Neural Networks: Architectures and Applications for NLP

Session 03

Julia Kreutzer

15. November 2016

Institut für Computerlinguistik, Heidelberg

Overview

1. Recap
2. Metaparameter
3. In der Praxis
4. NLP (almost) from Scratch
5. Ausblick

Recap

Trainingsprozess

Festlegen: $NN(x)$, Loss $L(\hat{y}, y)$, Lernrate η, Gradient-Descent-Variante
Für jedes Input-Output Paar (Singleton oder Batch):

1. Forward-Pass:
    - Output der Units
    - Output des Netzwerks
    - Berechne Loss
2. Backward-Pass:
    - Berechne Deltas
    - Berechne Gradienten
3. Update:
    - Berechne Gewichtsupdates

Metaparameter

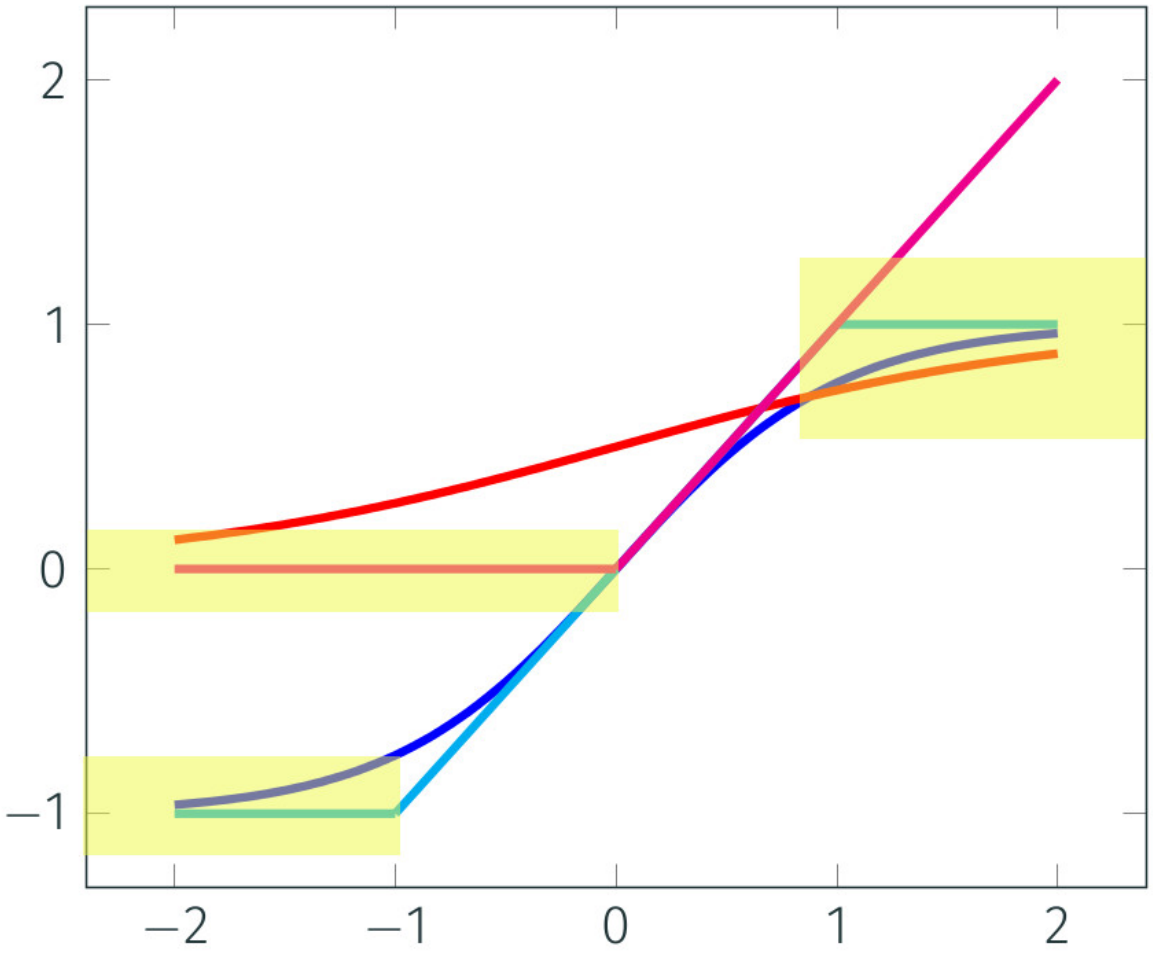

DECISIONS
DECISIONS EVERYWHERE
quickmeme.com

Metaparameter

Entscheidungen:

- Initialisierung
- Welche Loss-Funktion?
- Welche Lernraten?
- Wie viele Units pro Layer, wie viele Layer?
- Wie viele Trainingsamples pro Batch?
- Wie lange trainieren?

Aktivierungsfunktionen



tanh
hardtanh
sigmoid
ReLU

Sättigung i

Problematisch: wenn die Ausgabe der Aktivierungsfunktion dieselbe für alle Inputs ist und der Gradient (nah an) 0.

