Sequence-to-sequence Machine Translation Report

Ethan Xuanyue Yang
xuanyuey@cs.cmu.edu

1 Introduction

In this report we present the implementation (Github) of an RNN-based sequence-to-sequence model for machine translation, composed an encoder for source sentence and a decoder with attention over source encodings for target sentence. We conduct a series of experiment and show the effectiveness of our settings of certain hyperparameters by the comparison of BLEU scores. We then discuss several factors that show influence on the performance.

2 Model

The model has a classic encoder-decoder architecture to translate a source sentence into a target sentence.

2.1 Encoder

We first use a source embedding matrix $\mathbf{E}_{\text{src}} \in \mathbb{R}^{V_{\text{src}} \times D}$, where V_{src} is source vocabulary size, D z to embed the input tokenized source sentence $\{x_t\}_{t=1}^{T_{\text{src}}}$ (word ID sequence) to $\{\mathbf{x}_t\}_{t=1}^{T_{\text{src}}}$. Then we use an L-layer bidirectional LSTM_{src} to encode it:

$$\overrightarrow{\mathbf{h}}^{\text{src}}_{l,t} = \overrightarrow{\text{LSTM}}_{\text{src}}(\mathbf{h}^{\text{src}}_{l-1,t}, \overrightarrow{\mathbf{h}}^{\text{src}}_{l,t-1}), \tag{1}$$

$$\overleftarrow{\mathbf{h}}^{\text{src}}_{l,t} = \overleftarrow{\text{LSTM}}_{\text{src}}(\mathbf{h}^{\text{src}}_{l-1,t}, \overleftarrow{\mathbf{h}}^{\text{src}}_{l,t+1}), \tag{2}$$

$$\mathbf{h}^{\text{src}}_{l,t} = \begin{bmatrix} \overrightarrow{\mathbf{h}}^{\text{src}}_{l,t} \\ \overleftarrow{\mathbf{h}}^{\text{src}}_{l,t} \end{bmatrix}, \tag{3}$$

where

$$\overrightarrow{\mathbf{h}}^{\text{src}}_{l,0} = \overleftarrow{\mathbf{h}}^{\text{src}}_{l,T_{\text{src}}+1} = \mathbf{0}, \tag{4}$$

$$\mathbf{h}^{\text{src}}_{0,t} = \mathbf{x}_t. \tag{5}$$

Eventually we obtain source encoding matrices for each layer:

$$\mathbf{H}^{\text{src}}_{l} = \left[\mathbf{h}^{\text{src}}_{l,1}, \cdots, \mathbf{h}^{\text{src}}_{l,T_{\text{src}}}\right]. \tag{6}$$

2.2 Decoder

Given a target sentence $\{y_t\}_{t=1}^{T_{\text{tgt}}}$, similarly we use a target embedding matrix $\mathbf{E}_{\text{tgt}} \in \mathbb{R}^{V_{\text{tgt}} \times D}$ to embed it to $\{\mathbf{y}_t\}_{t=1}^{T_{\text{tgt}}}$. Then an L-layer LSTM_{tgt} serve as target language model along with attentional contexts over the source encodings:

$$\mathbf{h}^{\text{tgt}}_{l,t} = \text{LSTM}_{\text{tgt}}\left(\begin{bmatrix} \mathbf{h}^{\text{tgt}}_{l,t} \\ \mathbf{c}_{l,t-1} \end{bmatrix}, \mathbf{h}^{\text{tgt}}_{l,t-1}\right), \tag{7}$$

where

$$\mathbf{h}^{\text{tgt}}_{l,0} = \begin{bmatrix} \overrightarrow{\mathbf{h}}^{\text{src}}_{l,T} \\ \overleftarrow{\mathbf{h}}^{\text{src}}_{l,0} \end{bmatrix}, \tag{8}$$

