement can easily be verified as correct or incorrect by non-expert users. We show that counterfactual learning is possible using this setup to collect feedback from human users.

The described feedback setup can easily be integrated into virtual personal assistants that employ a semantic parsers. Collecting feedback from users is for example also employed for the task of machine translation by Google Translate[41], where users can suggest a different translation and, once submitted, this data is "used to improve the translation quality" (see Figure 8.1). Alternatively, our feedback setup could be used in a crowd worker setup for QA tasks where both the annotation of parses as well as the collection of gold answers are impractical. For example, in the OSM domain, the underlying MRL is only known by experts and answer sets can be open-ended, fuzzy or too large to enumerate without error or within reasonable time constraints. However, using our feedback collection method, a human can easily provide feedback for a question-parse pair. In the vast he majority of the

[41] `translate.google.com`, 1st September 2018

cases, feedback for one question-parse pair can be provided in less than 10 seconds, which shows that our method is very efficient.

Figure 8.1.: Google Translate interface for editing a translation which is "used to improve the translation quality", Screenshot taken on 1st September 2018.

Neural networks are the state-of-the-art models for sequence-to-sequence tasks. In the previous chapter, we applied counterfactual learning to linear models. Here, we show that counterfactual learning is also possible if the underlying model is a non-linear neural network.[42] But with the move to neural networks, we can no longer employ batch updates based on the entire training data due to hardware limitations. Instead, neural networks are trained using stochastic gradient ascent with minibatches. However, the DPM+R estimator assumes batch updates after the entire training data as been processed. We introduce a new estimator, One-Step-Late Reweighted Deterministic Propensity Matching (DPM+OSL), which retains all advantages of DPM+R and is applicable to stochastic (minibatch) gradient ascent. In experiments we can show that in a stochastic gradient ascent setup with small minibatches DPM+OSL is superior to DPM+R.

Additionally, we modify the DPM estimator to decompose over tokens, leading to the Token-level Deterministic Propensity Matching (DPM+T) estimator. This is a natural extension for neural networks as they produce output sequences token by token. It also enables us to take advantage of the feedback collection setup where a parse is broken up into a set of statements. Each statement is produced by a specific set of tokens in the parse. An estimator that decomposes over tokens is thus able to assign different rewards to different tokens. This allows us to assign blame correctly in an incorrect parse and enables us to learn from partially correct parses.

Finally, combining both new estimators leads to the Token-Level One-Step-Late Reweighted Deterministic Propensity Matching (DPM+T+OSL) estimator. This estimator is able to outperform the baseline semantic parser by nearly 1 percentage point in answer F1 score using feedback collected from human users. To show that our approach scales to larger amounts of feedback, we also present a large-scale experiment with simulated feedback where we can gain 7.45 points in F1 score over the baseline. In both cases, this estimator is able to out-

[42] The feasibility of employing counterfactual learning on neural networks has been shown contemporaneously by Joachims et al. (2018) on a separate task.

perform the strong Bandit-to-Supervised (B2S) baseline, where the subset of fully correct question-parse pairs are treated as a supervised data set.

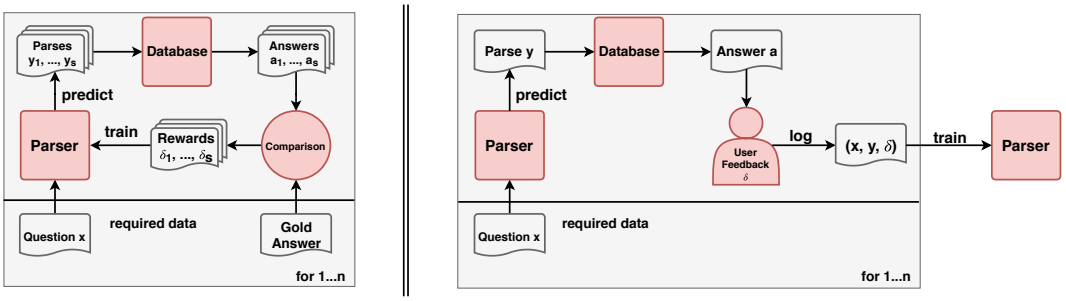


Figure 8.2.: Graphical overview of two semantic parsing setups, once with gold answers available and once without. Left: Common semantic parsing setup where both questions and gold answers are available. The parser attempts to find correct parses by producing multiple parses, obtaining corresponding answers, and comparing them against the gold answer. Right: In our setup, a question does not have an associated gold answer. The parser outputs a single parse and the corresponding answer is shown to a user who provides some feedback. Such triplets are collected in a log which can be used to train a parser.

The success of neural networks for semantic parsing has previously been demonstrated using either question-parse pairs (Jia and Liang, 2016; Dong and Lapata, 2016) or question-answer pairs (Neelakantan et al., 2017). If only question-answer pairs are available, the semantic parser has to suggest parses which are then executed against a database and the resulting answer is compared to the gold answer. Based upon this comparison, parses are assigned rewards and the semantic parser is updated accordingly (see left half of Figure 8.2). We explored this scenario in Chapter 6.

In contrast to the setup in Chapter 6, in this chapter our counterfactual setup assumes that no gold answers are available. Furthermore, only one output of the parser can be judged because rewards are obtained from users who interact with only one output. The obtained reward, together with the associated question-parse pair, is collected in a log and updates to a parser are performed off-policy (see right half of Figure 8.2). In semantic parsing, it is crucial that the chosen parse is the most likely parse under the logging policy of the parser because typically only one or a small number of parses lead to the correct answer. Since we showed in the previous chapter that implicit exploration is sufficient for sequence-to-sequence tasks in NLP, we focus only on deterministic logging in this chapter.

Using weak feedback to improve a semantic parser has previously been shown to be successful (Goldwasser and Roth (2013); Artzi and Zettlemoyer (2013); *inter alia*). Our work most closely resembles the work of Iyer et al. (2017), who make the assumption of only being able to judge one output. They improve their parser using simulated and real user feedback. Parses with negative feedback are given to experts to obtain the correct parse. Corrected parses and parses with positive feedback are added to the training corpus and learning continues with a MLE objective (see Equation 2.28 in Chapter 2). In contrast to this, we show that such a bandit-to-supervision approach can be outperformed by off-policy

bandit learning from partially correct parses. Similar to our feedback collection method, Yih et al. (2016) proposed a user interface for the Freebase database that enables a fast and easy creation of parses. However, in their setup the worker still requires expert knowledge about the Freebase database, whereas in our approach feedback can be collected freely and from any user interacting with the system.

The main contributions of this chapter are as follows:

- Application of counterfactual policy learning to a non-linear neural network for a QA task using bandit feedback from human users.
- Modification the DPM+R estimator to be applicable for stochastic (minibatch) gradient ascent, leading to the DPM+OSL estimator.
- Definiton of the new counterfactual estimator DPM+T where rewards can be decomposed over the tokens in the output sequence. Combined with the DPM+OSL estimator, we also formulate the DPM+T+OSL estimator.
- Implementation of the estimators:
  DPM, DPM+R, DPM+OSL, DPM+T and DPM+T+OSL in Nematus.
- Designing an user interface to collect real human bandit feedback for counterfactual learning.

The work presented in this chapter has previously been published in (Lawrence and Riezler, 2018).

The structure of this chapter is as follows: In Section 8.1 we introduce extensions to the DPM estimator, namely DPM+OSL, DPM+T and DPM+T+OSL. The experimental setup to collect bandit token-level feedback from human users is presented in Section 8.2. The human feedback collection and empirical results for policy learning using the human feedback can be found in Section 8.3. This section also includes an experiment using large-scale simulated feedback, an error analysis, a comparison between DPM+R and DPM+OSL and an analysis of the optimal one-step-late strategy. Finally, comparing results from this chapter and Chapter 6, we conclude by drawing a direct comparison between response-based on-policy and counterfactual off-policy learning.

8.1. Counterfactual Estimator Extensions

For convenience, we repeat here the general notation of the counterfactual learning setup introduced in the previous chapter: We assume an input $x \in \mathbf{X}$ that is mapped to an output $y \in \mathbf{Y}(x)$ by a parametrised policy. Furthermore, there exists a feedback function $\delta(x, y)$ that assigns a reward to (x, y) pairs. Based on the logging policy μ, a log of triplets can be collected: $\mathcal{D} = \{(x_t, y_t, \delta_t)\}_{t=1}^{n}$, where y_t is an output produced by μ given an input x_t and δ_t is the corresponding feedback. Note that on the QA task presented here, we assume that rewards are either one or zero, i.e. $\delta_t \in \{0, 1\}$. As a consequence of this restriction, the degenerate behaviour described for DPM in Section 7.2 cannot occur here: all rewards $\delta_t > 0$

are rewards of 1 and thus the corresponding sample should always be reinforced and have its probability raised. This makes DPM a viable estimator in this setup.

8.1.1. One-Step-Late (OSL) Reweighting

DPM+R employs a multiplicative control variate Y to reduce the variance of DPM. In addition to the variance reduction, the reweighting over the log $\mathcal{D}$ can be seen as defining a probability distribution over the log. Consequently, during policy learning, the DPM+R estimator avoids allocating probability mass to low reward outputs of the log because this takes away probability mass from high reward outputs. Due to these two advantages, DPM+R promises to be a better estimator than DPM, even though the degenerate behaviour of DPM does not occur for the QA task due to $\delta_t \in \{0, 1\}$.

Unfortunately, the DPM+R estimator cannot be used in stochastic minibatch learning, which is used to train neural networks. The multiplicative control variate Y introduces a bias of order $O(\frac{1}{n})$ that decreases as n increases (Swaminathan and Joachims, 2015b). To keep the bias as low as possible, the empirical estimate of Y should be based on the entire log $\mathcal{D}$ of size n, i.e. $Y = \frac{1}{n}\sum_{t=1}^{n} \pi_w(y_t|x_t)$. Defining the reweighting variable Y only over a small minibatch of size m, with $m \ll n$, is not sufficient because the introduced bias is too high. In fact, this naive application of DPM+R in a minibatch setup, results in a worse performance than not employing the multiplicative control variate at all (see corresponding experiment in Section 8.3.4). At the same time, minibatches that would be large enough are infeasible for large neural networks due to hardware limitations.[43]

We present a new estimator that will retain all desirable properties of DPM+R, but can be applied to stochastic learning with small minibatches. Updates are performed using minibatches of size m and these minibatches are reweighted by a control variate Y that is estimated on the entire training data of size n. By ensuring that Y is updated regularly, both advantages of the DPM+R estimator can be preserved. Updating Y after every minibatch m would be quite costly, thus we propose to update Y asynchronously. Using some past parameters w', Y is estimated on the entire log:

$$Y = \frac{1}{n}\sum_{t=1}^{n} \pi_{w'}(y_t|x_t). \tag{8.1}$$

An analogous strategy has previously been presented by Green (1990) for EM algorithms and inspired by their name, we refer to our new estimator as One-Step-Late Reweighted Deterministic Propensity Matching (DPM+OSL):

[43] For example, for counterfactual learning with linear models (see Chapter 7) we employ minibatch sizes of $10k$ and $30k$, whereas for a typical sequence-to-sequence neural model at most minibatches of size 200 fit on a GPU with 8GB of RAM.

$$\mathcal{V}_{DPM+OSL}(\pi_w) = \frac{1}{m}\sum_{t=1}^{m}\delta_t \bar{\pi}_{w,w'}(y_t|x_t) \tag{8.2}$$
$$= \frac{\frac{1}{m}\sum_{t=1}^{m}\delta_t \pi_w(y_t|x_t)}{\frac{1}{n}\sum_{t=1}^{n}\pi_{w'}(y_t|x_t)}.$$

The gradient for optimizing the DPM+OSL estimator with respects to w may be found in Table 8.1 (also see Appendix B) and corresponding pseudo-code in Algorithm 9.