- `tanh`, `sigmoid`-Units können Sättigung erreichen, d.h. Outputs sind immer nah an 1
- `ReLU`s können “sterben”: wenn die Inputwerte immer negativ sind, z.B. durch starken negativen Bias, d.h. Outputs sind immer 0

$\rightarrow$ Anzahl und Größe der Inputs sind entscheidend für den Lernprozess.

Sättigung ii

Lösungen:

- Input-Range skalieren
- geschickte Initialisierung
- Lernrate anpassen, z.B. für **ReLU**s wenn nach einem großen Update alle Inputs negativ sind
- Schichten normalisieren, z.B. $g(h) = \frac{tanh(h)}{||tanh(h)||}$, aufwendig!

Initialisierung i

Ausgangswerte der Gewichte bestimmen, zu welchem lokalen Minimum der Lernprozess konvergiert [Sutskever et al., 2013]

Initialisierung der Gewichte:

- von **Null**
- **Uniformverteilung** [LeCun et al., 2012]
  Werte zufällig nach Uniformverteilung mit Erwartungswert 0 und Standardabweichung $\sigma_w = m^{-\frac{1}{2}}$ bestimmen, wobei m der "fan-in" ist, d.h. die Anzahl von Inputs an diesem Neuron
- **Sparse Initialization** [Martens, 2010]
  Jedes Neuron wird nur mit k zufällig bestimmten Neuronen der unteren Schicht verbunden.

Initialisierung ii

- **Glorot / Xavier** [Glorot and Bengio, 2010]
  $\mathbf{W} \sim \mathcal{U}(-\frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}, +\frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}})$
  mit **ReLU**: $\mathbf{W} \sim \mathcal{N}(0; \sqrt{\frac{2}{d_{in}}})$ [He et al., 2015]
- **Orthogonal / Einheitsmatrix**
  für 2D quadratische Matrizen, v.a. für RNNs. Eigenwerte sind 1, d.h. Gradienten explodieren/schwinden nicht.
- **Vor-trainiert**
  können Vorwissen / Bias repräsentieren, z.B. Word Embeddings

Loss-Funktionen i

Multi-Class Klassifikation

j ist die korrekte Klasse, $p = \arg\max_{i \neq j} \hat{y}_i$, die inkorrekte Klasse mit dem höchsten Score.

1. **Hinge Loss**: $\max(0, 1 - (\hat{y}_j - \hat{y}_p))$,
   keine probabilistische Interpretation, Margin ≥ 1.
2. **Log Loss**: $\log(1 + \exp(-(\hat{y}_j - \hat{y}_p)))$,
   "soft" Hinge mit unendlichem Margin
3. **Categorical Cross-Entropy**: $-\sum_i y_i \log(\hat{y}_i)$,
   probabilistische Interpretation der Klassenzuordnung.

Loss-Funktionen ii

Multi-Class Klassifikation: Beispiel

y	$\hat{y}$	Hinge-Loss	Log-Loss	Xent-Loss
[1 0 0]	[0.80 0.15 0.05]	0.35	0.42	0.22
	[0.35 0.30 0.35]	1	0.69	1.05
	[0.20 0.10 0.70]	1.5	0.97	1.61

Loss-Funktionen iii

Ranking

Ranking der Ausgabe für x (korrektes Input) über x' (inkorrektes Input). Gut geeignet für unüberwachte Verfahren, z.B. Erlernen von Wordrepräsentationen oder Informationsexktraktion.

1. **Margin Ranking**: $\max(0, 1 - (NN(x) - NN(x')))$
2. **Log Ranking**: $\log(1 + \exp(-NN(x) - NN(x')))$