$$\mathbf{h}^{\text{tgt}}_{0,t} = \mathbf{y}_t. \tag{9}$$

We feed the previous context $\mathbf{c}_{l,t}$ as a part of input at t to inform the decoder of previous alignment decision similar to Luong et al. (2015). Each $\mathbf{c}_{l,t}$ is obtained using scaled dot product attention (Vaswani et al., 2017):

$$\mathbf{a}_{l,t} = (\text{softmax}(\mathbf{q}^{\mathsf{T}}_{l,t}\mathbf{K}_l))^{\mathsf{T}}, \tag{10}$$

$$\mathbf{c}_{l,t} = \frac{1}{\sqrt{H}}\mathbf{V}_l\mathbf{a}_{l,t}, \tag{11}$$

where H is the hidden size, and the query $\mathbf{q}_{l,t-1}$, keys $\mathbf{K}_l$, and values $\mathbf{V}_l$ is computed by:

$$\mathbf{q}_{l,t} = \text{MLP}_{\text{q}}(\mathbf{h}_{l,t}), \tag{12}$$

$$\mathbf{K}_l = \text{MLP}_{\text{k}}(\mathbf{H}^{\text{src}}_l), \tag{13}$$

$$\mathbf{V}_l = \text{MLP}_{\text{v}}(\mathbf{H}^{\text{src}}_l). \tag{14}$$

Finally we project the L-th target encodings and contexts to a V_{tgt}-dimensional logit vector, and softmax it with temperature T as the predicted probability distribution over the target vocabulary:

$$\mathbf{o}_t = \tanh\left(\mathbf{W}_{\text{o}}\begin{bmatrix} \mathbf{h}^{\text{tgt}}_{L,t} \\ \mathbf{c}_{L,t} \end{bmatrix}\right), \tag{15}$$

$$\mathbf{s}_t = \mathbf{W}_{\text{tgt}}\mathbf{o}_t, \tag{16}$$

$$\hat{\mathbf{p}}_t = \text{softmax}\left(\frac{\mathbf{s}_t}{T}\right), \tag{17}$$

$$\mathbb{P}(y|\{x_\tau\}_{\tau=1}^{T_{\text{src}}}, \{y_\tau\}_{\tau=1}^{t-1}) = \hat{\mathbf{p}}_t[y]. \tag{18}$$

We apply weight tying (Press and Wolf, 2016) to regularize target projection and embedding matrix by enforcing $\mathbf{W}_{\text{tgt}} = \mathbf{E}_{\text{tgt}}$.

2.3 Training Process

During training time we have the gold target sentence and thus we maximize the log-likelihood of it given by the model, or equivalently the cross entropy:

$$\mathcal{L} = \sum_{t=1}^{T_{\text{tgt}}} \log \mathbb{P}(y_t | \{x_\tau\}_{\tau=1}^{T_{\text{src}}}, \{y_\tau\}_{\tau=1}^{t-1}). \quad (19)$$

Generally in an encoder-decoder model we have a discrepancy between training and inference, specifically the decoder has access to the previous gold word when making sequential predictions during the former, while lacks it during the latter. To bridge this gap we occasionally feed the decoder with the previous predicted word $\hat{y}_{t-1}$ instead of the gold word y_{t-1}, similar to scheduled sampling (Bengio et al., 2015) but we are using a constant rate R. An issue comes consequently is the exploration-exploitation trade-off: trivially feeding the

$$\hat{y}_{t-1} = \arg\max_y \mathbb{P}(y | \{x_\tau\}_{\tau=1}^{T_{\text{src}}}, \{y_\tau\}_{\tau=1}^{t-1}) \quad (20)$$

might cause the sequential decisions to be stuck in local optima and even misled when the model is not well-trained yet. We thus change the deterministic $\arg\max_y$ to a sampling:

$$\tilde{y}_{t-1} \sim \mathbb{P}(y | \{x_\tau\}_{\tau=1}^{T_{\text{src}}}, \{y_\tau\}_{\tau=1}^{t-1}) \quad (21)$$