Algorithm 9 Pseudo-code for minibatch off-policy learning from logs with OSL reweighting.

Input: collected log $\mathcal{D}$, minibatch size m, learning rate η, initial policy π_w, OSL update counter $\bar{o}$
$o = 0$
repeat
  for $t = 0, \ldots, m$ **do**
    Sample input-output pair (x_t, y_t) from log $\mathcal{D}$
    Calculate $\pi_w(y_t|x_t)$
    $o \mathrel{+}= 1$
    if $o \,\%\, \bar{o} = 0$ **then**
      Recalculate reweighting variable Y (see Equation 8.1)
    end if
  end for
  Calculate $\mathcal{V}(\pi_w)$
  $w = w + \eta \nabla_w \mathcal{V}(\pi_w)$
until Stopping Criterion Reached

8.1.2. Incorporating Token-Level Feedback

We intend to collect the feedback for our QA task by converting the parse suggested by the parser into a set of human-understandable statements. The statements are extracted from spans of tokens in the parse and each statement can be traced back to the tokens in the parse that created them. Given all statements for a parse, one could derive a reward for the entire parse. However, with the given setup it is possible to assign different rewards to different tokens in the parse. This is a powerful feature because it allows us to learn from partially correct parses.

To benefit from the design of our token-level feedback task for counterfactual learning, we define an estimator that can assign rewards at the token level. For this, we move to log probabilities, which allows us to decompose the sequence into a sum over the individual tokens. If left as a product of probabilities, a sequence's probability would become zero if a single token is incorrect, i.e. $\exists \delta_j = 0$, and thus the entire sequence would be ignored in an update. By employing log probabilities, we can learn from partially correct parses because tokens can be manipulated individually, which allows us to encourage tokens with positive feedback. Thus, for policies that decompose over tokens, we can define the Token-level Deterministic Propensity Matching (DPM+T) estimator:

$$\mathcal{V}_{DPM+T}(\pi_w) = \frac{1}{n}\sum_{t=1}^{n}\left(\sum_{j=1}^{|y_t|}\delta_{t,j}\log\pi_w(y_{t,j}|y_{t,<j},x_t)\right), \tag{8.3}$$

where $y_{t,<j} = y_{t,1}, y_{t,2}\ldots y_{t,j-1}$.

Combining both the one-step-late reweighting and the token-level feedback, leads to the Token-Level One-Step-Late Reweighted Deterministic Propensity Matching (DPM+T+OSL) estimator:

$$\mathcal{V}_{DPM+T+OSL}(\pi_w) = \frac{\frac{1}{m}\sum_{t=1}^{m}\left(\sum_{j=1}^{|y_t|}\delta_{t,j}\log\pi_w(y_{t,j}|y_{t,<j},x_t)\right)}{\frac{1}{n}\sum_{t=1}^{n}\pi_{w'}(y_t|x_t)}. \tag{8.4}$$

The gradients for the DPM+T and DPM+T+OSL estimators may be found in Table 8.1 (also see Appendix B).

$\nabla_w\mathcal{V}_{DPM+OSL} = \frac{1}{m}\sum_{t=1}^{m}\delta_t\bar{\pi}_{w,w'}(y_t\|x_t)\nabla_w\log\pi_w(y_t\|x_t).$
$\nabla_w\mathcal{V}_{DPM+T} = \frac{1}{n}\sum_{t=1}^{n}\left(\sum_{j=1}^{\|y_t\|}\delta_{t,j}\nabla_w\log\pi_w(y_{t,j}\|y_{t,<j},x_t)\right).$
$\nabla_w\mathcal{V}_{DPM+T+OSL} = \frac{\frac{1}{m}\sum_{t=1}^{m}\left(\sum_{j=1}^{\|y_t\|}\delta_{t,j}\nabla_w\log\pi_w(y_{t,j}\|y_{t,<j},x_t)\right)}{\frac{1}{n}\sum_{t=1}^{n}\pi_{w'}(y_t\|x_t)}.$

Table 8.1.: Gradients of various counterfactual estimators.

Type	Explanation
Town	OSM tags of "*area*"
Reference Point	OSM tags of "*center*"
POI(s)	OSM tags of "*search*" if "*center*" is set, else of "*nwr*"
Question Type	Arguments of "*qtype*"
Proximity : Around/Near	If "*around*" is present
Restriction : Closest	If "*around*" and "*topx*" are present
Distance	Argument of "*maxdist*"
Cardinal Direction	"*north*", "*east*", "*south*" or "*west*" are present

Table 8.2.: All possible statements types, which are used to transform a parse into a human-understandable block of statements.

8.2. Collecting Human Feedback

For many questions in the NLmaps corpus, it is difficult for humans to judge whether an answer given to their question is correct or not. Similarly, non-experts cannot judge whether a parse is correct or not for a given question. Instead we propose to automatically convert the parse into a set of human-understandable statements. These statements can be individually judged as correct or incorrect by any non-expert user. Once a user has judged a set of statements, the feedback given for each statement can automatically be mapped back to the original tokens in the parse that correspond to the statement. If a statement is marked as correct, the underlying tokens receive a reward of 1, and 0 otherwise.

There are 8 different statement types. Each is triggered by the shape of the parse and certain tokens. For example, the token "*area*" triggers the statement type "*Town*" and the arguments of "*area*" in the parse are used to populate the statement, e.g. "*area*(*keyval*(*name*,'*Paris*'))" becomes "*Town : Paris*". All statement types and their triggers can be found in Table 8.2. The set of statements for one parse is presented in a user interface as one HTML form. Each statement receives two radio buttons with the options "*Yes*" and "*No*" to indicate whether that statement is true or not. The submission of a form is only accepted if a radio button has been selected for each statement. A screenshot with a correctly filled out example may be found in Figure 8.3.

The OSM tags that appear in the statements are generally human-understand-able. But in case the meaning might be unclear to the user, we provide an automatically extracted description for each OSM tag and key. The descriptions are extracted from the corresponding Wikipedia page on the OSM Wiki.[44] It is made available to the user in the form of a tool tip that appears when hovering over OSM tags or keys. The existence of a tool tip is indicated by a bold blue font in the form. For example, hovering over "*amenity : parking*" in Figure 8.3, will show a small pop-up box with the description: "*A place for parking cars*".

In our experiments, we prepared forms that were then filled out by recruited human users. In the future, we envision to incorporate the feedback form directly into the online natural language interface to OSM[45]. At the moment, a semantic parser might parse the question "*How many hotels are there in Paris?*" with the tag "*amenity=restaurant*" in the parse, rather than the correct "*tourism=hotel*". A user would remain ignorant that the parser misunderstood the question and that a wrong answer has been presented. With the feedback form, the user can verify for their own comfort that their question was understood correctly.

Going one step further, the feedback form could be transformed into an interactive experience. If a user marks a statement as incorrect, the semantic parser can automatically traverse the k-best list for the highest ranking parse where the incorrect statement does not appear and present the new parse and its answer to the user instead. Alternatively, we can allow people to edit the form and directly provide the correct information. In the case of the question type, any user could easily select the correct one in a drop-down menu of the 4 possible question types present in the NLmaps v2 corpus. In the case of OSM tags, a short list from

[44] https://wiki.openstreetmap.org/, 1st September 2018
[45] http://nlmaps.cl.uni-heidelberg.de/

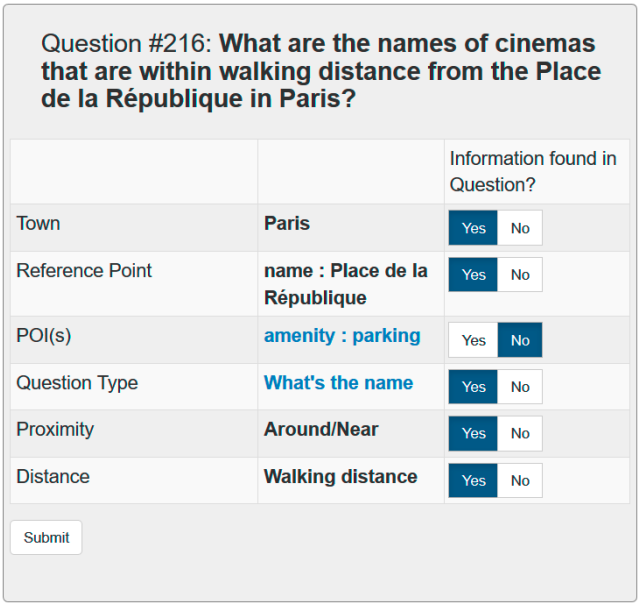


Figure 8.3.: The user interface for collecting feedback from humans with an example question and a correctly filled out form.

the k-best list could be produced and the user could search this list for a fitting tag. Users with knowledge of OSM tags could even directly enter the correct tag.

An interactive user-system setup would deliver a better user experience while simultaneously collecting valuable feedback that can be used to improve performance. Additionally, this is a step towards ensuring that model decisions are visible and understandable to the user. Having greater transparency in the decision-making of artificially intelligent systems has become an important concern to many users. At the same time, the system can learn from the mistakes it makes, rather than remaining ignorant after a dissatisfied user leaves the platform, because they were not able to easily provide feedback and inform the system of their dissatisfaction. Viable and intuitive user-system interactions ensure a bilateral dialogue from which both sides can profit.

8.3. Empirical Results

In our experiments we use the sequence-to-sequence neural network package Nematus (Sennrich et al., 2017), which follows the setup outlined in Section 2.4. Following the approach described in Section 3.2 (see Listing 3.4), we split parses into individual tokens by taking a pre-order traversal of the original tree structure. We use the learning rate optimiser Adadelta (Zeiler, 2012). The model employs 1,024 hidden units and word embeddings of size 1,000. The maximum sentence length is 200 and gradients are clipped to 1.0 if they exceed a value of 1.0. A stopping point is determined by validation on the development set and selecting the point at which the highest evaluation score is obtained. Validation is run after every 100 updates, and each update is made on the basis of a minibatch of size 80.

The evaluation of all models is based on the answers obtained by executing the most likely parse found after a beam search with a beam of size 12. We report the F1 score, which is the harmonic mean of precision and recall. Recall is defined as the percentage of fully correct answers divided by the set size. Precision is the percentage of correct answers out of the set of answers with non-empty strings. Statistical significance between models is measured using an approximate randomisation test (Noreen, 1989).

Our experiment design assumes a baseline neural semantic parser that is trained in fully supervised fashion. For this purpose, we select 2,000 question-parse pairs randomly from the full extended NLMaps v2 corpus. We call this dataset $\mathcal{D}_{sup}$. Using this dataset, a baseline semantic parser is trained in supervised fashion under a MLE objective (see Equation 2.28). It obtains an F1 score of 57.45% and serves as the logging policy μ and as the starting point for counterfactual learning from a log. Furthermore we randomly split off 1,843 and 2,000 pairs for a development and test set, respectively, which mirrors the split used in Chapter 6. This leaves a set of 22,765 question-parse pairs. The questions can be used as inputs and bandit feedback can be collected for the most likely output of the semantic parser. We refer to this dataset as $\mathcal{D}_{log}$.

