SeaRNN

M. Bacher

ICL HD

18th of October 2018
Outline

1. Intro
2. SeaRNN
   • 1st Experiment
3. Performance
   • 2nd experiment
4. Neural Machine Translation
5. Conclusion | Critique
idea:
- improve RNN training
- integrate L2S into RNNs
RNNs

- neural network
- works on sequences
- used for tagging, machine translation
Motivation

problems of RNN training:
- maximum likelihood estimation
- teacher forcing
- local loss
MLE

- comes close to 0/1 loss
- only rewards reference
- not close to reference
Teacher Forcing

- training on ground truth
- testing on predictions
- leads to compounding errors
Local Level Loss

- loss is “local”
- loss function doesn’t consider the full sequence
Searn

- roll-ins
- roll-outs
$x = \text{command}$

$y = \text{command}$

$\phi(x) \xrightarrow{} \text{Roll-in} \xrightarrow{\text{Cost-sensitive loss} \ L_3(c_3)} \text{Roll-outs} \xrightarrow{} \hat{y}_m : \text{command} \rightarrow c_3(m) : 0.00$

$\hat{y}_n : \text{contend} \rightarrow c_3(n) : 0.43$

$\hat{y}_o : \text{cooperate} \rightarrow c_3(o) : 0.78$

\textbf{Figure 1:} Leblond et al.
- apply Searn to RNNs
- use rollouts
- rollouts enable new losses
SeaRNN Basic Idea

- roll-in the RNN
- at timestep t
- roll-out all actions
- calculate cost
- use cost for gradient update
Algorithm 1 SEARNN algorithm (for a simple encoder-decoder network)

1: Initialize the weights $\omega$ of the RNN network.
2: for $i$ in 1 to $N$ do
3:     Sample $B$ ground truth input/output structured pairs $\{(x^1, y^1), \ldots, (x^B, y^B)\}$
4:         # Perform the roll-in/roll-outs to get the costs. This step can be heavily parallelized.
5:     for $b$ in 1 to $B$ do
6:         Compute input features $\phi(x^b)$
7:             # Roll-in.
8:         Run the RNN until $t^{th}$ cell with $\phi(x^b)$ as initial state by following the roll-in policy
9:             (see Appendix A.2 for details in the case of reference roll-in policy)
10:        Store the sequence of hidden states in order to perform several roll-outs
11:    for $t$ in 1 to $T$ do
12:        # Roll-outs for all actions in order to collect the cost vector at the $t^{th}$ cell.
13:            for $a$ in 1 to $A$ do
14:                Pick a decoding method (e.g. greedy or beam search)
15:            Run the RNN from the $t^{th}$ cell to the end by first enforcing action $a$ at cell $t$,
16:                and then following the decoding method.
17:            Collect the cost $c_t^b(a)$ by comparing the obtained output sequence $\hat{y}_t^b(a)$ to $y^b$
18:        end for
19:    end for
20:    Derive a loss for each cell from the collected costs
21:    Update the parameters of the network $\omega$ by doing a single gradient step
22: end for
Policy choice

- follow LOLS [Chang et al.]

<table>
<thead>
<tr>
<th>roll-out →</th>
<th>Reference</th>
<th>Mixture</th>
<th>Learned</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓ roll-in</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td></td>
<td>Inconsistent</td>
<td></td>
</tr>
<tr>
<td>Learned</td>
<td>Not locally opt.</td>
<td>Good</td>
<td>RL</td>
</tr>
</tbody>
</table>

Figure 3: Chang et al.
New Losses

- use sequence information
- reward good, non-ground truth results
Log Loss

\[ L_t(s_t; c_t) = -\log\left( \frac{e^{s_t(a^*)}}{\sum_{i=1}^{A} e^{s_t(i)}} \right) \]  

\[ a^* = \text{argmin}_{a \in A} c_t(a) \]
Log Loss

- very similar to MLE
- maximize lowest cost action
- not ground truth action
Kullback Leibler

- see costs as probability distribution
- apply softmax to cost distribution
\[- P_C = \text{Cost distribution over actions} \]
\[- P_M = \text{Model probability over actions} \]
- similarity \( P_C \) and \( P_M \)
Kullback Leibler

\[ L_t(s_t; c_t) = - \sum_{a=1}^{A} (P_C(a) \ast \log(P_M(a))) \]  \hspace{1cm} (3)
Kullback Leibler

- provides more information
Optimization

- use online variant of Searn
- taken from Chang et al.
- works on mini-batches
First Experiment