Mehr Lossfunktionen z.B. hier: Liste der Lostfunktionen in Torch

Lernkurven

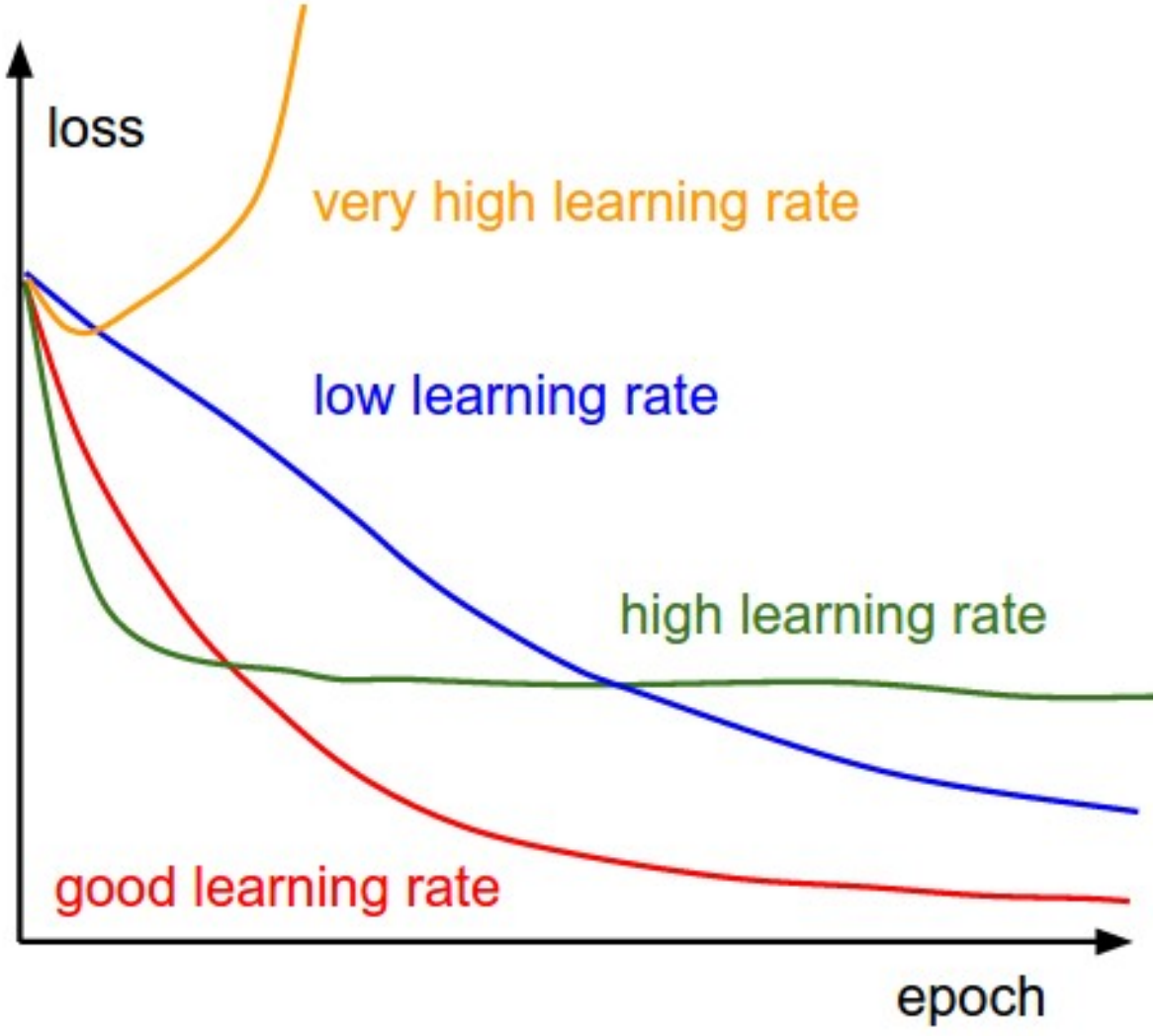


Abbildung 1: Quelle: CS231n

Lernraten i

Ideale Lernrate:
große Schritte in der Suchphase, kleine in der Konvergenzphase,
Lernkurve is “smooth”

- zu groß: keine Konvergenz
- zu klein: sehr langsame Konvergenz

Lernraten ii

Varianten

- konstant
- abnehmend
  - mit der Zeit t reduzieren, z.B. $\eta_t = \frac{\eta_0}{t+1}$
  - [Bottou, 2012] $\eta_t = \frac{\eta_0}{1+\eta_0 \lambda t}$, Metaparameter λ
  - Fehler beobachten: wenn er nicht mehr sinkt, Lernrate verringern

Momentum i

Momentum

- **Klassisches Momentum** [Polyak, 1964]
  Momentum-Koeffizient γ, Velocity-Vector v_t:
  $v_{t+1} = \gamma v_t - \eta \nabla f(\theta_t)$
  $\theta_{t+1} = \theta_t + v_{t+1}$
- **Nesterov's Accelerated Gradient** [Nesterov, 1983]
  $v_{t+1} = \gamma v_t - \eta \nabla f(\theta_t + \gamma v_t)$
  $\theta_{t+1} = \theta_t + v_{t+1}$