8.3.1. Learning from Human Feedback

We select a random subset of 1,000 questions from $\mathcal{D}_{log}$. The questions are parsed with the baseline policy μ and the most likely parse for each question is returned. Each parse is transformed into a set of statements that can be judged as correct or not by non-expert users as described in Section 8.2. 5 question-parse pairs are discarded because the underlying parse structure is invalid. We recruited 9 humans to provide feedback for the remaining 995 question-parse pairs. Each pair is purposely judged by one user only to mimic the real-world scenario where we cannot rely on several users asking the same question and providing feedback. The 995 question-parse pairs together with the corresponding feedback constitute the log $\mathcal{D}_{human}$.

To appraise the efficiency of our feedback collection setup, we measure the time a user took from loading the feedback form for one question-parse pair to submitting the fully filled out form. On average, it took a user 16.4 seconds to submit a form, with a standard deviation of 33.2 seconds. Out of all feedback forms, a vast majority of 728 are submitted in less than 10 seconds. This indicates that our feedback collection method is efficient and shows a high promise of being an efficient alternative compared to collecting parse annotations or correct answer sets.

The log $\mathcal{D}_{human}$ is employed as the training data to improve the parser using the counterfactual estimators DPM, DPM+OSL, DPM+T and DPM+T+OSL. For the sequence-level objectives, a parse receives a reward of 1 if all statements are marked correct and 0 otherwise. For the token-level objectives, tokens receive rewards of 1 if the corresponding statement is marked as correct and 0 otherwise. We also consider a B2S setup, where the subset of 531 question-parse pairs marked as fully correct by the users is used as supervised training data and learning continues with MLE. For the OSL update strategy, a separate experiment (see

Subsection 8.3.5) showed the the best trade-off between speed and performance is to update the reweighting variable Y at every validation step.

		F1	Δ F1
1	Baseline	57.45	
2	B2S	57.79 ±0.18	+0.34
3	DPM[1]	58.04 ±0.04	+0.59
4	DPM+OSL	58.01 ±0.23	+0.56
5	DPM+T[1]	58.11 ±0.24	+0.66
6	DPM+T+OSL[1,2]	**58.44**±0.09	+0.99

Table 8.3.: Human Feedback: F1 scores on the NLMaps v2 test set for various counterfactual objectives. Results are averaged over three independent runs. Best results are indicated in **bold face**. Statistical significance of system differences at $p < 0.05$ are indicated by experiment number in superscript.

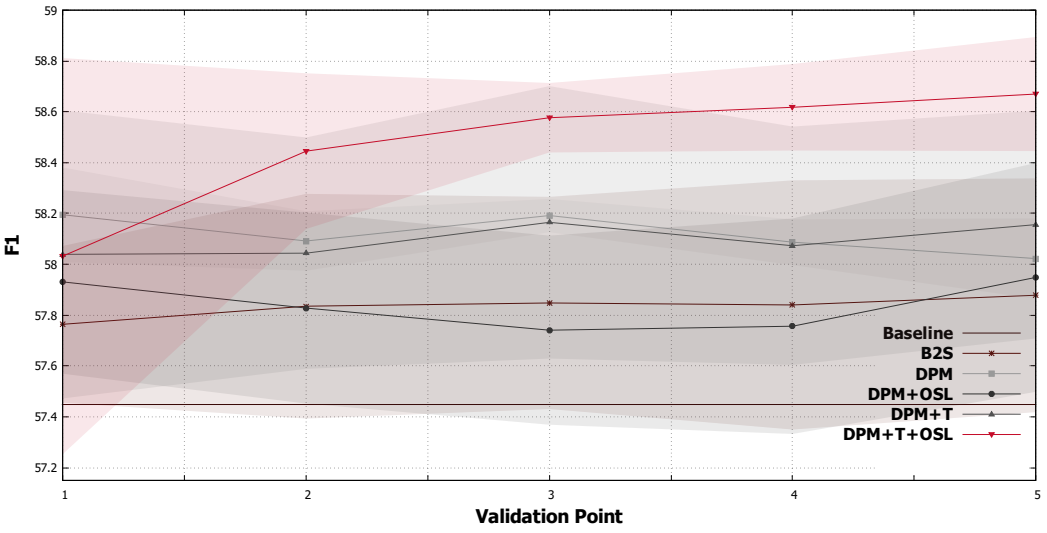


Figure 8.4.: Human Feedback: F1 performance on the test set at the validation points for various counterfactual objectives.

Results, averaged over three runs, are reported in Table 8.3. The B2S model can slightly improve upon the baseline but not significantly. DPM improves further, significantly beating the baseline. Using the multiplicative control variate modified to use OSL updates does not seem to help in this setup. By moving to token-level rewards, it is possible to learn from partially correct parses. These partially correct parses provide valuable information that is not present in the subset of correct answers employed by the previous models.

Optimizing DPM+T leads to a slight improvement and, combined with the multiplicative control variate, DPM+T+OSL yields an improvement of about 1.0 percentage point in F1 score upon the baseline. It beats both the baseline and the B2S model by a significant margin. This is a great result as we can now conclude that counterfactual learning is possible, both for non-linear models and when the feedback is elicited from real human users.

Furthermore, this experiment again re-confirms the feasibility of deterministic logging for sequence-to-sequence tasks in NLP.

Figure 8.4 reports the results on the test set for every validation point. It shows that DPM, DPM+T and DPM+T+OSL always lie numerically above B2S and that DPM+T+OSL beats B2S significantly from the third validation point onward.

8.3.2. Learning from Large-Scale Simulated Feedback

We want to investigate whether the results scale if a larger log is used. Thus, we use μ to parse all 22,765 questions from $\mathcal{D}_{log}$ and obtain for each the most likely output parse. Using gold parses to simulate feedback, we assign a sequence level reward of 1 if the parse is identical to the gold parse, 0 otherwise. We simulate token-level rewards by iterating over the indices of the output and assigning a feedback of 1 if the same token appears at the current index for the gold parse, 0 otherwise. An analysis of $\mathcal{D}_{log}$ shows that 46.27% of the parses have a sequence level reward of 1 and are thus completely correct. This subset is used to train the B2S model using MLE.

Experimental results for the various setups, averaged over three runs, are reported in Table 8.4. The B2S model outperforms the baseline model by a large margin, yielding an increase in F1 score by 6.24 percentage points. Employing the DPM estimator also produces a significant increase over the baseline, but its performance falls short of the stronger B2S baseline. Using the DPM+OSL estimator leads to a substantial improvement in F1 score over optimizing DPM, but it still falls slightly short of the B2S baseline. Token-level rewards are again crucial to beat the B2S baseline significantly. DPM+T is already able to significantly outperform B2S in this setup and DPM+T+OSL can improve upon this further. Figure 8.5 reports the results on the test set at every validation point and confirms that DPM+T and DPM+T+OSL significantly outperform B2S at various validation points.

		F1		Δ F1
1	Baseline	57.45		
2	B2S^{1,3}	63.22	±0.27	+5.77
3	DPM^{1}	61.80	±0.16	+4.35
5	DPM+OSL^{1,3}	62.91	±0.05	+5.46
6	DPM+T^{1,2,3,5}	63.85	±0.20	+6.40
7	DPM+T+OSL^{1,2,3,5}	**64.41**	±0.05	+6.96

Table 8.4.: Large-Scale Simulated Feedback: F1 scores on the NLMaps v2 test set for various counterfactual objectives. Results are averaged over three independent runs. Best results are indicated in **bold face**. Statistical significance of system differences at $p < 0.05$ are indicated by experiment number in superscript.

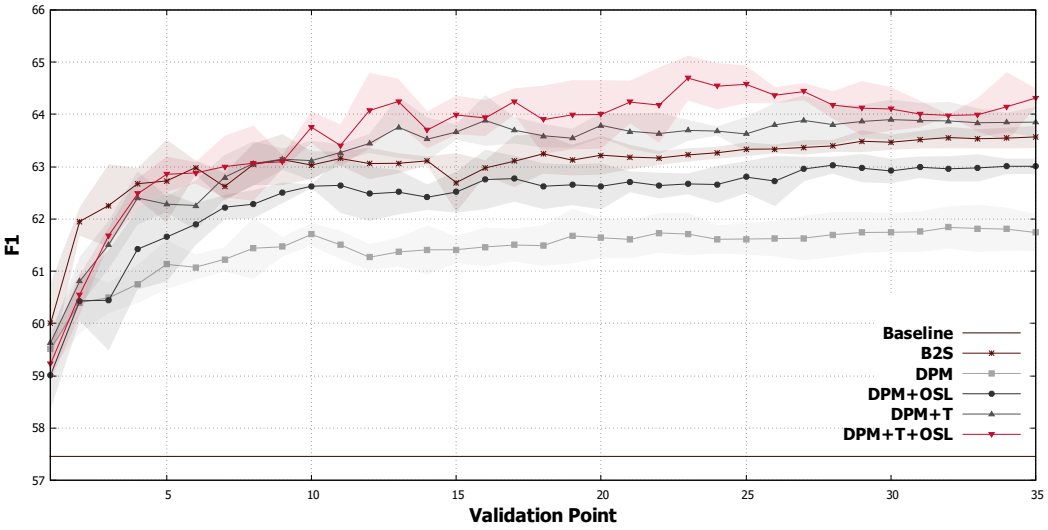


Figure 8.5.: Large-Scale Simulated Feedback: F1 performance on the test set at the validation points for various counterfactual objectives.

8.3.3. Error Analysis

Following the error analysis setup introduced in Section 4.3.2, we present in Table 8.5 the type of errors made by the baseline model as well as the DPM+T+OSL model, for both human and simulated large-scale feedback. For both DPM+T+OSL models there is a decrease in both OSM tag and Qtype errors. The decrease is larger for the simulated feedback, which is in accordance with the larger amount of available feedback. For the human feedback model, there is no change in the number of Skeleton errors, whereas this error type exhibits the largest decrease in percentage for the simulated feedback setup. For the error types Wrong Distance and Invalid Parse we observe a slight increase of errors for both DPM+T+OSL models. But overall the human feedback model can reduce the number of errors by slightly more than 3% and, with more feedback points available, the simulated model exhibits an error reduction of nearly 19%.

	Baseline	Human	Large-scale
OSM tag	1,248	1,199 $_{\downarrow 3.93\%}$	1,025 $_{\downarrow 17.87\%}$
Qtype	135	131 $_{\downarrow 2.96\%}$	108 $_{\downarrow 20.00\%}$
Wrong Dist.	3	6 $_{\uparrow 100.00\%}$	8 $_{\uparrow 166.67\%}$
Skeleton	70	70 $_{\rightarrow 0.00\%}$	38 $_{\downarrow 45.71\%}$
Invalid Parse	7	9 $_{\uparrow 28.57\%}$	12 $_{\uparrow 71.43\%}$
Total	1,463	1,415 $_{\downarrow 3.28\%}$	1,191 $_{\downarrow 18.57\%}$

Table 8.5.: Overview of which typed of error the baseline makes and the percental in- or decrease for DPM+T+OSL with regards to the baseline, using the human and large-scale simulated log, respectively.

8.3.4. Comparison to Naive Reweighting

We also implement DPM+R naively for stochastic minibatch learning. In this implementation the reweighting variable Y is simply calculated on the current minibatch of size m. Employing the setup for the large-scale experiment, we report in Table 8.6 the results for this naive DPM+R implementation and compare it to DPM and DPM+OSL. Using a naive DPM+R leads to worse results than not using a control variate, i.e. DPM. This confirms empirically that calculating Y on small minibatches introduces large bias and the variance reduction effect cannot counter this sufficiently. In contrast to this, calculating Y on the basis of the entire log of size 22,766, DPM+OSL can significantly outperform DPM.

