11-731 Assignment 1: Neural Machine Translation

Shruti Rijhwani
Carnegie Mellon University
srijhwan@andrew.cmu.edu

Abstract

This report describes the implementation and results of a neural machine translation system built using DyNet, a dynamic neural network toolkit. The model is the standard encoder-decoder paradigm, to which attention is added in order to enhance translation. Additional techniques to improve BLEU score, like beam search decoding, removing singletons from the corpus and using dropout, are detailed as well, along with experiments showing the effects of various configurations. The system is trained, tuned and tested on German-English parallel data. The best BLEU score obtained on the test set is $\mathbf{23.68}$, a reasonable improvement over the baseline of 18.6.

1 Introduction

Machine translation (MT) is a well-studied Natural Language Processing task. An MT system is designed to take a sentence in a source language and automatically translate it to another language (the target language). It is, in general, a challenging task because the translation result needs to be fluent in the target language, as well as adequate in representing the meaning of the source sentence.

Traditionally, phrase-based statistical MT was used in state-of-the-art systems, which contains several components, including the language model, phrase segmentation, reordering and translation. Recently, neural approaches to MT have gained attention. These are trained from end-to-end to perform the complete translation process, instead of training different components separately. These techniques have been observed to improve on some of weaknesses seen in conventional MT (Wu et al., 2016). The network learns a representation of the source sentence (encoding) and then uses this encoding to generate tokens of the target sentence in the appropriate order (decoding) (Cho et al., 2014). Attention is a concept that facilitates better alignment between the source and target words using a soft 'attention' that represents the relevance of source words at each decoding time step. This also attempts to alleviate the problem of using a fixed-length encoding for variable source sentence lengths (Bahdanau et al., 2014).

This report describes a neural MT model using this attentional encoder-decoder design, that is implemented with the DyNet framework (Neubig et al., 2017). I also implemented minibatching, beam search, unknown word replacement and dropout, and report the effects these have on training time, validation set perplexity and BLEU score.

The best model, as tuned on the validation set, is then used to translate the test corpus. The BLEU score obtained is $\mathbf{23.68}$, with 1.7 seconds per sentence decoding time.

2 Model

The training corpus contains parallel sentences in the source language and the target language. Each pair has a source sentence $F = f_1, f_2, ..., f_{|F|}$ and a target sentence $E = e_1, e_2, ..., e_{|E|}$.

The encoded representation for F is bidirectional – an LSTM is run in each direction (forward and reverse), and the outputs are concatenated. For each source word j, we have

$$\overrightarrow{\boldsymbol{h}}_j^{(f)} = \text{LSTM}(embed(f_j), \overrightarrow{\boldsymbol{h}}_{j-1}^{(f)})$$

$$\overleftarrow{\boldsymbol{h}}_j^{(f)} = \text{LSTM}(embed(f_j), \overleftarrow{\boldsymbol{h}}_{j-1}^{(f)})$$

$$\boldsymbol{h}_j^{(f)} = \text{concat}[\overrightarrow{\boldsymbol{h}}_j^{(f)}; \overleftarrow{\boldsymbol{h}}_j^{(f)}]$$

In order to create a representation of the whole sentence (that is, over all time steps), the vectors are concatenated into a matrix.

$$H_{(f)} = \text{concatenate\_cols}(\boldsymbol{h}_1^{(f)}, \cdots, \boldsymbol{h}_{|F|}^{(f)})$$

During decoding, at each time step t, we have

$$\boldsymbol{h}_t^{(e)} = \text{dec}([\boldsymbol{c}_{t-1}; embed(e_{t-1})], \boldsymbol{h}_{t-1}^{(e)})$$

$\boldsymbol{h}_0^{(e)}$ is the embedding of the sentence boundary symbol `<eos>`. $\boldsymbol{c}_t$ is a vector, obtained using the attention vector $\boldsymbol{\alpha}_t$. It is initialized as $\boldsymbol{c}_0 = \boldsymbol{0}$. The attention vector has probabilities representing the relevance, or attention, of each source word in producing the next target word at decoding time step t. It is, therefore, of length $|F|$. The probabilities for predicting the next target word are obtained using an attention score between the source encoding j and the current target representation t.