Momentum ii

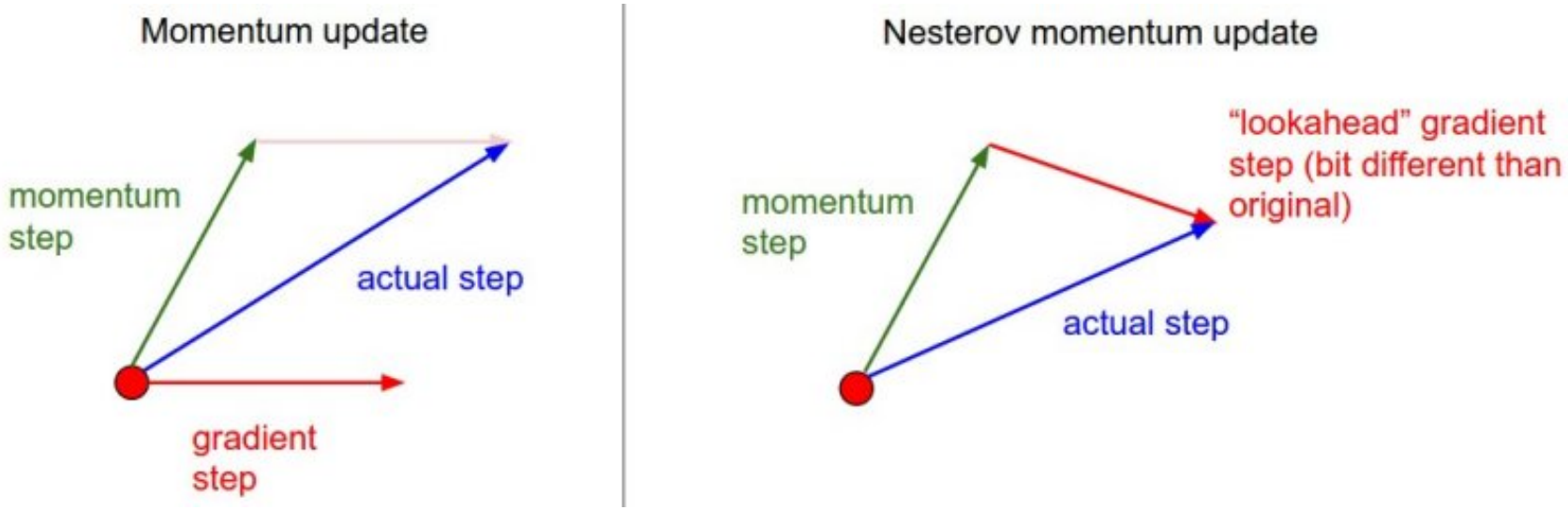


Abbildung 2: Momentum-Update, Quelle: CS231n

Beide führen zu stabilerem Training und können das Lernen beschleunigen, typischerweise mit kleineren Lernraten.

Adaptive Lernraten i

Idee: für jedes Gewicht eine individuelle Lernrate,
aktuelles Update $\Delta\theta_t$ hängt nicht nur vom Gradienten $g_t = \frac{\partial L}{\partial \theta_{t-1}}$ ab

1. **AdaGrad** [Duchi et al., 2011]

   $$\Delta\theta_t = -\frac{\eta}{\sqrt{\text{diag}(G_t) + \epsilon}} \odot g_t, \quad G_t = \sum_{t'=1}^{t} g_{t'} {g_{t'}}^\top$$

   G_t enthält die Summen der Quadrate aller Gradienten bis zum Zeitpunkt t

   Vorteil: frequente Updates werden gedämpft, infrequente gestärkt.
   Nachteil: G wächst monoton

Adaptive Lernraten ii

2. **RMSProp** [Tieleman and Hinton, 2012]

$$\Delta\theta_t = -\frac{\eta}{\sqrt{\mathbb{E}[g^2]_t + \epsilon}} g_t, \quad \mathbb{E}[g^2]_t = \gamma \mathbb{E}[g^2]_{t-1} + (1-\gamma) g_t^2$$

Vorteil: wie AdaGrad, nur mit moving average und exponential decay

3. **AdaDelta** [Zeiler, 2012]

$$\Delta\theta_t = -\frac{\sqrt{\mathbb{E}[\Delta\theta^2]_{t-1} + \epsilon}}{\sqrt{\mathbb{E}[g^2]_t + \epsilon}} g_t, \quad \mathbb{E}[\Delta\theta^2]_t = \gamma \mathbb{E}[\Delta\theta^2]_{t-1} + (1-\gamma) \Delta\theta_t^2$$

Vorteil: kein Parameter η

Adaptive Lernraten iii

4. **Adam** [Kingma and Ba, 2015]

$$\Delta\theta_t = -\frac{\eta}{\sqrt{\hat{v}_t + \epsilon}}\hat{m}_t,$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad m_t = \beta_1 v_{t-1} + (1-\beta_1)g_t$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}, \quad v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2$$

