Neural Machine Translation using Sequence Level Training

Kyu-Young Kim
Department of Computer Science
Stanford University
kykim153@stanford.edu

Abstract

Deep learning has recently been successfully applied to a wide range of natural language processing (NLP) tasks to advance the state of the art. Recurrent neural networks (RNN), in particular, are now widely used for such tasks as text summarization and machine translation. In a typical training scheme, these models are provided at each time step with the ground truth history with some context to predict the next token in a sequence. At inference time, however, the model is conditioned on its own predictions to generate the entire sequence. This discrepancy between the conditioning distributions makes generation at inference time unstable and potentially lead to significant accumulation of errors when generating long sequences. In this work, we explore several approaches that have recently been proposed to address this issue, and, in particular, a sequence level training method using an idea from reinforcement learning (RL) in depth.

1 Introduction

Recurrent neural networks are designed to process sequential data and, hence, have recently been widely adopted for many NLP applications. For instance, RNNs (or variants thereof such as LSTM) have been applied to tasks such as machine translation, image captioning, and abstractive summarization to achieve impressive results [3].

Recurrent models are trained to maximize the likelihood of generating the sequence of target tokens given some context. This is generally done by maximizing in each time step the likelihood of each target token given the model's previous state and the previous *target* token. At test time, however, the model generates one token at a time conditioned on its own state and the previous *predicted* token. This training scheme of using per-token, cross-entropy loss is often referred to as maximum likelihood estimation (MLE) or teacher-forcing. MLE training poses two issues that limit the quality of the output generated by these sequence models: *exposure bias* and *loss mismatch* [9].

Exposure bias refers to the discrepancy between the distribution the model is conditioned on during training and that during inference. The process of generating a sequence one token at a time based on its own predictions is unstable since the model can end up in a state that it has not encountered during training. Once in such a state, the model can generate a garbled output, which is then fed back to the model for the subsequent steps, quickly leading to a low-quality output. It is even possible for the model to repeatedly generate a series of tokens without ever stopping.

Loss mismatch refers to the problem where the loss function used to train the model is different than the metric used to evaluate the model. The loss function used at training time is often at the word level in order to have the model maximize the likelihood of the next correct word. For evaluation, however, sequence level, discrete metrics such as BLEU [8] or ROUGE [6] are typically used. These metrics are supposed to capture the notion of the quality of the model generated sequence in comparison to

German Sentences
Wir werden alle geboren. Wir bringen Kinder zur Welt. Das ist ein Bild der Cannery Row von 1932 .
English Sentences
We’re all born. We all bring our children into the world. This is a shot of Cannery Row in 1932 .

Table 1: Sample sentences from the train set.

the human generated sequence. Such evaluation metrics are difficult to directly optimize for, because they are often not differentiable.

These issues with MLE training have recently been explored by the research community using ideas from reinforcement learning. We study in depth one such approach that uses an alternative sequence level training method based on what is referred to as the REINFORCE algorithm [11].

2 Related Work

Various techniques have been proposed to address the problem with accumulation of errors when generating long sequences using typical recurrent models. Beam search is one such technique that potentially finds a higher quality sequence by maintaining a set of best candidates during the decoding stage and choosing the highest scoring candidate at the end of generation. This approach often produces a better sequence than the one from greedy decoding that greedily selects the token with the highest score in each time step. The main downside of using beam search is that it is computationally heavy. It can often be significantly slower than greedy decoding depending on the size of the beam (the number of candidates). Hence, for applications where latency is critical, beam search poses a different challenge unless an accelerated hardware is used.

3 Dataset

The sequence models explored in this work are applicable to a variety of NLP tasks, but we in particular considered machine translation. The data for the MT task is from the German-to-English text translation track of the IWSLT 2014 evaluation campaign, which contains lecture transcription and translation of TED and TEDx talks [4]. The training set contains over 150,000 German and English sentences. We pre-processed the training data using the Moses tokenizer [5] and lower-cased all sentences as done in [9]. To create input and output vocabulary sets, we collected all distinct words in the training data except for the words that occurred less than three times. This was for better generalization performance and computational efficiency. The dev and test sets were also prepared the same way as described in [9]. Specifically, dev2010 and dev2012 were combined to form the validation set and tst2010, tst2011, and tst2012 were combined to form the test set.

4 Models and Methods

For the training schemes studies in this work, we focus on RNN-based models which are a popular choice for text generation tasks.