$$\boldsymbol{c}_t = H_{(f)}\boldsymbol{\alpha}_t$$

$$a_{t,j} = \text{score}(\boldsymbol{h}_j^{(f)}, \boldsymbol{h}_t^{(e)})$$

$a_{t,j}$ over all j form a vector of scores $\boldsymbol{a}_t$.

$$\boldsymbol{\alpha}_t = \text{softmax}(\boldsymbol{a}_t)$$

Finally, in order to predict the next word at time step t, we have the probability vector

$$\boldsymbol{p}_t^{(e)} = \text{softmax}(W_y[\boldsymbol{c}_t; \boldsymbol{h}_t^{(e)}] + b_y)$$

2.1 Attention Score Calculation

Several methods can be used to calculate the attention score $a_{t,j}$. The one used for this implementation is a multilayer perceptron.

$$a_{t,j} = W_2 \tanh(W_1[\boldsymbol{h}_j^{(f)}; \boldsymbol{h}_t^{(e)}])$$

The attentional model was implemented with help from the class pseudocode, the class notes and the Dynet attention example [1].

[1] `https://github.com/clab/dynet/blob/master/examples/python/attention.py`

3 Modifications

The basic model is described above, but in an attempt to improve speed and translation performance, additional techniques were implemented. Several of these are inspired by nmt-tips [2].

3.1 Minibatching

I implemented minibatching for improving speed during training the model. Rather than creating even-sized batches from the training corpus, I used a suggestion by Pengcheng Yin on Piazza which greatly simplified the implementation. Batches are constrained by two conditions

- Each batch contains pairs with source sentences that all have the same length.
- Each batch can have a maximum of `batch_size` sentence pairs.

Therefore, there can be multiple batches of sentences with the same source sentence length. However, it is not necessary that all batches contain `batch_size` pairs.

These constraints make implementing the encoder very easy – the tokens are transposed and using Dynet's batch-specific functions (`lookup_batch`), the encoded representation is computed. Since all the sentences are of the same length here, we do not need to perform masking.

For the target sentences, we mask the sentences and compute the loss on the target word using `pickneglogsoftmax_batch` and `sum_batches`. The code was modularized in such a way that calculating attention did not need to change for minibatching.

Minibatching was implemented with assistance from the Dynet RNN-LM tutorial [3].

3.2 Dropout

Dropout was incorporated into the model in order to control overfitting (Srivastava et al., 2014). This was simple to add using the built-in Dynet function `set_dropout`.

[2] `https://github.com/neubig/nmt-tips`

[3] `https://github.com/clab/dynet_tutorial_examples/blob/master/tutorial_rnnlm_minibatch.py`

3.3 Unknown Word Replacement

There might be several words in the test data that do not occur in the training data. The model does not know how to handle these words. Therefore, I replaced words occurring in the training corpus (both source and target) only once with an unknown symbol, `<unk>`. The model was then trained with these sentences.

During test time, if the decoder outputs `<unk>` as the next target word, it is replaced by the word that had the highest probability in the attention vector $\boldsymbol{\alpha}_t$ in the target sentence. Therefore, no `<unk>` symbols appear in the final translation.

3.4 Early Stopping

The perplexity on the validation set is computed after roughly every 20000 training sentences. The model is written out whenever the perplexity is an improvement over the previous lowest perplexity. This is done so that even if the model begins overfitting after several epochs of training (i.e., perplexity on the train set gets lower), the model that performed best on the validation set is still stored. This is referred to as early stopping because the best model (the one we want to use) may not be the model that is trained for the most number of epochs/training batches.

3.5 Rate Decay

Every time the perplexity on the validation set becomes worse, the learning rate for the trainer is reduced. This makes the trainer update the parameters in smaller steps, thereby slowing down training and controlling overfitting.

3.6 Greedy and Beam Search

During test time, I experimented with both greedy search as well as beam search. Greedy search simply chooses the target word at time step t as the one with the highest probability. Beam search, on the other hand, allows room for exploring more hypotheses by choosing k potential target words at each time step.