Vorteil: Momentum und Bias-Korrektur
Nachteil: 3 Metaparameter β_1, β_2, η

Over/Underfitting

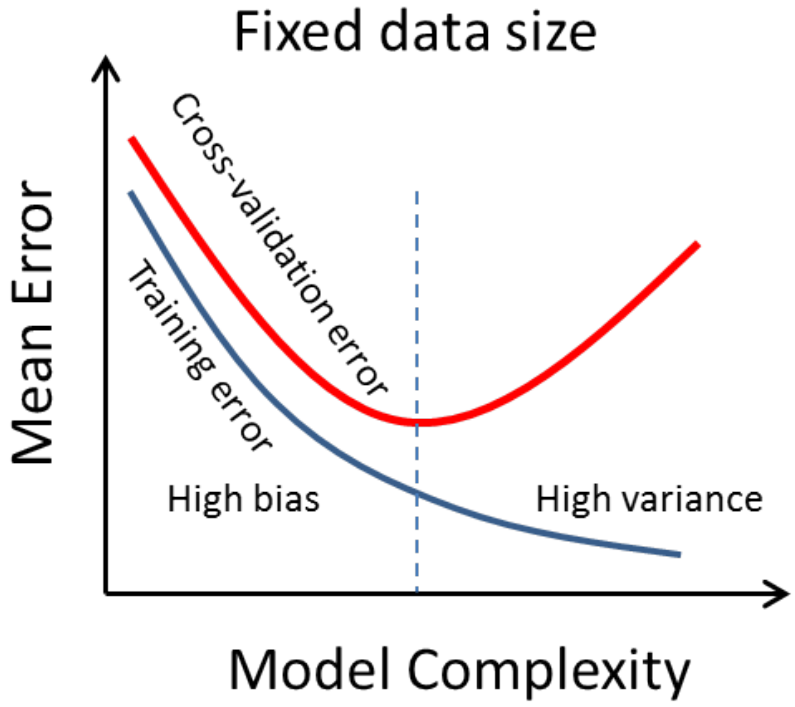
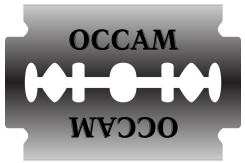


Abbildung 3: Under- vs. Overfitting, Bias vs. Variance, Quelle

Je größer das Netzwerk und je länger das Training, desto schneller kommt es zur Überanpassung an die Trainingsdaten.

Regularisierung der Gewichtsnorm



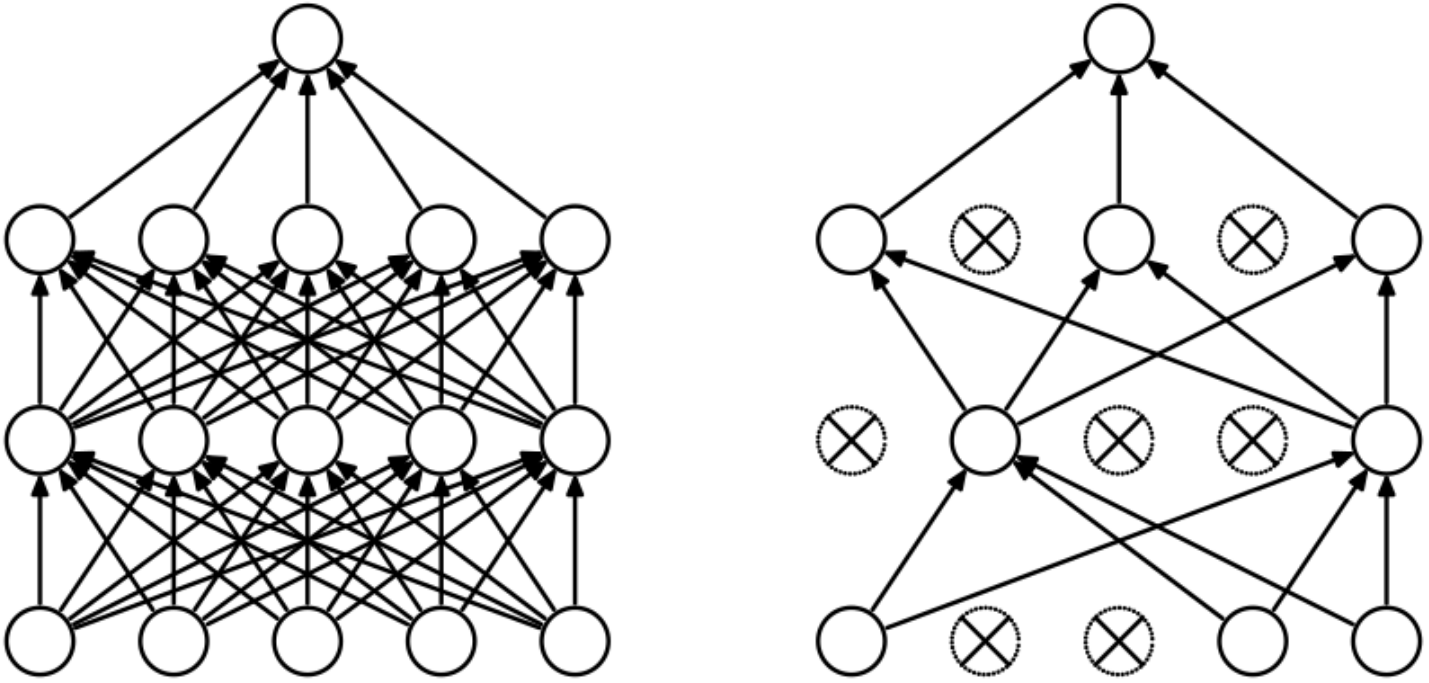
Ockhams Rasiermesser:
Bevorzuge das einfachste Modell

Bestrafe große Gewichtsnormen:

- l1: addiere $\lambda||\theta||$ zum Loss und minimiere
- l2: addiere $\frac{\lambda}{2}||\theta||^2$ zum Loss und minimiere

$\rightarrow$ Gewichte mit kleiner Norm werden bevorzugt

Dropout i



(a) Standard Neural Net

(b) After applying dropout.