		F1	Δ F1
1	Baseline	57.45	
2	DPM+R^{1}	60.94 $\pm$0.01	+3.49
3	DPM^{1,2}	61.80 $\pm$0.16	+4.35
4	DPM+OSL^{1,2,3}	**62.91**$\pm$0.05	+5.46

Table 8.6.: Large-Scale Simulated Feedback: F1 scores on the NLmaps v2 test set for the comparison between a naive use of the reweighting objective (DPM+R) to using no reweighting (DPM) and using one-step-late reweighting (DPM+OSL). Results are averaged over three independent runs. Best results are indicated in **bold face**. Statistical significance of system differences at $p < 0.05$ are indicated by experiment number in superscript.

8.3.5. Varying the OSL Update Frequency

Using the DPM+T+OSL estimator and the simulated feedback setup, we vary the frequency of updating the reweighting variable Y. Results are reported in Table 8.7. Calculating Y only once at the beginning leads to a near identical result in F1 score as not using a control variate. The more frequent update strategies, once or four times per epoch, are more effective and lead to higher F1 scores. Updating four times per epoch compared to once per epoch, leads to a nominally higher performance in F1. It has the additional benefit that the re-calculation is done at the same time as the validation, leading to no additional slow down as executing the parses for the development set against the database takes longer than the re-calculation of Y. Updating after every minibatch is impractical as it slows down training too much. Compared to the setup of updating at every validation, one epoch takes approximately an additional 5.5 hours when updating after every minibatch. Over all 30 validation points this equates to an additional 165 hours, i.e. over 6 days.

	OSL Update	F1	Δ F1
1	no OSL (DPM+T)	63.85 ±0.2	
2	once	63.82 ±0.1	−0.03
3	every epoch	64.26 ±0.04	+0.41
4	every validation / 4x per epoch[2]	**64.41**±0.05	+0.56
5	every minibatch	N/A	N/A

Table 8.7.: Large-Scale Simulated Feedback: F1 scores on the NLMaps v2 test set for DPM+T and DPM+T+OSL with varying OSL update frequencies. Results are averaged over three independent runs. Updating after every minibatch is infeasible as it significantly slows down learning. Best results are indicated in **bold face**. Statistical significance of system differences at $p < 0.05$ are indicated by experiment number in superscript.

8.3.6. Comparison: Response-Based On-Policy & Counterfactual Off-Policy Learning

In Chapter 6, we employ response-based on-policy learning objectives on the same NLMaps v2 training data and upon the same baseline semantic parser. This allows us to draw a direct comparison between the best response-based on-policy learning objective, RAMP+T, and the best counterfactual off-policy learning objective, DPM+T+OSL. Results are reported in Table 8.8. With an additional increase of 5.04 percentage points in F1 score, RAMP+T can significantly outperform DPM+T+OSL. This difference can be easily understood because response-based on-policy learning has a stronger learning signal available: it can leverage the existing gold answers to obtain feedback for arbitrarily many model outputs. Counterfactual off-policy learning on the other hand can access only one feedback point for one model output and, furthermore, this model output is biased by the logging policy.

In terms of training speed, counterfactual off-policy learning is faster than response-based on-policy learning. For counterfactual off-policy learning, the log and thus all necessary feedback points are already collected when training begins. But response-based on-policy learning tries out several model outputs per input. This leads to a significant slow-down if the process of obtaining feedback for a model output is time consuming. With up to 10 parses per input that need to be executed against the OSM database, response-based on-policy learning is an order of magnitude slower than counterfactual off-policy learning in our experiments.

If speed is no consideration, the choice between response-based on-policy and counterfactual off-policy learning reduces to how expensive it is to obtain gold targets. On the OSM domain, both obtaining gold parses or gold answers is impractical. The underlying MRL is only known to a handful of experts and thus it is expensive to obtain new gold parses. But obtaining gold answers is outright infeasible because in many instances the answer set is open-ended, fuzzy or too large to be enumerated in a reasonable amount of time and

		F1	Δ F1
1	Baseline	57.45	
2	DPM+T+OSL[1]	64.41±0.05	+ 6.96
3	Ramp+T[1,2]	**69.45**±0.53	+12.00

Table 8.8.: Comparison: F1 scores on the NLmaps v2 test set for the best response-based on-policy learning objective, Ramp+T, and the best counterfactual off-policy learning objective, DPM+T+OSL. Results are averaged over three independent runs. Best results are indicated in **bold face**. Statistical significance of system differences at $p < 0.05$ are indicated by experiment number in superscript. *Note:* The F1 score of Ramp+T differs to the score reported in Table 6.2 because here, three independent runs, rather than two, have been run for a direct comparison to the three independent runs of DPM+T+OSL.

without error. In such scenarios, obtaining feedback to model outputs from human users can offer a viable alternative. This feedback collection can either be done by recruiting human workers to provide feedback or by incorporating a feedback collection method in a deployed system, where users can provide feedback as they use the system. For the OSM domain, our experiments showed that the feedback for one model output can in most cases be collected in 10 seconds or less from a human user, which proves the efficiency of our proposed method.

Thus, we conclude, counterfactual off-policy learning should be chosen if gold targets are impossible, too time consuming or too expensive to obtain, whereas feedback for model outputs can be easily collected. Otherwise, response-based on-policy learning is a more promising approach because the available gold target offers a stronger learning signal as it allows the model to try out and receive feedback for arbitrarily many outputs.

Conclusion

We demonstrated that counterfactual learning can be used to improve a semantic parser for the QA task defined by the NLmaps v2 corpus. Because deterministic logs are on par with stochastic logs for the machine translation task in the previous chapter, we employed only deterministic logs for the QA task. Deterministic logs are especially important in a deployed QA system because typically only one parse leads to a correct answer. Akin to the machine translation task, counterfactual learning can significantly outperform the semantic parser baseline for the QA task. This is another validation for the feasibility of deterministic logging for sequence-to-sequence tasks.

Additionally, we moved to state-of-the-art neural networks. Counterfactual learning had previously only been applied to linear models. We were able to show that counterfactual learning is also successful if the underlying policy is a non-linear neural network. Moving to neural networks, it is necessary to perform stochastic learning with small minibatches because larger minibatches or one batch consisting of the entire log cannot be used for large,

state-of-the-art neural networks on current hardware. However, DPM+R requires a calculation on the basis of the entire log and can thus not be employed in a stochastic (mini-batch) gradient ascent setting. Because DPM+R offers crucial advantages over DPM, we introduced a new estimator, DPM+OSL, that preserves the advantages of DPM+R and is applicable to stochastic learning. We validated its effectiveness and success over DPM empirically, while also showing that a naive implementation of DPM+R leads to worse results than simply using DPM.

Most crucially, we showed that counterfactual learning is possible if feedback is collected from real human users. In QA tasks, the human user would typically have to provide feedback for either the answer or the parse that the semantic parser produced for a given question. But in many cases, e.g. when many objects have to be counted, the answer cannot easily be judged as correct or incorrect by a human user. Similarly, non-expert users cannot judge a parse as right or wrong because they do not know the underlying MRL. Thus, we proposed a new method for easily collecting feedback from non-expert users: We automatically convert the parse into a set of human-understandable statements. These statements can easily be marked as correct or not by non-experts. Feedback was collected from 9 recruited users and was subsequently used to significantly improve the baseline system. The vast majority of feedback forms were filled out in 10 seconds or less, which demonstrates the efficiency of our proposed approach.

Our feedback collection method offers an additional advantage. Each statement can be traced back to a set of tokens in the parse which produced the statement. Thus, it is possible to use the statements to give rewards to individual tokens. This allows us to assign blame within a parse, enabling us to learn from partially correct parses. To make use of this, we introduced a new counterfactual estimator, DPM+T, that decomposes over tokens. Combining this estimator with the one-step-late approach, leads to the DPM+T+OSL estimator. This estimator was successfully able to significantly outperform a simple Bandit-to-Supervised (B2S) baseline, where all entries in the log that were marked as fully correct, are treated as a supervised data set and training continues with MLE. We also repeated our experiments in a large-scale setting with simulated feedback. Here, we were able to show that the gains over the baseline scale with the amount of available feedback. Additionally, in an error analysis we showed which type of parse errors are reduced by employing counterfactual off-policy learning.

Finally, we were able to draw a direct comparison between response-based on-policy and counterfactual off-policy learning because we employed the same training data and baseline semantic parser in this chapter and in Chapter 6. The comparison allowed us to conclude that response-based on-policy learning offers more potential because with gold answers available in this setup, arbitrarily many model outputs can be tried out. This is a stronger learning signal than the setup given for counterfactual off-policy learning, where feedback is available for only one model output and this model output is biased by the logging policy. However, in certain instances, collecting gold targets might be impossible, too time consuming or too expensive, whereas obtaining feedback for a model output is efficient and easy. This is for example the case in the OSM domain, where gold parses are expensive to obtain because the MRL is only known to a handful of expert users. Similarly, gold answers are infeasible to obtain because the answer sets are often open-ended, fuzzy or too large to

enumerate without error or within reasonable time constraints. In such instances, counterfactual off-policy learning is a viable alternative.

Part III Conclusion

Counterfactual off-policy learning offers an approach of learning from user-system interactions that does not require any gold targets for learning. This is a crucial advantage in scenarios where gold targets are too expensive to obtain, whereas collecting feedback for model outputs is cheap.

We successfully employed the approach to two NLP sequence-to-sequence tasks. For the first task, we tuned a machine translation system in a domain adaptation scenario. With this, we verified that it is possible to use counterfactual learning for tasks with large output spaces. Furthermore, we showed empirically with simulated feedback that deterministic logs are on par with stochastic logs and gave an intuitive explanation of why the implicit exploration of deterministic logging is sufficient for sequence-to-sequence tasks in NLP. In a second task, we improved a semantic parser. On this task, we confirmed that counterfactual learning is possible for neural models. Furthermore, we proved that counterfactual learning is possible if feedback is collected from non-expert human users.

Together, the results from Chapter 7 and Chapter 8 demonstrate that promising gains can be obtained if counterfactual learning is employed in deployed sequence-to-sequence tasks where it is easy and cheap to collect large logs from human-system interactions. In the future, it would be interesting to explore scenarios where the logging policy is not also the starting policy that we want to improve. This might lead to further challenges, but would be an interesting setup for scenarios where a new, better model is found that is fundamentally different to the logging policy. Learning from the log could ensure that the new model is fine-tuned to user preferences before deployment.

Thesis Conclusion

We explored two distinct approaches to improve sequence-to-sequence models for Natural Language Processing (NLP) tasks using feedback obtained for model outputs. In addition, we also introduced a new application where users can use natural language questions to search the geographical database OpenStreetMap (OSM). The basis for this application is the Question-Answering (QA) corpus, NLMaps, and its extension, NLMaps v2, with both of which semantic parsing models were trained. We designed an appropriate graphical web interface and tailored the semantic parsing models for this use. The semantic parsing task was also used to exemplify the two distinct approaches to learn from feedback.

The first approach is response-based on-policy learning, where weak feedback in form of downstream gold targets is available. We first employed it to improve the first model in a two-model pipeline configuration by grounding the model in the final, downstream task. We confirmed its success empirically on a multilingual semantic parsing setup for QA based on the NLMaps corpus and using log-linear models. First, a machine translation system translated a German question into English. Next, a semantic parser maps the English question to a semantic parse. This parse is executed against a database to obtain an answer and the answer can be compared to the available gold answer.