and feed $\tilde{y}_{t-1}$ as input to explore more possibilities. With the help of the straight-through Gumbel-softmax re-parameterization trick (Jang et al., 2016) we are able to do continuous optimization through these sampling step and over each decision $\hat{\mathbf{p}}_{t-1}$. Specifically the sampling could be reparameteriazed by a Gumbel distribution:

$$g_i \overset{\text{i.i.d.}}{\sim} \text{Gumbel}(0, 1), i = 1, \dots, V_{\text{tgt}} \quad (22)$$

$$\tilde{\mathbf{p}}_{t-1} = \text{softmax}\left(\frac{\mathbf{s}_{t-1} + \mathbf{g}}{T}\right), \quad (23)$$

$$\tilde{y}_{t-1} = \arg\max_y \tilde{\mathbf{p}}_{t-1}, \quad (24)$$

and we use $\mathbf{E}_{\text{tgt}}\tilde{\mathbf{p}}_{t-1}$ to calculate the gradient in the backward pass, but use $\tilde{y}_{t-1}$ in the forward pass to avoid propagating mixing errors due to feeding a mixture of word embeddings (Gu et al., 2018).

2.4 Decoding Process

Searching the best target sequence is essentially an exponential process and we use beam search to obtain a sub-optimal result in a polynomial space. With a beam size B, we start by feeding the start token <s> to the decoder and at each step $t > 1$ we have B target sequences as inputs. For each sequence we keep top B next words out of V_{tgt}, and then we keep top B new sequences out of BV_{tgt}. For ranking we consider the length-normalized log-likelihood of each sequence $\{\hat{y}_\tau\}_{\tau=1}^t$:

$$l(\{\hat{y}_\tau\}_{\tau=1}^t) = \sum_{\tau=1}^t \mathbb{I}(\hat{y}_\tau \neq \text{id}(\texttt{</s>})), \quad (25)$$

$$s(\{\hat{y}_\tau\}_{\tau=1}^t) = \frac{\sum_{\tau=1}^t \log \mathbb{P}(\hat{y}_t | \{x_\tau\}_{\tau=1}^{T_{\text{src}}}, \{\hat{y}_\tau\}_{\tau=1}^{t-1})}{l(\{\hat{y}_\tau\}_{\tau=1}^t)}, \quad (26)$$

where we enforce the decoder to only predict </s> for a sequence ended with </s>. Either when each of the B sequences has ended or maximum decoding length T_{max} has been reached we terminate the decoding and choose the highest-scored sequence as the result.

3 Experiment

3.1 Dataset

We use the IWSLT 2014 de-en dataset[1](Cettolo et al., 2012), which has 150K German-English training sentences. We use the pre-processed version of the training set from Ranzato et al. (2015) with long sentences chopped and rared words replaced by a special <unk> token and use the original validation and test sets. Wu build the source and target vocabularies out of the training set with words of frequency more than 2, resulting in sizes $V_{\text{src}} = 22824$ and $V_{\text{tgt}} = 32011$.

3.2 Settings

We list our experiment settings for the best model in Table 1. For LSTM_{src} we use $H/2$ per each direction. We also consider a RNN weight dropout probability for weight-dropped RNN (Merity et al., 2017) where a fixed dropout mask is applied to RNN weights at each time step per run, though in the final model we cancel this setting due to its unpromising performance as shown in Table 2. During training we reduce

[1]https://wit3.fbk.eu/mt.php?release=2014-01

Item	Value
Embedding Dimension D	512
RNN Hidden Size H	512
#(RNN Layer) L	2
Temperature T	0.5
Sampling Rate R	0.2
Dropout Probability	0.2
RNN Weight Dropout Probability	0
Optimizer	Adam
Initial Learning Rate	10^{-3}
Batch Size	64
Beam Size B	32
Max Decoding Length $T_{\max}$	200