4.1 Sequence-to-Sequence Model

A sequence-to-sequence model is an RNN-based sequence learning model that uses one RNN cell called encoder to map the input sequence to a vector of a fixed size and another RNN cell called decoder to map this vector into the target sequence [10]. More formally, given the input sequence $(x_1, ..., x_T)$, a vanilla seq2seq model first generates the fixed-sized vector representation v which is the last hidden state of the encoder. Using this representation vector v as its initial hidden state, the decoder produces one token at a time to generate the sequence $(y'_1, ..., y'_{T'})$. Each token y'_t is generated by projecting the hidden state of the decoder at time t to a vector of size equal to the size of the output vocabulary and selecting the one with the highest probability after feeding it through a softmax layer:

$$p_t = W_o h_t \tag{1}$$
$$l_t = \text{softmax}(p_t) \tag{2}$$
$$y'_t = \text{argmax}(l_t) \tag{3}$$

The loss is calculated by summing the cross-entropy loss over each token in the generated sequence and normalizing the sum by the batch size:

$$loss = -\frac{1}{m} \sum_{t=1}^{T'} \sum_{c=1}^{s} y_{t(c)} \log(l_{t(c)}) \tag{4}$$

where s is the size of the output vocabulary and m is the batch size.

4.2 Reinforcement Learning

Viewing a sequence-to-sequence task as an instance of RL problem, we can apply various RL techniques to sequence model training. One way to frame text generation task as an RL problem is by considering the RNN decoder as an agent and its hidden state as the environment it interacts with. The probability distribution over the output vocabulary as computed by the softmax layer can then be viewed as the set of actions the agent can take. Based on the chosen action, the agent receives reward computed using an evaluation metric such as BLEU.

To formalize, given a set of input and target sequences $\{X^1, ..., X^m\}$ and $\{Y^1, ..., Y^m\}$, the loss function is

$$loss = -\sum_{i=1}^{m} \mathbb{E}_{\pi(Y'^i|X^i)}[R(Y'^i|Y^i)] \tag{5}$$

where $R(Y'^i|Y^i)$ is the reward the agent receives for generating the sequence Y'^i when the target sequence is Y^i. The gradient of the loss is then given by

$$\nabla loss = -\sum_{i=1}^{m} \mathbb{E}_{\pi(Y'^i|X^i)}[R(Y'^i|Y^i) \nabla \log \pi(Y'^i|X^i)] \tag{6}$$

and this is a policy gradient method referred to as the REINFORCE algorithm [11].

Conceptually, the above formulation allows us to deal with both the exposure bias and loss mismatch problems by sampling sequences directly from the model and using a reward function R that can be an evaluation metric of our choice.

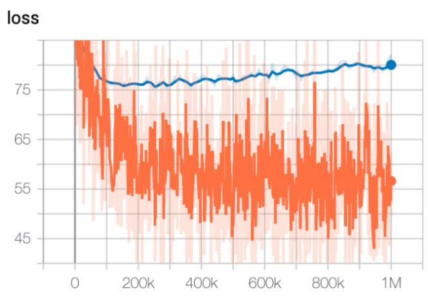
However, direct application of the algorithm starting with a random policy is likely to be problematic. This is because in text generation the size of the possible tokens (actions) the model can produce is often large. Hence, the MIXER algorithm described in [9] proposes a type of curriculum training where the model is initially trained using MLE with ground truth sequences and in later epochs trained using REINFORCE with its own predictions. This approach has a resemblance to the scheduled sampling idea in [3] except that it gradually have the model produce the whole sequence rather than sampling from the model at the token level.

5 Experiments and Analysis

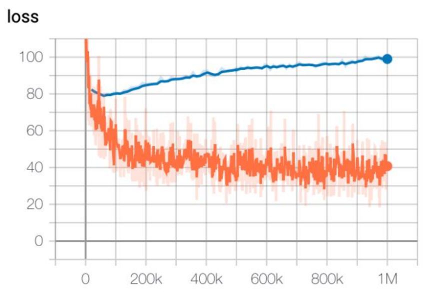
All of the experiments were conducted using the TensorFlow library [1]. The code is available on GitHub at https://github.com/kky1638/seq2seq.

5.1 Vanilla Seq2seq Model

We ran an initial set of experiments on vanilla seq2seq model with a basic encoder and decoder structure. The models were trained with batch size of 16, learning rate of 0.001, and for 1 million



(a) 128 hidden units



(b) 512 hidden units

Figure 1: Vanilla seq2seq model

steps in order to observe how the overall training progresses. We used Adam optimizer and applied gradient clipping at 10.0 to avoid potential NaNs in gradient. Basic LSTM cell implementation was used for both encoder and decoder.

The figure above shows training (orange) and evaluation (blue) graphs for two particular models – one with 128 hidden units and another with 512 hidden units. Comparing the two training graphs, it is evident that the model needs relatively large capacity in order to fit the training set well. However, simply increasing the number of hidden units leads to overfitting and results in higher loss on the evaluation set. Hence, we need to increase the model capacity with an appropriate regularization.

5.2 Attention Model

In a vanilla seq2seq model, the encoder maps the input sequence into a fixed-sized vector representation that is provided as an initial state to the decoder. However, this is often too limiting as the encoder has to compress all the necessary information in its state across many time steps. To address this issue, we allow the decoder to *attend* to the encoder states as it generates the output, effectively increasing the memory capacity of the model [7].