Abbildung 4: NN mit Dropout-Units, Quelle: [Srivastava et al., 2014]

Dropout ii

Dropout verhindert, dass das Netzwerk sich nur auf einzelne Gewichte verlässt. [Srivastava et al., 2014]

Training: Jedes Neuron wird mit Wahrscheinlichkeit p ausgeschaltet (“drop out”, d.h. deren Gewichte zu null gesetzt).

Testing: Alle Neuronen werden benutzt, aber deren Output wird mit p skaliert.

$\rightarrow$ Test-Output entspricht dem Erwartungswert des Training-Outputs

Dropout iii

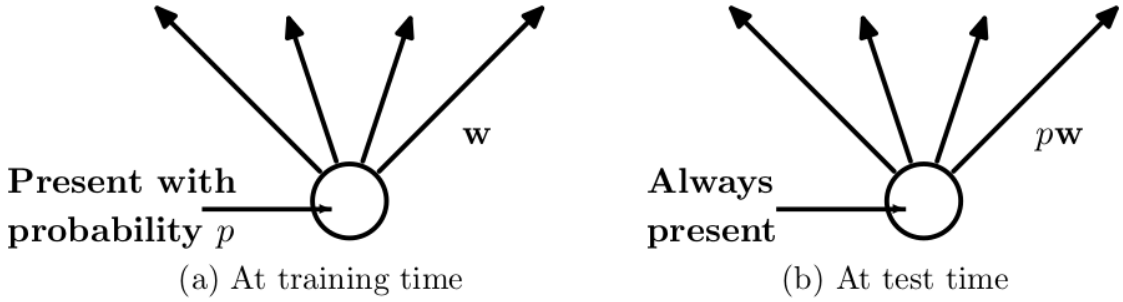


Abbildung 5: Dropout NN Training vs. Testing, Quelle: [Srivastava et al., 2014]

Implementierungsdetails: invertiertes Dropout

Masken: binäre Masken für jede Schicht bestimmen, welche Neuronen ausgeschaltet werden

Training: Dropout mit Wahrscheinlichkeit p UND Skalierung des Outputs mit p

Testing: Alle Neuronen werden normal benutzt.

Stoppkriterien

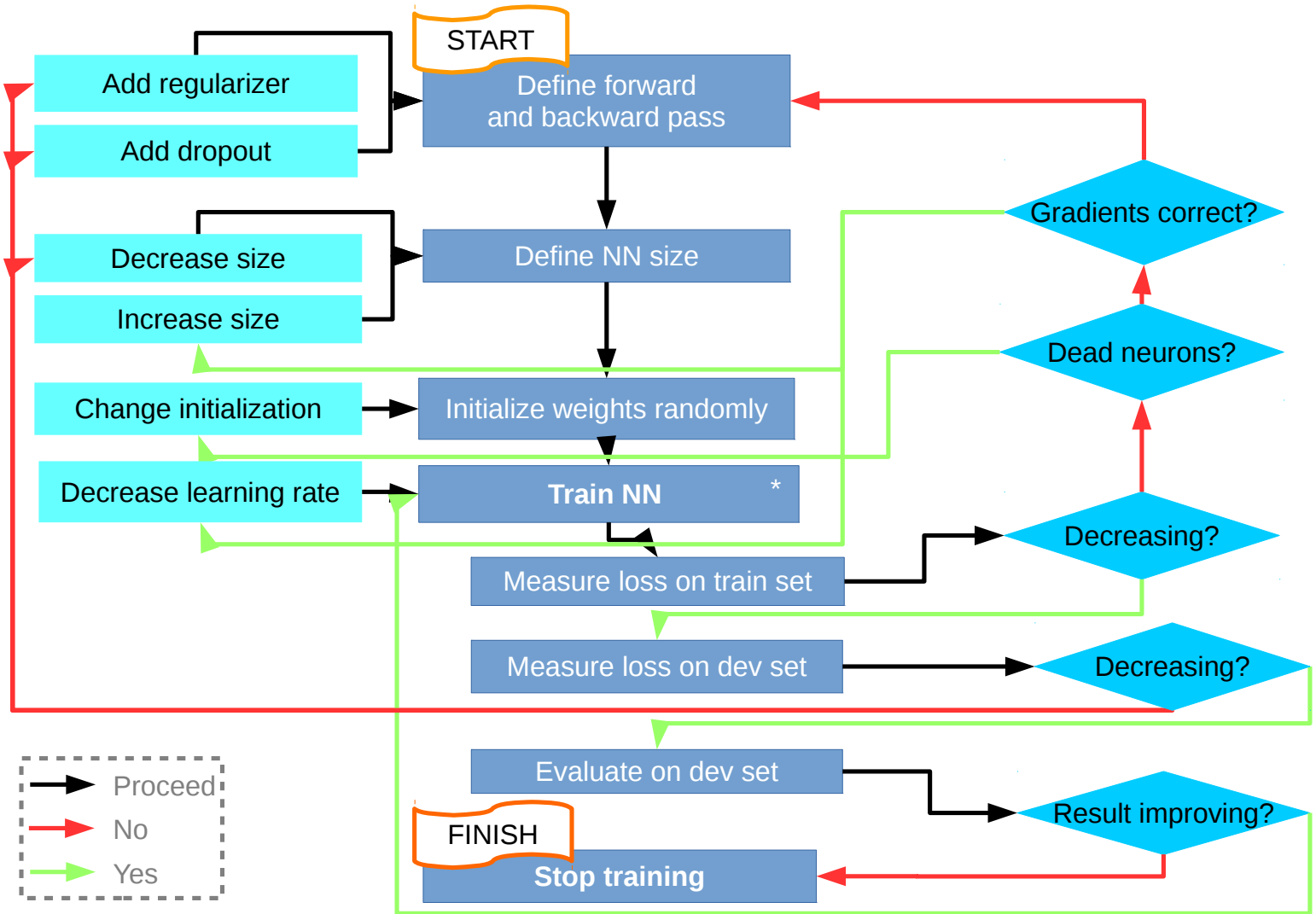
Wann soll der Lernprozess beendet werden?