Table 1: Settings

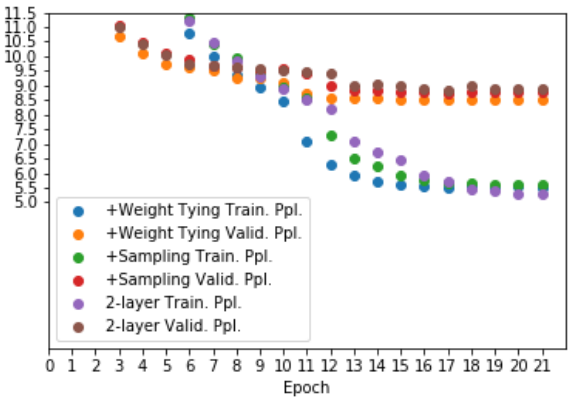

Figure 1: Training and Validating Perplexity vs Epoch

the learning rate by half and reload the previous model checkpoint each time we have seen a non-increasing validating perplexity after 1 patient step. We do early-stopping after 4 times of adjusting the learning rate but not witnessing improvement.

3.3 Results

We report the results in terms of validating perplexity (ppl.), validating BLEU and test BLEU[2] in Table 2, where we start from a "Base" 1-layer RNN model and gradually apply several changes. We use validating perplexity for model selection.

Model	Val Ppl.	Val. BLEU	Test BLEU
Base	9.11	29.65	27.61
+ Weight Dropping	9.35	29.12	27.03
2-Layer	8.85	29.99	27.75
+Sampling	8.74	29.79	27.78
+Weight Tying	**8.54**	**30.12**	**28.01**

Table 2: Results

3.4 Analysis

From the experiment result we mainly find three changes that are particularly helpful for improving the performance. The number of RNN Layer, weight tying and sampling (see 2.3). We have also tried weight dropping for RNNs but the eventually the converged model gives both a higher training and validation perplexity, suggesting that under the current settings we have, applying dropout on RNN weights act as a too strong regularization and potentially hinder the learning of the model. As shown effective in generalizing the RNN-based language models in Merity et al. (2017), this method might further improve our performance given more sophisticated hyper-parameter tuning.

Increasing the number of RNN layers is both a trivial and heavy-weight change, as it greatly increase the step time and GPU usage of the model. Its improvement could mainly be attributed to enable to model to consider higher-level information by adding the depth of the RNN-encoding layer. Intuitively the second layer of the encoder might render more informative word encodings based on the first layer encodings. And the second layer of the decoder might be able to make better prediction decision by considering the adjusting the decision from the first layer.

We plot the convergence processes in terms of training and validating perplexity for the 2-layer model and models we apply sampling and weight tying in Fig. 1. We would discuss the effect of adding sampling and weight tying below.

As discussed before, sampling would help reducing the training-inference discrepancy by simulating the decoding behaviour in the training time, ie, feeding the output from the previous time step as current input. These sampling steps would intuitively result in slower initial convergence rate for training when the model is not well-trained and have higher loss for making decision out of a non-gold input. This phenomenum is well reflected in Fig. 1 where for the first 10 epochs the training and validating perplexities are higher than others. But after this phase the model starts to achieve better performance than its simpler 2-layer peer. And

[2] calculated by multi-bleu.perl

eventually the model could have higher validating and test BLEU score due to these decoding steps have been more or less exposed in training time.

The last and the most significant improvement we make is by applying weight tying. From the convergence curves we could that it constantly achieves a better training and validating perplexities over others, and converges faster (4-5 epochs less). And it greatly increase the test BLEU by 0.23, more than that brought by deepening RNNs (0.14). We conjecture that tying the weights for the embedding matrix and the projection matrix informs the model to directly utilize the word embedding similarities for word prediction, and in turn force the model to learn more task-specific embeddings. Also without learning two large weight matrices, the model enjoys a simpler structure and thus converges faster. It's interesting that this regularization technique would result in both faster convergence of training and validating perplexity, thanks to its reasonable simplification without hurting the model's capability to make prediction in training time, compared to dropout.