The hypotheses generated by each of these k words are explored in future time steps. This is useful because greedy search chooses the local optimal value, which may not lead to the best hypothesis overall. Although it is intractable to search over all hypotheses, beam search often leads to better scoring hypotheses than greedy search, as it accounts for some future optimality.

The beam size k can vary and we experiment with several values.

4 Experiments

This section describes the dataset used, model configurations as well as the effects of the proposed modifications. All effects and tuning are performed on the validation set. We choose the model that scores the best on the validation data and run it on the test data and report results. No tuning or experiments were performed on the test set.

4.1 Dataset

The dataset for training, validation and testing is a German-English parallel corpus from the IWSLT 2016 workshop on translation. There are approximately 100000 training sentence pairs, 887 validation pairs and 1565 test sentences.

4.2 Configuration

The values detailed below remain constant through all the experiments.

- **Encoder input embedding size**: 512
- **Hidden layer size**: 512
- **Attention vector size**: 256
- **Batch size**: 32
- **Processor**: GeForce GTX TITAN X

The bidrectional encoding, therefore, outputs a vector of size 1024.

4.3 Basic Model Experiments

I first experimented with the basic model, without any of the suggested modifications (section 3), apart from minibatching. Since I was not using early stopping here, I set the number of epochs to a fixed number – 20. The model has a training set perplexity of 7.5 and a validation set perplexity of 20.7.

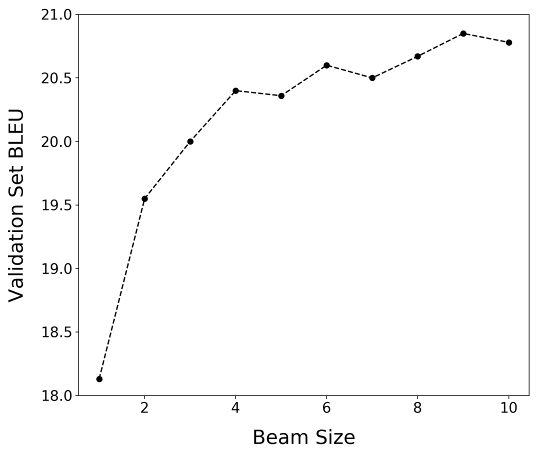

Figure 1: Validation BLEU with Beam Size

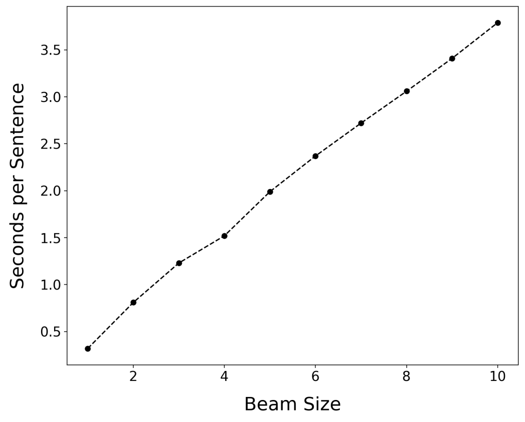

Figure 2: Time per Sentence with Beam Size

In order to identify an appropriate beam size for subsequent experiments, I used this model for decoding the validation set with beam size k varying from 1 to 10. Figure 1 shows the BLEU score and Figure 2 shows the decoding time per sentence for different k values. The decoding time at $k = 4$ is 1.5 seconds per sentence.

A sharp rise in validation set BLEU is observed from the greedy decoding $k = 1$ (BLEU 18.13) to $k = 4$ (BLEU 20.40). After this, a slow increase (with some ups-and-downs) is seen (BLEU 20.85 at $k = 9$). However, the decoding time for each sentence almost linearly increases with k.

It is evident that using beam search is far superior to greedy search in terms of BLEU. As a speed-accuracy trade-off, we choose $k = 4$ for experiments with models with further modifications and added techniques.