By leveraging feedback from the answer-level comparison, we introduced two algorithms which tailored the machine translation system to work well in conjunction with the semantic parser, improving the overall task performance. The algorithms each use a ramp loss objective where a hope translation and fear translation are identified and, respectively, encouraged and discouraged. The first algorithm only requires the existence of gold answers, whereas the second additionally requires gold reference translations. The latter outperforms the former, as well as another algorithm that assumes only the existence of gold references. However, requiring two gold targets might be too expensive in praxis.

As a second application, we explored if response-based on-policy learning can be used to improve a semantic parser because for many domains it is cheaper to obtain gold answers rather than gold parses. Additionally we moved from log-linear models to neural networks and to NLMaps v2. A semantic parser is improved using the feedback obtained by comparing the answers of parses produced by the model to the gold answer. We lifted several ramp loss objectives to neural networks and compared them to Minimum Risk Training (MRT). We showed that a ramp loss objective is the most effective because it naturally employs the bipolar principle, where negative outputs are discouraged while positive ones are encouraged. Furthermore, we extended this objective to operate at the token level. The novel

objective outperforms all others, including its sequence-level counterpart, by more effectively contrasting the tokens in a fear output against the tokens in the corresponding hope output.

However, response-based on-policy learning ultimately still requires the existence of expensive gold targets, albeit gold targets of a downstream task, which might be easier to obtain. To further alleviate the need for such gold targets, we turned to a second approach that does not require any gold targets.

The second approach is counterfactual off-policy learning. Instead of collecting gold targets, we proposed to collect user-system interaction logs of deployed applications. Such a log records user input, model output and collects feedback from the user for the proposed model output. Using this log, we employed counterfactual reward estimators to improve a target system. This poses a challenging learning scenario because only one model output can receive feedback and this output is biased to the deployed model, leading to a bandit, off-policy learning setup. Prior theory suggests that the log has to be created stochastically by sampling from the logging model. However, sampling is dangerous for sequence-to-sequence tasks in NLP and because of this, the most likely output under the logging system is displayed. This leads to deterministic logging for which theory indicates that it does not offer sufficient exploration of the output space.

We employed both stochastic and deterministic counterfactual estimators to improve a machine translation system with an underlying log-linear model in a domain adaptation task. By simulating the feedback, we were able to directly compare stochastic estimators to their deterministic counterparts. We identified that the best deterministic estimator performs on par with the best stochastic estimator. Our offered explanation of implicit exploration suggests that deterministic logging is sufficient for sequence-to-sequence tasks in NLP because there is enough exploration at the token level, on both input and output side. Additionally, the experiments showed for the first time that counterfactual learning is possible for tasks with large outputs space. Finally, we also analysed how some counterfactual learning estimators can behave in degenerate ways and supplied corresponding proofs.

As a second application, we applied counterfactual off-policy learning to semantic parsing. Mirroring the second task for response-based on-policy learning, we employed the same baseline neural semantic parser based on NLmaps v2. To collect feedback from humans, we presented a method which transforms a semantic parse into a set of human-understandable statements that can be marked as correct or incorrect by non-expert human users. We confirmed empirically that counterfactual learning works on another sequence-to-sequence task for NLP and that learning is possible if the feedback is collected from human users.

With the move to neural networks, which are trained using stochastic gradient ascent with small minibatches, we were no longer able to use a counterfactual estimator that required updates made on the basis of larger batches. We presented an appropriate modification and confirmed its success empirically. Finally, we introduced a new estimator that can leverage token-level feedback, which allows it to outperform its sequence-level counterpart. Our successful application of counterfactual learning to sequence-to-sequence tasks are particularly interesting for commercial applications where the (deterministic) logging of user feedback does not incur any cost and is plentiful.

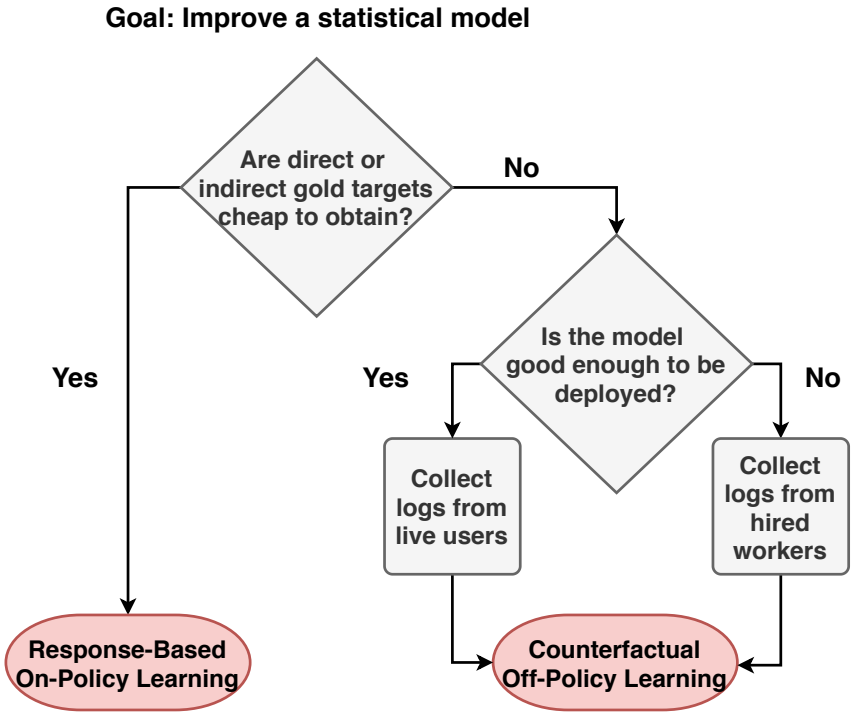


Figure 8.6.: Decision tree to guide whether to employ response-based on-policy learning or counterfactual off-policy learning for a given statistical model that is to be improved.

By posing the same semantic parsing task based on NLmaps v2 for both response-based on-policy and counterfactual off-policy learning, we were able to draw a direct comparison between the two approaches. We saw that the best model from response-based on-policy learning outperforms the best model produced by counterfactual off-policy learning. However, it requires the existence of gold answers which might be too expensive or impossible to obtain. Especially for the OSM domain, it is in many instances infeasible to ask a human worker to produce a gold answer because answer sets are too large or fuzzily defined. In such scenarios, it is much easier to ask humans to give feedback for a model output. Concretely, we proposed a method where non-expert users could typically provide the required feedback for one model output in 10 seconds or less.

The optimal choice between response-based on-policy or counterfactual off-policy learning thus depends on the given problem. Given a statistical model that we want to improve, we can follow the decision tree presented in Figure 8.6. If direct or downstream gold targets can easily be collected, response-based on-policy learning should be preferred because gold targets offer a stronger learning signal, as it allows the model to try out different model outputs and receive feedback for all of them. If however, gold targets are impossible to obtain or their collection is vastly more time consuming than obtaining feedback for model outputs, then counterfactual off-policy learning offers a viable alternative. If the statistical model performs sufficiently well to be deployed, the required logs can be collected freely from live

users who are willing to provide feedback. Otherwise, workers can be hired to provide the required feedback.

In summary, we presented two different approaches to improve sequence-to-sequence models in NLP using feedback given to model outputs. Each approach was employed on two applications (machine translation and semantic parsing for QA) and two different model types (log-linear models and non-linear neural networks). The first approach, response-based on-policy learning, grounds a model in its final, downstream task by trying out various outputs to find an output that leads to positive task feedback. But this approach requires gold targets based on the downstream task, which might be cheaper to obtain than directly supervised gold targets, but are ultimately still costly. The second approach, counterfactual off-policy learning, collects feedback from human-system interactions of a deployed model. It does not require gold targets but poses a more difficult learning scenario because feedback is only available for one output and the output is biased to the deployed model. These factors should be weighed against each other when choosing which approach is more suitable for a given problem.

Overall, both approaches allow us to learn by using weaker feedback signals than directly supervised data. Exploring such approaches is important because nowadays many applications are built upon statistical machine-learnt models for which direct supervision is too costly and time-consuming to obtain. With the trend towards virtual personal assistants serving an increasing number of people on an increasing number of everyday tasks, it will become even more important in the future to adapt to user needs without requiring expensive supervised data. Our two investigated approaches of using feedback obtained for model outputs are an important step in this direction.

Bibliography

If I have seen further it is by standing on the shoulders of Giants.

- Isaac Newton

Pieter Abbeel, Adam Coates, Timothy Hunter, and Andrew Y Ng. 2009. Autonomous autorotation of an RC helicopter. In *Experimental Robotics*. 98

Jacob Andreas, Andreas Vlachos, and Stephen Clark. 2013. Semantic Parsing as Machine Translation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics* (*ACL*), Sofia, Bulgaria. 36, 45

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly Supervised Learning of Semantic Parsers for Mapping Instructions to Actions. *Transactions of the Association for Computational Linguistics* (*TACL*), 1(1). 123

Fei Arumae, Kristjanand Liu. 2018. Reinforced extractive summarization with question-focused rewards. In *Proceedings of ACL 2018, Student Research Workshop*, Melbourne, Australia. 82

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations* (*ICLR*), San Diego, California, USA. xv, 19, 20, 21, 56

Sergio Barrachina, Oliver Bender, Francisco Casacuberta, Jorge Civera, Elsa Cubel, Shahram Khadivi, Antonio Lagarda, Hermann Ney, Jesús Tomás, Enrique Vidal, and Juan-Miguel Vilar. 2008. Statistical Approaches to Computer-assisted Translation. *Computational Linguistics*, 35(1). 67

H. Bastani, M. Bayati, and K. Khosravi. 2017. Exploiting the Natural Exploration In Contextual Bandits. *ArXiv e-prints*, 1704.09011. 118

Johnathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic Parsing on Freebase from Question-Answer Pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* (*EMNLP*), Seattle, Washington, USA. 4, 25, 38, 67, 81

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*, 1st edition. O'Reilly Media, Inc. 48

Léon Bottou, Jonas Peters, Joaquin Quiñonero-Candela, Denis X. Charles, D. Max Chickering, Elon Portugaly, Dipanakar Ray, Patrice Simard, and Ed Snelson. 2013. Counterfactual Reasoning and Learning Systems: The Example of Computational Advertising. *Journal of Machine Learning Research*, 14. 6, 98

Johan Boye, Morgan Fredriksson, Jana Götze, and Jürgen Königsmann. 2014. A demonstration of a natural-language pedestrian routing system. In *Proceedings of the 5th international workshop on spoken dialogue systems* (*IWSDS*), Napa, California, USA. 26

Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2). 17

Sébastian Bubeck and Nicolò Cesa-Bianchi. 2012. Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems. *Foundations and Trends in Machine Learning*, 5(1). 14, 97

Qingqing Cai and Alexander Yates. 2013. Large-scale Semantic Parsing via Schema Matching and Lexicon Extension. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics* (*ACL*), Sofia, Bulgaria. 2, 26, 38, 68

Olivier Chapelle and Lihong Li. 2011. An Empirical Evaluation of Thompson Sampling. In *Advances in Neural Information Processing Systems* (*NIPS*). Curran Associates, Inc. 117

David Chiang. 2005. A Hierarchical Phrase-Based Model for Statistical Machine Translation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics* (*ACL*), Ann Arbor, Michigan, USA. 17

Kyunghyun Cho. 2015. Natural Language Understanding with Distributed Representation. *ArXiv e-prints*, 1511.07916. 19

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (*EMNLP*), Doha, Qatar. 19, 56

Eunsol Choi, Daniel Hewlett, Jakob Uszkoreit, Illia Polosukhin, Alexandre Lacoste, and Jonathan Berant. 2017. Coarse-to-Fine Question Answering for Long Documents. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics* (*ACL*), Vancouver, Canada. 82