Frequency	**+W.T.**	+S.	2-Layer
< 1	0	0	0
1	**0.280**	0.272	0.270
2	**0.392**	0.387	0.383
3	**0.424**	0.421	0.417
4	0.442	**0.444**	0.439
$[5, 10)$	**0.508**	0.503	0.497
$[10, 100)$	0.56	**0.572**	0.569
$[100, 1000)$	**0.603**	0.601	0.600
≥ 1000	**0.753**	0.753	0.750

Table 3: Word F-measures

Length	**+W.T.**	+S.	2-Layer
< 10	**32.78**	32.27	32.84
$[10, 20)$	30.29	**30.44**	30.18
$[20, 30)$	**28.22**	28.13	27.77
$[30, 40)$	**26.58**	25.98	25.93
$[40, 50)$	**25.72**	25.20	25.28
$[50, 60)$	**24.68**	23.81	24.51
≥ 60	20.61	**20.71**	21.62

Table 4: Sentence BLEUs

We further use `compare-mt` (Neubig et al., 2019) to compute the word F-measures bucketed by word frequency and sentence BLEUs bucketed by sentence length. As shown in Table 3 and 4, where "+W.T." is our best model with weight tying, "+S."the one with only sampling, and "2-Layer" the one without both, weight tying and sampling consistently give better results than the simple 2-layer version, but weight tying is outperformed by just sampling in some cases. This could be because weight tying is a hard regularization and a more complex model with more parameters without it would be able to cover some special cases. Overall weight tying still delivers the best result, in terms of predicting the correct words and composing the sentences relevant to the references.

Ref.	this is a game for outside.
+W.T.	**this is a game for outside.**
+S.	that's a game out there.
Ref.	the mechanism comes from the lie.
+W.T.	**the mechanism comes from the lie.**
+S.	the mechanism is from lie.
Ref.	so where do we go?
+W.T.	**so where do we go?**
+S.	so where are we going?
Ref.	and i think it makes the world look like this.
+W.T.	**and i think it makes the world look like this.**
+S.	and i think it's a sense of the world looks like this.
Ref.	imagine a plane full of smoke.
+W.T.	**imagine a plane full of smoke.**
+S.	imagine a plane full of a fish.
Ref.	the orange tractor.
+W.T.	the orange is <unk>.
+S.	**the orange tractor.**
Ref.	but it gets worse.
+W.T.	but it's even worse.
+S.	**but it gets worse.**
Ref.	i'm going to stop that.
+W.T.	i'll stop that.
+S.	**i'm going to stop that.**
Ref.	it's kind of like a potato.
+W.T.	it's about a potato.
+S.	**it's kind of like a potato.**
Ref.	if you burn natural gas, no.
+W.T.	if you burn gas , you're not.
+S.	**if you burn natural gas, no**

Table 5: Translation Examples

At last we showcase several translation examples produced by `compare-mt` (Neubig et al., 2019) in Table 5, where the first 5 our best model with weight typing wins, and the last 5 the one with just sampling wins. We could see that overall

our best model is able to generate more fluent sentences and cover the basic meaning, even in loss cases. While in some loss cases for the model without weight tying, sentences that are slightly incorrect such as "and i think it's a sense of the world looks like this.", and also sentences that are far off the references such as "imagine a plane full of a fish." are generated.

References

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179.

Mauro Cettolo, Christian Girardi, and Marcello Federico. 2012. Wit3: Web inventory of transcribed and translated talks. In *Conference of European Association for Machine Translation*, pages 261–268.

Jiatao Gu, Daniel Jiwoong Im, and Victor OK Li. 2018. Neural machine translation with gumbel-greedy decoding. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*.

Graham Neubig, Zi-Yi Dou, Junjie Hu, Paul Michel, Danish Pruthi, and Xinyi Wang. 2019. compare-mt: A tool for holistic comparison of language generation systems. In *Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL) Demo Track*, Minneapolis, USA.

Ofir Press and Lior Wolf. 2016. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.