	Validation Set BLEU
Basic Model	20.40
+Dropout	22.21
+UnkReplace	22.87
+RepeatWord	22.89

Table 1: BLEU Improvements with Additions

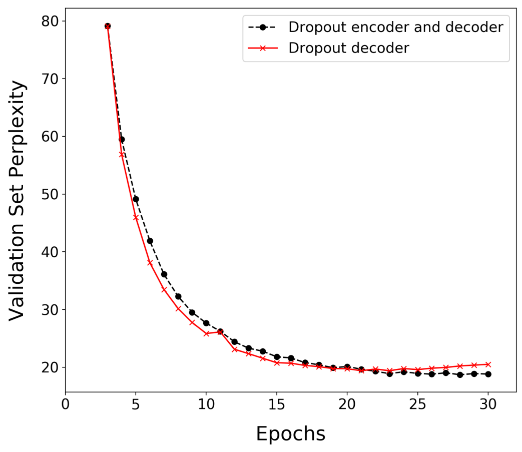

Figure 3: Effects of Dropout (Epochs 3 to 30)

4.4 Added Modifications

This subsection describes modifications that contribute positively to validation set BLEU score (Table 1). None of the tuning and experiments are performed on test data.

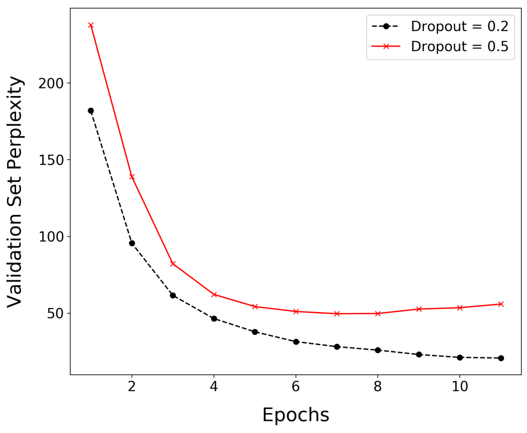

Figure 4: Different Dropout Values

4.4.1 Dropout and Early Stopping

I experimented with adding dropout at all three LSTMs (left-to-right encoder, right-to-left encoder and decoder), as well as just the decoder layer. The motivation was to use the entire encoding while producing target words and just use dropout in the decoder. As Figure 3 shows, these configurations do not show much variation in terms of perplexity on the validation set. A slow advantage of dropout on both encoder and decoder with respect to dropout while only decoding is seen around the 25th training epoch. The figure shows perplexity starting epoch 3 onwards, simply to scale the visualization such that the slight difference at the end of training is seen.

Experiments were also conducted with two different dropout values – 0.3 and the standard used in several models, 0.5. Figure 4 shows that 0.5 performs much worse in terms of validation perplexity than dropout 0.2. This experiment was conducted with dropout on both encoder-decoder. I stopped the experiment after 11 epochs, as the 0.5 dropout model had worsening perplexity (Fig. 4)

For further experiments, dropout 0.2 at all LSTMs (encoder and decoder) is used.

Table 1 shows the improvement using the chosen dropout over the basic model. Training was done for 30 epochs and the model used for computing validation set BLEU is based on the validation perplexity, i.e., *early stopping*. The model has a validation perplexity of 18.64.

4.4.2 Singleton Removal and Unknown Word Replacement

I implemented *singleton word removal* on the training corpus as described in the previous section. The trend in validation perplexity over 30 training epochs is shown in Figure 5. A rapid decrease in perplexity is seen in the first few epochs and it stabilizes as we train further.

It is observed that the perplexity is lower after adding removal of singletons, as compared to the model with only dropout. The model selected based on *early stopping* has a validation perplexity of 14.47.

After the model is trained, *unknown word replacement* is used during test time decoding. Unknown target words are replaced by the word with the

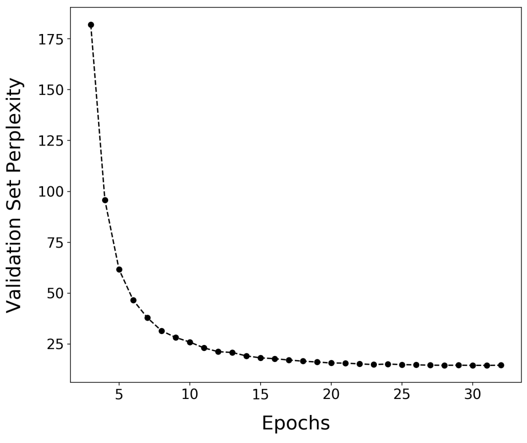

Figure 5: UnkReplacement Validation Perplexity

highest attention score for that time step. The effect on validation set BLEU score is seen in Table 1. UnkReplacement contributes reasonably to BLEU. It was noted to be particularly useful when *named entities* like person names and locations appeared in the test data.