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *ArXiv e-prints*, 1412.3555. 19

Jonathan Clark, Chris Dyer, Alon Lavie, and Noah Smith. 2011. Better Hypothesis Testing for Statistical Machine Translation: Controlling for Optimizer Instability. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies* (*ACL-HLT*), Portland, Oregon, USA. 49

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving Semantic Parsing from the World's Response. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, Uppsala, Sweden. 4, 67, 81

Michael Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Philadelphia, Pennsylvania, USA. 67, 69

Ronan Collobert, Fabian Sinz, Jason Weston, and Léon Bottou. 2006. Trading Convexity for Scalability. In *Proceedings of the 23rd International Conference on Machine Learning*, New York, New York, USA. 3, 66, 82

Koby Crammer and Yoram Singer. 2003. Ultraconservative Online Algorithms for Multiclass Problems. *Journal of Machine Learning Research*, 3. 18, 49

Deborah Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christing Pao, et al. 1995. ATIS3 Test Data LDC95S26. Web Download. Philadelphia: Linguistic Data Consortium. 45

Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. 2014. SAGA: A Fast Incremental Gradient Method with Support for Non-Strongly Convex Composite Objectives. In *Advances in Neural Information Processing Systems (NIPS)*, Montreal, Canada. 103

Chuong B. Do, Quoc Le, Choon Hui Teo, Olivier Chapelle, and Alexander J Smola. 2008. Tighter Bounds for Structured Estimation. In *Advances in Neural Information Processing Systems (NIPS)*, Vancouver, Canada. 66

Li Dong and Mirella Lapata. 2016. Language to Logical Form with Neural Attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, Berlin, Germany. 46, 123

Miroslav Dudik, John Langford, and Lihong Li. 2011. Doubly Robust Policy Evaluation and Learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, Bellevue, Washington, USA. 6, 98, 99, 106

Matthew Dunn, Levent Sagun, Mike Higgins, V. Ugur Guney, Volkan Cirik, and Kyunghyun Cho. 2017a. SearchQA: A New Q&A Dataset Augmented with Context from a Search Engine. *ArXiv e-prints*, 1704.05179. 25

Matthew Dunn, Levent Sagun, Mike Higgins, V. Ugur Güney, Volkan Cirik, and Kyunghyun Cho. 2017b. SearchQA: A New Q&A Dataset Augmented with Context from a Search Engine. *ArXiv e-prints*, 1704.05179. 38

Chris Dyer, Adam Lopez, Juri Ganitkevitch, Johnathan Weese, Ferhan Ture, Phil Blunsom, Hendra Setiawan, Vladimir Eidelman, and Philip Resnik. 2010. cdec: A Decoder, Alignment, and Learning Framework for Finite-State and Context-Free Translation Models. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, Uppsala, Sweden. 2, 16, 18, 45, 47, 74, 100, 112, 113

Sergey Edunov, Myle Ott, Michael Auli, David Grangier, and Marc'Aurelio Ranzato. 2018. Classical Structured Prediction Losses for Sequence to Sequence Learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (*HLT-NAACL*), New Orleans, Louisiana, USA. 82

Rudolf Flesch. 1974. *The Art of Readable Writing: With the Flesch Readability Formula*. Harper & Row. 38, 39

Michael C. Fu. 2006. Gradient Estimation. *Handbook in Operations Research and Management Science*, 13. 14, 107

Javier García and Fernando Fernández. 2015. A Comprehensive Survey on Safe Reinforcement Learning. *Journal of Machine Learning Research*, 16(1). 98

Kevin Gimpel and Noah A. Smith. 2012. Structured Ramp Loss Minimization for Machine Translation. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (*HLT-NAACL*), Montreal, Canada. xxii, 3, 4, 18, 67, 69, 73, 74, 82, 86

Dan Goldwasser and Dan Roth. 2013. Learning from Natural Instructions. *Machine Learning*, 94(2). 4, 67, 70, 123

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. Deep Learning. Book in preparation for MIT Press. 19

Peter J. Green. 1990. On the Use of the EM Algorithm for Penalized Likelihood Estimation. *Journal of the Royal Statistical Society, Series B* (*Methodological*), 52(3). 125

Evan Greensmith, Peter L. Bartlett, and Jonathan Baxter. 2004. Variance Reduction Techniques for Gradient Estimation in Reinforcment Learning. *Journal of Machine Learning Research*, 5. 103

Kelvin Guu, Panupong Pasupat, Evan Liu, and Percy Liang. 2017. From Language to Programs: Bridging Reinforcement Learning and Maximum Marginal Likelihood. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics* (*ACL*), Vancouver, Canada. 4, 46, 81, 82

Carolin Haas. 2014. Grounding Machine Translation in Semantic Parsing, Master's Thesis, Heidelberg, Germany. 68

Carolin Haas and Stefan Riezler. 2015. Response-based Learning for Machine Translation of Open-domain Database Queries. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (*HLT-NAACL*), Denver, Colorado, USA. 68

Carolin Haas and Stefan Riezler. 2016. A Corpus and Semantic Parser for Multilingual Natural Language Querying of OpenStreetMap. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (*HLT-NAACL*), San Diego, California, USA. 8, 9, 27, 47, 67, 68

Moritz Hardt, Ben Recht, and Yoram Singer. 2016. Train faster, generalize better: Stability of stochastic gradient descent. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, New York, New York, USA. 103

Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. 2013. Scalable Modified Kneser-Ney Language Model Estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL)*, Sofia, Bulgaria. 113

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9(8). 19

Matthew Hoffman. 2015. Variance Reduction Techniques for Stochastic Optimization. 104, 105, 106

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a Neural Semantic Parser from User Feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, Vancouver, Canada. 123

Mohit Iyyer, Jordan Boyd-Graber, Leonardo Claudino, Richard Socher, and Hal Daumé III. 2014. A Neural Network for Factoid Question Answering over Paragraphs. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar. 25, 38

Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 2017. Search-based Neural Structured Learning for Sequential Question Answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, Vancouver, Canada. 4, 82

Laura Jehl, Carolin Lawrence, and Stefan Riezler. 2019. Learning Neural Sequence-to-Sequence Models from Weak Feedback with Bipolar Ramp Loss. To appear in the Transactions of the Association for Computational Linguistics (TACL). 9, 83

Robin Jia and Percy Liang. 2016. Data Recombination for Neural Semantic Parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, Berlin, Germany. 46, 99, 123

Nan Jiang and Lihong Li. 2016. Doubly Robust Off-policy Value Evaluation for Reinforcement Learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, New York, New York, USA. 98, 104

Thorsten Joachims, Adith Swaminathan, and Maarten de Rijke. 2018. Deep learning with logged bandit feedback. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, Stockholm, Sweden. 98, 122

Rie Johnson and Tong Zhang. 2013. Accelerating Stochastic Gradient Descent using Predictive Variance Reduction. In *Advances in Neural Information Processing Systems (NIPS)*, Lake Tahoe, California, USA. 103

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, Vancouver, Canada. 38

Rohit J. Kate, Yuk Wah Wong, and Raymond J. Mooney. 2005. Learning to transform natural to formal languages. In *Proceedings of AAAI*, Pittsburgh, Pennsylvania, USA. 26

Dan Klein and Christopher D. Manning. 2003. Accurate Unlexicalized Parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*, Barcelona, Spain. 39

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for M-gram language modeling. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Detroit, Michigan, USA. 17

Philipp Koehn. 2005. Europarl: A Parallel Corpus for Statistical Machine Translation. In *Proceedings of the Machine Translation Summit*, Phuket, Thailand. 112

Philipp Koehn. 2010. *Statistical Machine Translation*. Cambridge University Press. 16, 48

Philipp Koehn and Barry Haddow. 2009. Interactive assistance to human translators using statistical machine translation methods. In *Proceedings of the Machine Translation Summit*. 67

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Birch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open Source Toolkit for Statistical Machine Translation. In *Proceedings of the ACL 2007 Demo and Poster Sessions*, Prague, Czech Republic. 45

Philipp Koehn and Josh Schroeder. 2007. Experiments in Domain Adaptation for Statistical Machine Translation. In *Proceedings of the Workshop on Machine Translation (WMT)*, Prague, Czech Republic. 113

Augustine Kong. 1992. A Note on Importance Sampling using Standardized Weights. Technical Report 348, Department of Statistics, University of Chicago, Illinois. 99, 103

Julia Kreutzer, Artem Sokolov, and Stefan Riezler. 2017. Bandit Structured Prediction for Neural Sequence-to-Sequence Learning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, Vancouver, Canada. 97

Nate Kushman and Regina Barzilay. 2013. Using Semantic Unification to Generate Regular Expressions from Natural Language. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (HLT-NAACL)*, Atlanta, Georgia, USA. 38

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. 2013. Scaling Semantic Parsers with On-the-Fly Ontology Matching. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Seattle, Washington, USA. 67

John Langford, Alexander Strehl, and Jennifer Wortman. 2008. Exploration Scavenging. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, Helsinki, Finland. 98, 99, 118

Carolin Lawrence, Pratik Gajane, and Stefan Riezler. 2017a. Counterfactual Learning for Machine Translation: Degeneracies and Solutions. In *Proceedings of the NIPS WhatIf Workshop*, Long Beach, California, USA. 9, 101

Carolin Lawrence and Stefan Riezler. 2016. NLmaps: A Natural Language Interface to Query OpenStreetMap. In *Proceedings of the 26th International Conference on Computational Linguistics: System Demonstrations* (*COLING*), Osaka, Japan. 8, 47

Carolin Lawrence and Stefan Riezler. 2018. Improving a Neural Semantic Parser by Counterfactual Learning from Human Bandit Feedback. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics* (*ACL*), Melbourne, Australia. 8, 9, 27, 47, 87, 124

Carolin Lawrence, Artem Sokolov, and Stefan Riezler. 2017b. Counterfactual Learning from Bandit Feedback under Deterministic Logging : A Case Study in Statistical Machine Translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* (*EMNLP*), Copenhagen, Denmark. 9, 101

Lihong Li, Shunbao Chen, Jim Kleban, and Ankur Gupta. 2015. Counterfactual Estimation and Optimization of Click Metrics in Search Engines: A Case Study. In *Proceedings of the International World Wide Web Conference* (*WWW*), Florence, Italy. 98

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. 2017. Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics* (*ACL*), Vancouver, Canada. 4, 46, 81, 89

Percy Liang, Alexandre Bouchard-Côté, Dan Klein, and Ben Taskar. 2006. An End-to-end Discriminative Approach to Machine Translation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics* (*ACL*), Sydney, Australia. 69

François Mairesse, Milica Gasic, Filip Jurcícek, Simon Keizer, Blaise Thomson, Kai Yu, and Steve J. Young. 2009. Spoken language understanding from unaligned data using discriminative classification models. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing,* (*ICASSP*), Taipei, Taiwan. 38

David A. McAllester and Joseph Keshet. 2011. Generalization Bounds and Consistency for Latent Structural Probit and Ramp Loss. In *Advances in Neural Information Processing Systems* (*NIPS*), Granada, Spain. 66, 69

Michael Minock and Johan Mollevik. 2013. Context-dependent 'near' and 'far' in spatial databases via supervaluation. *Data & Knowledge Engineering*, 86. 33

Dipendra Misra, Ming-Wei Chang, Xiaodong He, and Wen-tau Yih. 2018. Policy Shaping and Generalized Update Equations for Semantic Parsing from Denotations. *ArXiv e-prints*, 1809.01299. 4, 82, 89