- two datasets:
- OCR
- Corrupted Text Correction
OCR

- English words
- sequence of hand written characters
Corrupted Text

- 10 character sequence
- some characters randomly replaced
- random replacement of 30% or 50%
## Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>MLE</th>
<th>LL</th>
<th>KL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>learned mixed</td>
<td>reference learned</td>
</tr>
<tr>
<td>OCR</td>
<td>2.8</td>
<td>1.9</td>
<td>2.5</td>
</tr>
<tr>
<td>Spelling</td>
<td>0.3</td>
<td>17.7</td>
<td>19.5</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>37.1</td>
<td>43.2</td>
</tr>
</tbody>
</table>

**Figure 4:** Leblond et al.
Performance

- obvious problem
- for one gradient step
- roll-out over all actions
- OCR roll-out -> 26 actions
Performance

- machine translation -> 100 000 actions
Scaling Up

- don’t roll-out all actions
- sample actions
- 3 sampling techniques
Scaling Up

- stochastic policy sampling
- biased policy sampling
- top-k sampling
Adapted Losses

- LL and KL applied to the sampled tokens
2nd experiment

- use sampling techniques
- on OCR and Spelling
Results

- 5x computation time speedup

<table>
<thead>
<tr>
<th>Dataset</th>
<th>MLE</th>
<th>LL</th>
<th>KL</th>
<th>sLL</th>
<th>sKL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>uni.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>pol.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>bias.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>top-k</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>uni.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>pol.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>bias.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>top-k</td>
<td></td>
</tr>
<tr>
<td>OCR</td>
<td>2.8</td>
<td>1.9</td>
<td>1.0</td>
<td>1.7</td>
<td>1.8</td>
</tr>
<tr>
<td>Spelling</td>
<td>0.3</td>
<td>19.3</td>
<td>17.7</td>
<td>17.6</td>
<td>17.6</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>41.9</td>
<td>37.1</td>
<td>37.0</td>
<td>37.0</td>
</tr>
</tbody>
</table>

Figure 5: Leblond et al.
NMT

- neural Machine Translation
- German - English
- don’t use one of the sampling methods
- use “custom” sampling
- mixing of top-k and ground truth neighbors
Policy Choice

"we use a reference roll-in and a mixed roll-out."

<table>
<thead>
<tr>
<th>roll-out →</th>
<th>Reference</th>
<th>Mixture</th>
<th>Learned</th>
</tr>
</thead>
<tbody>
<tr>
<td>↓ roll-in</td>
<td>Inconsistent</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learned</td>
<td>Not locally opt.</td>
<td>Good</td>
<td>RL</td>
</tr>
</tbody>
</table>

Figure 6: Chang et al.
“One potential factor is that our reference policy is not good enough to yield valuable signal when starting from a poor roll-in. Another possibility is that the underlying optimization problem becomes harder when using a learned rather than a reference roll-in.”

[Leblond et al.]
## Results

<table>
<thead>
<tr>
<th>MLE*</th>
<th>MIXER*</th>
<th>SeaRNN (conv)</th>
<th>MLE†</th>
<th>BSO†</th>
<th>MLE’</th>
<th>AC’</th>
<th>MLE</th>
<th>SeaRNN</th>
<th>MLE (dropout)</th>
<th>SeaRNN (dropout)</th>
</tr>
</thead>
<tbody>
<tr>
<td>17.7</td>
<td>20.7</td>
<td>20.5</td>
<td>22.5</td>
<td>23.8</td>
<td>25.8</td>
<td>27.5</td>
<td>24.8</td>
<td>26.8</td>
<td>27.4</td>
<td><strong>28.2</strong></td>
</tr>
</tbody>
</table>

**Figure 7:** Leblond et al.
Conclusion

- “SEARNN is a full integration of the L2S ideas to RNN training, whereas previous methods cannot be used for this purpose directly. “[Leblond et al.]”
- clear improvement over MLE
- mostly consistent results with LOLS
Critique

Computational performance?
Improvement by sampling?
Performance comparison to other systems?
NMT Policy Choice

Choice of Reference Roll In?
Chang literally says “Roll-in with $\pi^{ref}$ is bad.”
Is this somewhat teacher forcing?
Would this work with better learned policy?
Sources

Remi Leblond, Jean-Baptiste Alayrac, Anton Osokin, and Simon Lacoste-Julien. SEARNN: Training RNNs with global-local losses