4.4.3 Repeating Unknown Word

On observing the patterns in unknown word replacement in the validation set translations, I noticed several instances where the same source word was being repeated for consecutive `unk` targets. For example, if both the 2nd and 3rd target words were decoded as `unk`, UnkReplacement often assigned the same source word to both the 2nd and 3rd positions in the target sentence, leading to errors as source words as rarely repeated within the target.

In order to alleviate this problem, I used a simple heuristic. If two consecutive `unk` words e_t and e_{t+1} were assigned the same source word f_j, the assignment for e_{t+1} will be the next word in the source sequence, f_{j+1}. This heuristic is very simple and essentially assumes languages that follow similar word order. This heuristic slightly improves BLEU, as seen in Table 1.

4.5 Other Attempted Modifications

These are modifications experimented with, but not added, to the final model.

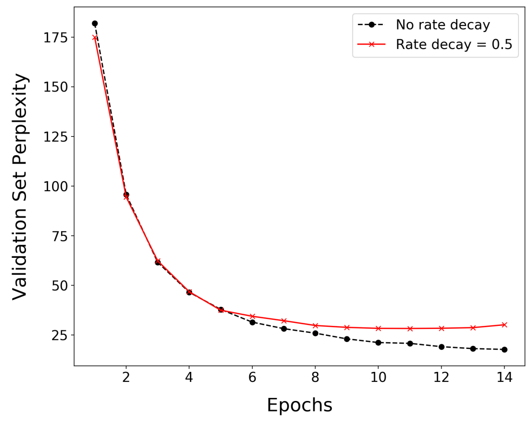

Figure 6: Rate Decay Validation Perplexity

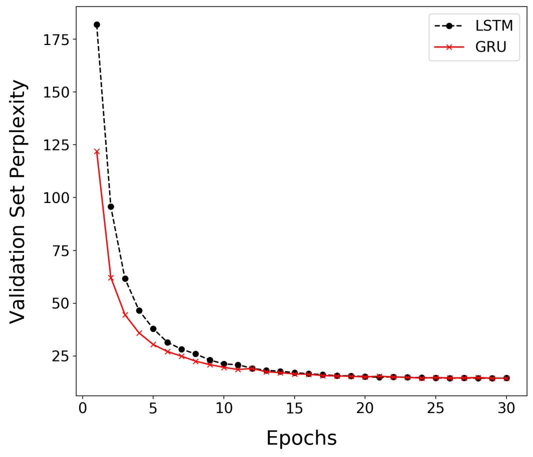

Figure 7: GRU vs LSTM Val. Perplexity

4.5.1 Rate Decay

On adding a rate decay of 0.5 when the validation perplexity worsens, it is seen that the model begins training poorly. This is likely because the rate decay was rather strict and validation perplexity was tested around 5 times per epoch, making the learning rate half every time it was observed to be worse than the previous best. Figure 6 shows how the model diverges from that with no rate decay around the 6th epoch.

With the strict and rapid rate decay that was implemented, the model likely had a very small learning rate, therefore showing worse performance than a constant learning rate.

4.5.2 GRU and LSTM

Figure 7 shows an implementation of the final model (described above), with GRUs instead of LSTMs. The validation perplexity effectively converges to very similar values. We do not change the final implementation to GRUs, as performance on BLEU is likely similar as well.

5 Final Model and Results

The model was tuned based on validation set perplexity and BLEU score. The modifications that worked favorably were incorporated into the basic model, while those that did not improve BLEU or perplexity were discarded.

The modifications added to the basic model were:

- Beam search, with beam size 4
- Early stopping based on validation perplexity
- Dropout 0.2 on encoder and decoder
- Unknown word replacement
- Heuristic for repeating unknown words

This final model is now used to translate the test and blind data sets.

The BLEU score obtained on the test set is **23.68**. Decoding took around 1.7 seconds per sentence.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.