Besides adding an attention layer, we experimented with dropout as a regularization technique. In the context of an RNN-based model, dropout can be applied to the input, output, and state layer. In our experiment, we applied dropout with keep probability of 0.8 only to the input and output layers and not to the state layer. We made this choice because the cell state conceptually represents its memory and hence applying dropout to the memory made less sense than applying it to the input and output layers.

Lastly, in palce of the basic LSTM cell, we used layer-normalized LSTM cell [2] for both the encoder and decoder that led to better convergence.

The figure below shows that the attention model suffers less from overfitting and achieves lower training error than the vanilla seq2seq model even when trained for only 1/10 of the steps. Note, however, that the speed as measured by the number of training steps per second also decreased by about 1/4. We suspect that this is due to using the more complex LSTM implementation.

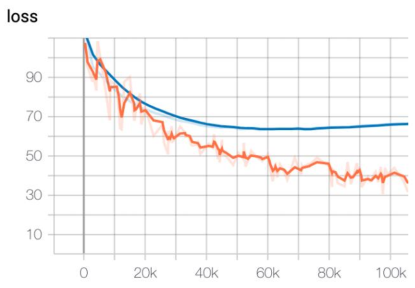


Figure 2: Attention model with dropout

5.3 RL Model

We closely followed the MIXER idea explored in [9] to train our RL model. Specifically, we implemented a curriculum learning schedule where we initially train the model in the regular MLE fashion and gradually transition to using the model's own predictions to compute the loss based an evaluation metric (negation thereof). This learning schedule is controlled by two parameters – the training step at which to start the transition and the training step at which to exclusively use the RL loss. Based on these two parameters, we decide when to have the decoder use its own predictions in place of the ground truth sequence. As training progresses, we apply a linear annealing to gradually expose the model to its own distribution. The loss is then a weighted sum of the cross-entropy loss and the RL loss.

For the RL metric, we used the BLEU score which estimates the quality of a generated sentence as compared to a set of reference sentences using *n*-gram overlap:

$$BLEU = BP \cdot \exp(\frac{1}{4}\sum_{n=1}^{4}\log p_n) \tag{7}$$

In the equation, p_n is the *n*-gram precision and BP is the brevity penalty that penalizes a candidate sequence that is too short compared to the reference sentences [8]. It is a popular metric especially for machine translation task.

We experimented with the same attention model as before but additionally applied the RL-based curriculum learning. In particular, we set the anneal start step at 10,000 and the stop step at 200,000 on top of the layer-normalized LSTM encoder and decoder with input and output dropout. The main challenge was with figuring out an appropriate annealing schedule which seems to have a large impact on model convergence. Also, we learned that it is important to properly normalize the RL metric since otherwise its scale may be significantly different than that of the cross-entropy loss with which is combined to yield the final loss. Unfortunately, the RL-based model did not perform any better than the the attention model that used a more sophisticated LSTM cell with dropout.

6 Conclusion

RNN-based models are widely adopted for a variety of NLP tasks. Typically, the model is trained using the ground truth sequences but is conditioned on its own predictions to generate output at inference time. This discrepancy can lead to poor quality output especially when generating long sequences. Various approaches such as beam search and decoder attention are widely used to address this issue. Viewing sequence learning as an RL problem, we can also apply RL techniques such as the REINFORCE algorithm to deal with the problem. This has the conceptual benefit of addressing both the exposure bias and loss mismatch problems.

We explored both MLE and RL-based training schemes on various seq2seq model architectures, and, on the particular dataset used, more sophisticated model architecture with MLE training was sufficient to achieve good results. We hypothesize that this is due to that (a) sub-optimal curriculum learning schedule was used and (b) the dataset contained sequences that are too short to uncover the issue of learning long range dependencies.

For future work, we would like to run more experiments with various curriculum learning schedules to gain more insight into how they relate to model convergence. It would also be interesting to explore when the RL-based approach begins to shine with respect to the length of the sequences the model needs to learn.

7 Contributions

I was the sole member of the team and worked on all parts of the project including literature review, implementation, and writing the report.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow,

Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

[3] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS)*, 2015.

[4] M. Cettolo, J. Niehues, S. Stüker, L. Bentivogli, and M. Federico. Report on the 11th iwslt evaluation campaign, iwslt 2014. In *Proceedings of IWSLT*, 2014.

[5] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of ACL Demo and Poster Sessions*, 2007.

[6] C.-Y. Lin. Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, 2004.

[7] M.-T. Luong, H. Pham, and C. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

[8] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Association for Computational Linguistics (ACL)*, 2002.

[9] M. Ranzato, S. Chopara, C. Auli, and W. Zarembra. Sequence Level Training with Recurrent Neural Networks. In *International Conference on Learning Representations (ICLR)*, 2016.

[10] I. Sutskever, O. Vinyals, and Q. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS)*, 2014.

[11] R. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 1992.