Lili Mou, Zhengdong Lu, Hang Li, and Zhi Jin. 2017. Coupling Distributed and Symbolic Execution for Natural Language Queries. In *Proceedings of the 34th International Conference on Machine Learning* (*ICML*), Sydney, Australia. 4, 46, 81

Preslav Nakov, Francisco Guzmán, and Stephan Vogel. 2012. Optimizing for Sentence-Level BLEU+1 Yields Short Translations. In *Proceedings of the 24th International Conference on Computational Linguistics* (*COLING*), Bombay, India. 18, 69, 73, 112

Arvind Neelakantan, Quoc V. Le, Martín Abadi, Andrew McCallum, and Dario Amodei. 2017. Learning a Natural Language Interface with Neural Programmer. In *International Conference on Learning Representations* (*ICLR*), Toulon, France. 46, 123

Vassilina Nikoulina, Bogomil Kovachev, Nikolaos Lagos, and Christof Monz. 2012. Adaptation of Statistical Machine Translation Model for Cross-lingual Information Retrieval in a Service Context. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics* (*EACL*), Avignon, France. 67

Eric W. Noreen. 1989. *Computer Intensive Methods for Testing Hypotheses: An Introduction*. Wiley, New York. 49, 74, 88, 115, 130

Franz Josef Och. 2003. Minimum Error Rate Training in Statistical Machine Translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (*HLT-NAACL*), Edmonton, Cananda. 18, 48, 114

Franz Josef Och and Hermann Ney. 2002. Discriminative Training and Maximum Entropy Models for Statistical Machine Translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics* (*ACL*), Philadelphia, Pennsylvania, USA. 69

Franz Josef Och and Hermann Ney. 2003. A Systematic Comparison of Various Statistical Alignment Models. *Computational Linguistics*, 29(1). 48

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics* (*ACL*), Philadelphia, Pennsylvania, USA. 18, 74, 113

Panupong Pasupat and Percy Liang. 2015. Compositional Semantic Parsing on Semi-Structured Tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* (*ACL-IJCNLP*), Beijing, China. 4, 38, 81

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830. 113

Doina Precup, Richard S. Sutton, and Satinder P. Singh. 2000. Eligibility Traces for Off-Policy Policy Evaluation. In *Proceedings of the 17th International Conference on Machine Learning*(*ICML*), San Francisco, California, USA. 98, 99, 104

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (*EMNLP*). 4, 81

Rajesh Ranganath, Sean Gerrish, and David M. Blei. 2014. Black Box Variational Inference. In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics* (*AISTATS*), Reykjavik, Iceland. 103

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2016. Sequence Level Training with Recurrent Neural Networks. In *International Conference on Learning Representations* (*ICLR*), San Juan, Puerto Rico. 82

Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jarret Ross, and Vaibhava Goel. 2017. Self-critical sequence training for image captioning. In *Conference on Computer Vision and Pattern Recognition* (*CVPR*). 82

Stefan Riezler and John Maxwell. 2005. On Some Pitfalls in Automatic Evaluation and Significance Testing for MT. In *Proceedings of the ACL 2005 Workshop on Intrinsic and Extrinsic Evaluation Measures for MT and/or Summarization*, Ann Arbor, Michigan, USA. 115

Stefan Riezler, Patrick Simianer, and Carolin Haas. 2014. Response-Based Learning for Grounded Machine Translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics* (*ACL*), Baltimore, Maryland, USA. 67, 68

Paul R. Rosenbaum and Donald B. Rubin. 1983. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1). 99, 102

Sheldon M. Ross. 2013. *Simulation*, fifth edition. Elsevier. 15, 99, 104, 105

Avneesh Saluja, Ian Lane, and Ying Zhang. 2012. Machine Translation with Binary Feedback: a largemargin approach. In *Proceedings of the Tenth Biennial Conference of the Association for Machine Translation in the Americas* (*AMTA*), San Diego, California, USA. 67

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. 2015. Gradient Estimation Using Stochastic Computation Graphs. In *Advances in Neural Information Processing Systems* (*NIPS*), Montreal, Canada. 103

Rico Sennrich, Orhan Firat, Kyunghyun Cho, Alexandra Birch, Barry Haddow, Julian Hitschler, Marcin Junczys-Dowmunt, Samuel Läubli, Antonio Valerio Miceli Barone, Jozef Mokry, and Maria Nadejde. 2017. Nematus: a Toolkit for Neural Machine Translation. In *Proceedings of the Software Demonstrations of the 15th Conference of the European Chapter of the Association for Computational Linguistics* (*EACL*), Valencia, Spain. 2, 18, 19, 46, 47, 56, 83, 87, 129

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (*ACL*), Berlin, Germany. 16, 57

Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. 2016. Minimum Risk Training for Neural Machine Translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (*ACL*), Berlin, Germany. 82, 84

Patrick Simianer, Stefan Riezler, and Chris Dyer. 2012. Joint Feature Selection in Distributed Stochastic Learning for Large-Scale Discriminative Training in SMT. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics* (*ACL*), Jeju Island, South Korea. 49, 74

David A. Smith and Jason Eisner. 2006. Minimum Risk Annealing for Training Log-Linear Models. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, Sydney, Australia. 84

Jason Smith, Herve Saint-Amand, Magdalena Plamada, Philipp Koehn, Chris Callison-Burch, and Adam Lopez. 2013. Dirt Cheap Web-Scale Parallel Text from the Common Crawl. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics* (*ACL*), Sofia, Bulgaria. 74

Artem Sokolov, Julia Kreutzer, Christopher Lo, and Stefan Riezler. 2016. Stochastic Structured Prediction under Bandit Feedback. In *Advances in Neural Information Processing Systems* (*NIPS*), Barcelona, Spain. 97

Andreas Stolcke. 2002. SRILM - An Extensible Language Modeling Toolkit. 48

Alex Strehl, John Langford, Lihong Li, and Sham M. Kakade. 2010. Learning from Logged Implicit Exploration Data. In *Advances in Neural Information Processing Systems* (*NIPS*), Vancouver, Canada. 98, 99

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems* (*NIPS*), Montreal, Canada. 19, 20, 56

Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning. An Introduction*. The MIT Press. 97, 98

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning. An Introduction, Second Edition*. The MIT Press. 5, 8, 13, 14, 15

Adith Swaminathan and Thorsten Joachims. 2015a. Batch learning from logged bandit feedback through counterfactual risk minimization. *Journal of Machine Learning Research*, 16. 6, 98

Adith Swaminathan and Thorsten Joachims. 2015b. The Self-Normalized Estimator for Counterfactual Learning. In *Advances in Neural Information Processing Systems* (*NIPS*), Montreal, Canada. 98, 99, 103, 104, 110, 125

Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *ArXiv e-prints*, 1605.02688. 18

Philip Thomas and Emma Brunskill. 2016. Data-Efficient Off-Policy Policy Evaluation for Reinforcement Learning. In *Proceedings of the 33nd International Conference on Machine Learning* (*ICML*), New York, New York, USA. 6, 98, 99, 104

Philip S. Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. 2015. High Confidence Policy Improvement. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning* (*ICML*), Lille, France. 98

Jörg Tiedemann. 2012. Parallel Data, Tools and Interfaces in OPUS. In *Proceedings of the Eight International Conference on Language Resources and Evaluation* (*LREC*), Istanbul, Turkey. 113

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Modeling Coverage for Neural Machine Translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (*ACL*), Berlin, Germany. 60

Chong Wang, Xi Chen, Alexander J Smola, and Eric P Xing. 2013. Variance Reduction for Stochastic Gradient Optimization. In *Advances in Neural Information Processing Systems* (*NIPS*), Lake Tahoe, California, USA. 103

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a Semantic Parser Overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* (*ACL-IJCNLP*), Beijing, China. 38

Jason D. Williams and Steve Young. 2007. Partially observable Markov decision processes for spoken dialog systems. *Computer Speech Language*, 21(2). 38

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 20. 14

Yuk Wah Wong and Raymond J. Mooney. 2006. Learning for Semantic Parsing with Statistical Machine Translation. In *Proceedings of the 2006 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (*HLT-NAACL*), New York, New York, USA. 2, 26, 45, 68

Lijun Wu, Fei Tian, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. 2018. A Study of Reinforcement Learning for Neural Machine Translation. *ArXiv e-prints*, 1808.08866. 82

Yi Yang, Wen-tau Yih, and Christopher Meek. 2015. WikiQA: A Challenge Dataset for Open-Domain Question Answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (*EMNLP*), Lisbon, Portugal. 25, 38

Zhilin Yang, Junjie Hu, Ruslan Salakhutdinov, and William Cohen. 2017. Semi-Supervised QA with Generative Domain-Adaptive Nets. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics* (*ACL*), Vancouver, Canada. 82

Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. 2016. The Value of Semantic Parse Labeling for Knowledge Base Question Answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (*ACL*), Berlin, Germany. 26, 124

Matthew D. Zeiler. 2012. ADADELTA: An Adaptive Learning Rate Method. *ArXiv e-prints*, 1212.5701. 56, 115, 129

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of AAAI*, Portland, Oregon, USA. 36, 37, 45

Acknowledgments

I would like to thank my supervisor, Prof. Stefan Riezler, for his constant guidance, support and inspiration. He taught me the fundamentals of statistical methods which encouraged my interest in machine learning for Natural Language Processing and led me to pursue a doctoral degree on the subject. I am extremely grateful that he was always available to answer my questions and to help me identify the best next step. From the beginning, he encouraged my ideas and our discussions enabled me to find the path that led to this thesis. Thank you!

Additional thanks go to Prof. Artem Sokolov, who has lead me to the path of exploring counterfactual learning. I learnt a lot from him during our joint project and he was always willing to discuss any problems and potential solutions with me. Thank you for agreeing to be the second reviewer for this thesis!

Further, I am thankful to all my co-authors and colleagues. They were always there if I needed advice, someone to bounce my ideas back and forth with or just to talk. Thank you for providing a truly great work environment! Extra special thanks go to Laura Jehl and Julia Kreutzer who graciously read drafted chapters and provided me with valuable feedback.

I express my gratitude to my husband, Craig Lawrence, for being my rock of strength. He was always willing to discuss my ongoing projects with me and patiently proof-read drafts for this thesis. I am grateful to my parents, Claudia & Leo Haas, who have always encouraged me to be the best version of myself. It was their guidance that made me believe that I could do anything as long as I am determined enough.

Finally, I would like to thank the German research foundation (DFG) who funded my project with the grant RI-2221/2-1 "Grounding Statistical Machine Translation Machine Translation in Perception and Action".