- **Early Stopping**: sobald der Fehler auf dem Validationsset steigt
- **nach T Iterationen**: Zeitlimit, Datenlimit
- **Größe der Lernrate**: wenn $\eta_t \leq \epsilon$
- **Größe des Updates**: wenn Update $w_{t+1} - w_t \leq \epsilon$

In der Praxis


HOW TO TRAIN YOUR
Neural Network

How to train your NN



* Assuming simple SGD. Use adaptive learning rates, momentum and the like to push your results to state-of-the-art.

NLP (almost) from Scratch

NLP from Scratch [Collobert et al., 2011]: NLP-Aufgaben

Wort-Tagging-Probleme

- **POS-Tagging**:

And	now	for	something	completely	different
CC	RB	IN	NN	RB	JJ

- **Chunking**:

We	saw	the	yellow	dog
S-NP	B-VP	B-NP	I-NP	E-NP

- **Named Entity Recognition** (NER):

U.N.	official	Ekeus	heads	for	Baghdad
S-ORG	O	S-PER	O	O	S-LOC

- **Semantic Role Labeling** (SRL):

John	ate	the	apple
S-ARG0	S-REL	B-ARG1	E-ARG1

NLP from Scratch [Collobert et al., 2011]: Modell

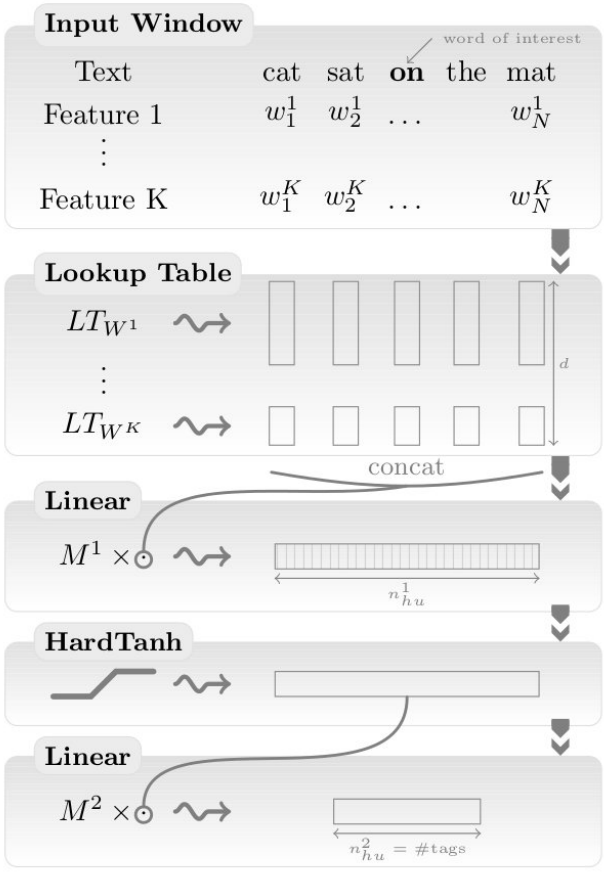


Abbildung 6: Window Approach, Source: [Collobert et al., 2011]

Training:

- SGD mit $\eta = 0.01$
- Cross-Entropy-Loss

Netzwerk:

- Wort-Embeddings der Dimensionalität 50
- 1 hidden Layer für POS, Chunking, NER mit 300 Neuronen
- 2tes hidden Layer für SRL mit 500 Neuronen

NLP from Scratch [Collobert et al., 2011]: Experimente

1. **“from Scratch”**: Input ist Wortfenster, nur Word-Embeddings
2. **Multi-Tasking**: Parameter für mehrere Aufgaben teilen, z.B. Word-Embeddings gemeinsam für POS und SRL lernen
3. **Pre-Training**: Pairwise-Ranking-Loss für korrektes Input und korruptem Input mit zentralem Wort ersetzt durch zufälliges Wort ($\rightarrow$ Language Modeling)
4. **Features**: Input um Features erweitert, pro Feature eine Embedding-Matrix, z.B. POS-Tags

NLP from Scratch [Collobert et al., 2011]: Embeddings i

FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
454	1973	6909	11724	29869	87025
PERSUADE	THICKETS	DECADENT	WIDESCREEN	ODD	PPA
FAW	SAVARY	DIVO	ANTICA	ANCHIETA	UDDIN
BLACKSTOCK	SYMPATHETIC	VERUS	SHABBY	EMIGRATION	BIOLOGICALLY
GIORGI	JFK	OXIDE	AWE	MARKING	KAYAK
SHAHEED	KHWARAZM	URBINA	THUD	HEUER	MCLARENS
RUMELIA	STATIONERY	EPOS	OCCUPANT	SAMBHAJI	GLADWIN
PLANUM	ILIAS	EGLINTON	REVISED	WORSHIPPERS	CENTRALLY
GOA'ULD	GSNUMBER	EDGING	LEAVENED	RITSUKO	INDONESIA
COLLATION	OPERATOR	FRG	PANDIONIDAE	LIFELESS	MONEO
BACHA	W.J.	NAMSOS	SHIRT	MAHAN	NILGIRIS

Table 6: Word embeddings in the word lookup table of a SRL neural network trained from scratch, with a dictionary of size 100,000. For each column the queried word is followed by its index in the dictionary (higher means more rare) and its 10 nearest neighbors (arbitrarily using the Euclidean metric).

Abbildung 7: SRL from Scratch embeddings, Quelle: [Collobert et al., 2011]

NLP from Scratch [Collobert et al., 2011]: Embeddings ii

FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
454	1973	6909	11724	29869	87025
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

Table 7: Word embeddings in the word lookup table of the language model neural network LM1 trained with a dictionary of size 100,000. For each column the queried word is followed by its index in the dictionary (higher means more rare) and its 10 nearest neighbors (using the Euclidean metric, which was chosen arbitrarily).

Abbildung 8: Language model embeddings, Quelle: [Collobert et al., 2011]

NLP from Scratch [Collobert et al., 2011]: Ergebnisse

- **Universal NLP-Tagger**: dieselbe einfache Architektur für Aufgaben unterschiedlicher Komplexität erfolgreich
- **Pre-Training** auf großen Textkorpora verbessert alle Modelle
- **“Almost from Scratch”**: State-of-the-Art Ergebnisse konnten nicht nur mit Word-Embeddings erreicht werden
- **Größerer Kontext**: Für SRL hilft es, größeren Kontext als nur Wortfenster zu betrachten (nächste Woche)


INSIGHTS
LIGHTYEAR
INSIGHTS EVERYWHERE
imgflip.com

Ausblick

Nächstes Mal

Word Embeddings

- word2vec
- Neural Language Models

Convolutional Neural Networks

References i


Bottou, L. (2012).
Stochastic gradient descent tricks.
In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer.


Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011).
Natural language processing (almost) from scratch.
The Journal of Machine Learning Research, 12:2493–2537.


Duchi, J., Hazan, E., and Singer, Y. (2011).
Adaptive subgradient methods for online learning and stochastic optimization.
Journal of Machine Learning Research, 12(Jul):2121–2159.

References ii


Glorot, X. and Bengio, Y. (2010).
Understanding the difficulty of training deep feedforward neural networks.
In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*.


He, K., Zhang, X., Ren, S., and Sun, J. (2015).
Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.
In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034.


Kingma, D. and Ba, J. (2015).
Adam: A method for stochastic optimization.
San Diego.

References iii


LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012).
Efficient backprop.
In *Neural networks: Tricks of the trade*, pages 9–48. Springer.


Martens, J. (2010).
Deep learning via hessian-free optimization.
In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 735–742.


Nesterov, Y. (1983).
A method of solving a convex programming problem with convergence rate o (1/k2).

References iv


Polyak, B. T. (1964).
Some methods of speeding up the convergence of iteration methods.
USSR Computational Mathematics and Mathematical Physics, 4(5):1–17.


Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014).
Dropout: a simple way to prevent neural networks from overfitting.
Journal of Machine Learning Research, 15(1):1929–1958.


Sutskever, I., Martens, J., Dahl, G. E., and Hinton, G. E. (2013).
On the importance of initialization and momentum in deep learning.
ICML (3), 28:1139–1147.

References v

Tieleman, T. and Hinton, G. (2012).
Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude.
COURSERA: Neural Networks for Machine Learning, 4(2).

Zeiler, M. D. (2012).
Adadelta: an adaptive learning rate method.
arXiv preprint arXiv:1212.5701.