Appendix

Appendix A

Context Free Grammar of NLmaps

$$
\begin{aligned}
[S] &\rightarrow [X, 1] \\
[S] &\rightarrow dist([X, 1], unit([KMMI, 2])) \\
[S] &\rightarrow dist([X, 1], [X, 2], unit([KMMI, 3])) \\
[S] &\rightarrow dist([X, 1]) \\
[S] &\rightarrow dist([X, 1], [X, 2]) \\
[S] &\rightarrow dist([X, 1], for([CW, 2])) \\
[S] &\rightarrow dist([X, 1], [X, 2], for([CW, 3])) \\
[X] &\rightarrow query(north([AROUND, 1]), [META, 2]) \\
[X] &\rightarrow query(north([QUERY, 1]), [META, 2]) \\
[X] &\rightarrow query(west([AROUND, 1]), [META, 2]) \\
[X] &\rightarrow query(west([QUERY, 1]), [META, 2]) \\
[X] &\rightarrow query(south([AROUND, 1]), [META, 2]) \\
[X] &\rightarrow query(south([QUERY, 1]), [META, 2]) \\
[X] &\rightarrow query(east([AROUND, 1]), [META, 2]) \\
[X] &\rightarrow query(east([QUERY, 1]), [META, 2]) \\
[X] &\rightarrow query([AROUND, 1], [META, 2]) \\
[X] &\rightarrow query([QUERY, 1], [META, 2]) \\
[AROUND] &\rightarrow around(center([QUERY, 1]), search([QUERY, 2]), \\
&\qquad maxdist([DIST, 3])) \\
[AROUND] &\rightarrow around(center([QUERY, 1]), search([QUERY, 2]), \\
&\qquad maxdist([DIST, 3]), [META\_TOPX, 4]) \\
[QUERY] &\rightarrow [AREA, 1], [OSM, 2] \\
[QUERY] &\rightarrow [OSM, 1] \\
[META] &\rightarrow qtype([META\_REQ, 1]) \\
[META] &\rightarrow qtype([META\_POS, 1]) \\
[META\_REQ] &\rightarrow [META\_REQ, 1], [META\_REQ, 2]
\end{aligned}
$$

$$[META\_REQ] \rightarrow [META\_REQ, 1], [META_POS, 2]$$
$$[META\_REQ] \rightarrow findkey(and([KEY, 1], [KEY, 2]))$$
$$[META\_REQ] \rightarrow findkey([KEY, 1])$$
$$[META\_REQ] \rightarrow findkey([KEY, 1], [META\_TOPX, 2])$$
$$[META\_REQ] \rightarrow count$$
$$[META\_REQ] \rightarrow latlong$$
$$[META\_REQ] \rightarrow latlong([META\_TOPX, 1])$$
$$[META\_REQ] \rightarrow least([META\_TOPX, 1])$$
$$[META\_POS] \rightarrow nodup([META\_REQ, 1])$$
$$[META\_TOPX] \rightarrow topx([INT, 1])$$
$$[AREA] \rightarrow area([INNER, 1])$$
$$[OSM] \rightarrow nwr([INNER, 1])$$
$$[OSM] \rightarrow nwr([INNER, 1]), [OSM, 2]$$
$$[INNER] \rightarrow and([INNER, 1], [INNER, 2])$$
$$[INNER] \rightarrow or([INNER, 1], [INNER, 2])$$
$$[INNER] \rightarrow keyval([KEY, 1], [VAL, 2]), [INNER, 3]$$
$$[INNER] \rightarrow keyval([KEY, 1], [VAL, 2])$$
$$[CW] \rightarrow car$$
$$[CW] \rightarrow walk$$
$$[KMMI] \rightarrow km$$
$$[KMMI] \rightarrow mi$$
$$[DIST] \rightarrow WALKDING\_DIST$$
$$[DIST] \rightarrow DIST\_INTOWN$$
$$[DIST] \rightarrow DIST\_OUTTOWN$$
$$[DIST] \rightarrow DIST\_DAYTRIP$$
$$[DIST] \rightarrow [INT, 1]$$
$$[VAL] \rightarrow or([VAL, 1], [VAL, 2])$$
$$[VAL] \rightarrow and([VAL, 1], [VAL, 2])$$
$$[VAL] \rightarrow \text{'valuevariable'}$$
$$[VAL] \rightarrow \text{'keyvariable'}$$
$$[KEY] \rightarrow \{\text{set of OSM key tags}\}$$
$$[INT] \rightarrow \{\text{set of all digits}\}\ [INT, 1]$$
$$[INT] \rightarrow \{\text{set of all digits}\}$$

Appendix B

Detailed Gradient Derivations

IPS

$$\mathcal{V}_{IPS}(\pi) = \frac{1}{n}\sum_{t=1}^{n} \delta_t \cdot \frac{\pi(y_t|x_t)}{\mu(y_t|x_t)}$$

Gradient

Based on the score function gradient estimator (see Equation 2.10, Section 2.2) and using $\nabla_w \log f = \frac{\nabla_w f}{f}$:

$$\nabla_w \mathcal{V}_{IPS}(\pi) = \nabla_w \left[\frac{1}{n}\sum_{t=1}^{n} \delta_t \cdot \frac{\pi(y_t|x_t)}{\mu(y_t|x_t)}\right]$$

$$= \frac{1}{n}\sum_{t=1}^{n} \delta_t \cdot \frac{1}{\mu(y_t|x_t)} \nabla_w \pi(y_t|x_t)$$

$$= \frac{1}{n}\sum_{t=1}^{n} \delta_t \cdot \frac{1}{\mu(y_t|x_t)} \cdot \frac{\pi(y_t|x_t)}{\pi(y_t|x_t)} \cdot \nabla_w \pi(y_t|x_t)$$

$$= \frac{1}{n}\sum_{t=1}^{n} \delta_t \cdot \frac{\pi(y_t|x_t)}{\mu(y_t|x_t)} \cdot \nabla_w \log(\pi(y_t|x_t))$$

Setting all $\mu(y_t|x_t) = 1$ recovers the gradient of DPM.

IPS+R

$$\mathcal{V}_{IPS+R}(\pi) = \sum_{t=1}^{n} \delta_t \frac{\rho_t}{\sum_{u=1}^{n} \rho_u} = \sum_{t=1}^{n} \delta_t \cdot \bar{\rho}_t,$$

$$\text{with } \bar{\rho}_t = \frac{\rho_t}{\sum_{u=1}^{n} \rho_u} \text{ and } \rho_t = \frac{\pi(y_t|x_t)}{\mu(y_t|x_t)}.$$

Gradient

Using the gradient of IPS,

$\nabla_w \rho_t = \rho_t' = \frac{\pi(y_t|x_t)}{\mu(y_t|x_t)} \cdot \nabla_w \log(\pi(y_t|x_t)) = \rho_t \cdot \nabla_w \log(\pi(y_t|x_t))$,

and the quotient rule, $\nabla_w \frac{f(w)}{g(w)} = \frac{g(w)\nabla_w f(w) - f(w)\nabla_w g(w)}{g(w)^2}$:

$$\nabla_w \mathcal{V}_{IPS+R}(\pi) = \sum_{t=1}^{n} \left[\delta_t \frac{\rho_t' \cdot \sum_{u=1}^{n} \rho_u - \rho_t \cdot \sum_{u=1}^{n} \rho_u'}{(\sum_{u=1}^{n} \rho_u)^2} \right]$$

$$= \sum_{t=1}^{n} \left[\delta_t \frac{\rho_t \cdot \nabla_w \log(\pi(y_t|x_t)) \cdot \sum_{u=1}^{n} \rho_u - \rho_t \cdot [\sum_{u=1}^{n} \rho_u \cdot \nabla_w \log(\pi(y_u|x_u))}{(\sum_{u=1}^{n} \rho_u)^2} \right]$$

$$= \sum_{t=1}^{n} \left[\delta_t \cdot \bar{\rho}_t \left(\nabla_w \log(\pi(y_t|x_t)) - \sum_{u=1}^{n} \bar{\rho}_u \cdot \nabla_w \log(\pi(y_u|x_u)) \right) \right]$$

Setting all $\mu(y_t|x_t) = 1$ recovers the gradient of DPM+R.

$\hat{c}$ DR

$$\mathcal{V}_{DR}(\pi) = \sum_{t=1}^{n} \left[(\delta_t - \hat{c}\hat{\delta}(x_t, y_t)) \cdot \bar{\rho}_t + \hat{c} \sum_{y \in \mathbf{Y}(x_t)} \hat{\delta}(x_t, y) \cdot \pi(y|x_t) \right]$$

Gradient

Using the gradient of IPS+R:

$$\nabla_w \mathcal{V}_{DR+R}(\pi) = \sum_{t=1}^{n} \left[(\delta_t - \hat{c}\hat{\delta}(x_t, y_t)) \cdot \bar{\rho}_t \left(\nabla_w \log(\pi(y_t|x_t)) - \sum_{u=1}^{n} \bar{\rho}_u \cdot \nabla_w \log(\pi(y_u|x_u)) \right) + \hat{c} \sum_{y \in \mathbf{Y}(x_t)} \hat{\delta}(x_t, y) \cdot \pi(y|x_t) \nabla_w \log(\pi(y|x_t)) \right]$$

Setting all $\mu(y_t|x_t) = 1$ recovers the gradient of $\hat{c}$ DC.
Setting $\hat{c} = 1$ recovers DR.
Setting all $\mu(y_t|x_t) = 1$ and $\hat{c} = 1$ recovers DC.

DPM+OSL

$$\begin{aligned}\mathcal{V}_{\text{DPM+OSL}}(\pi_w) &= \frac{1}{m}\sum_{t=1}^{m}\delta_t\bar{\pi}_{w,w'}(y_t|x_t),\\ &= \frac{\frac{1}{m}\sum_{t=1}^{m}\delta_t\pi_w(y_t|x_t)}{\frac{1}{n}\sum_{t=1}^{n}\pi_{w'}(y_t|x_t)}.\end{aligned}$$

Gradient

Based on the score function gradient estimator (see Equation 2.10, Section 2.2) and using $\nabla_w \log f = \frac{\nabla_w f}{f}$:

$$\begin{aligned}\nabla_w\mathcal{V}_{\text{DPM+OSL}} &= \frac{\frac{1}{m}\sum_{t=1}^{m}\delta_t\nabla_w\pi_w(y_t|x_t)}{\frac{1}{n}\sum_{t=1}^{n}\pi_{w'}(y_t|x_t)}\\ &= \frac{\frac{1}{m}\sum_{t=1}^{m}\delta_t\pi_w(y_t|x_t)\nabla_w\log(\pi_w(y_t|x_t))}{\frac{1}{n}\sum_{t=1}^{n}\pi_{w'}(y_t|x_t)}\\ &= \frac{1}{m}\sum_{t=1}^{m}\delta_t\bar{\pi}_{w,w'}(y_t|x_t)\nabla_w\log(\pi_w(y_t|x_t))\end{aligned}$$

DPM+T

$$\mathcal{V}_{\text{DPM+T}}(\pi_w) = \frac{1}{n}\sum_{t=1}^{n}\left(\sum_{j=1}^{|y_t|}\delta_{t,j}\log\pi_w(y_{t,j}|y_{t,<j},x_t)\right)$$

Gradient

$$\nabla_w\mathcal{V}_{\text{DPM+T}} = \frac{1}{n}\sum_{t=1}^{n}\left(\sum_{j=1}^{|y_t|}\delta_{t,j}\nabla_w\log\pi_w(y_{t,j}|y_{t,<j},x_t)\right)$$

DPM+T+OSL

$$\mathcal{V}_{\text{DPM+T+OSL}}(\pi_w) = \frac{\frac{1}{m}\sum_{t=1}^{m}\left(\sum_{j=1}^{|y_t|}\delta_{t,j}\log\pi_w(y_{t,j}|y_{t,<j},x_t)\right)}{\frac{1}{n}\sum_{t=1}^{n}\pi_{w'}(y_t|x_t)}$$

Gradient

$$\nabla_w\mathcal{V}_{\text{DPM+T+OSL}} = \frac{\frac{1}{m}\sum_{t=1}^{m}\left(\sum_{j=1}^{|y_t|}\delta_{t,j}\nabla_w\log\pi_w(y_{t,j}|y_{t,<j},x_t)\right)}{\frac{1}{n}\sum_{t=1}^{n}\pi_{w'}(y_t|x_t)}$$

Index