Neural Machine Translation with Attention

Samridhi Shree Choudhary
sschoudh@cs.cmu.edu

Abstract

This document presents an implementation of an encoder-decoder model using attention. The model is used to translate German sentences to English sentences. The implementation was done using Dynet. Training was done on ˜100,000 German-English sentence pairs. The model was tested on a validation set containing ˜900 sentence pair and on a test set containing ˜1600 sentence pairs. The following sections give an overview of the experiments done along with the results obtained in the form of BLEU scores of the translations generated by the model.

1 Introduction

Neural Machine Translation Models have been shown to achieve better performance with the potential to overcome many weaknesses of the traditional statistical translation models. These models are conceptually simple with minimum requirements for the domain knowledge. The most popular of them are the encoder-decoder models. One of the major limitations of such models however is that the source sentences are encoded as fixed-length vector. Attention is a step forward in the direction to overcome such problems. Instead of learning a single fixed length representation for the input sentence, attention stores the vectors for each word in the sentence. These word-vectors are then referenced during decoding at each time step. Therefore we overcome the problem of a fixed-length representation, since the number of encoded representations is the same as the number of words in the input/source sentence. Also, each of the source word is now more directly involved in performing the translations to the target language.

2 Model Equations

Following is a brief overview of the equations involved in the process. These were the reference equations used while doing the implementation too.

In order to create an encoded representation of the source sentence, we run an LSTM/RNN in both the directions over the sentence:

$$\overrightarrow{h_j^f} = LSTM(embed(f_j), \overrightarrow{h_{j-1}^f})$$
$$\overleftarrow{h_j^f} = LSTM(embed(f_j), \overleftarrow{h_{j+1}^f})$$

These two vectors are then concatenated to form the bidirectional representation h_j^f as shown below:

$$h_j^f = [\overleftarrow{h_j^f}; \overrightarrow{h_j^f}]$$

The individual vectors can be concatenated to form an encoded matrix. This concatenations allows for efficient computation.

$$H^{(f)} = concat\_col(h_1^{(f)}, h_{|F|}^{(f)})$$

The key idea of attention is defining an vector α_t, also called the attention vector which is multiplied with the above matrix to form a vector c_t called the context vector.

$$c_t = H^{(f)} \alpha_t$$

The attention and the context are updated during decoding as shown by the equations below:

$$h_t^{(e)} = decoder([embed(e_{t-1}); c_t], h_{t-1}^{(e)})$$

The value $h_t^{(e)}$ is then used to calculate the attention vector α_t mentioned above where each entry in the vector is:

$$a_{t,j} = attn\_score(h_j^f, h_t^e)$$

$$\alpha_t = softmax(a_t)$$

3 Implementation

The implementation of the model was done using Dynet. The subsections detail the different modifications tried for the model. The above mentioned equations were used as a reference. The mini-batched implementation however, has some issue and is currently facing the problem where it is not able to learn. The code for all the modifications attempted is present in the github link. The results detailed and the translations provided are however for the model trained without mini-batching.

3.1 Hyper Parameters

The values of the hyper-parameters used for the model as as follows:

- Hidden State Size = 512
- Embeddings Size = 512
- Attention Size = 256
- Batch Size = 32
- Dropout Value = 0.2
- `multi-bleu.perl` script is used to calculate the corpus level BLEU score

3.2 Mini-Batching

Training a Neural Machine Translation model by calculating the loss for each sentence-pair in the training data consumes a large amount of time. A smart way to make the computations and hence the training faster is mini-batching. The training data is divided into mini-batches of a certain size. Instead of calculating the loss at each time step of the decode function, the loss is now calculated for a batch of sentences. The learning might become slow but the processing is much faster.

3.3 Drop-out

In order to improve training, Some of the hidden neurons in the hidden layers of the neural network are randomly dropped during the forward computation. The dropout value used for the implementation was 0.2

3.4 Singleton Removal and $< UNK >$

In order to enable the model to handle the words it has not seen in the training data, we replace some of the words in the training data with a special symbol $< unk >$. While processing the data, we maintain a count of the words and replace all the words that occur just once in the training set with $< unk >$. This symbol is used when we encounter an unknown word in the testing data.

4 Results

Training the model while calculating the loss for each of the sentences sequentially took a long time. The training sentences were reshuffled randomly in each epoch. The model was trained for 15 epochs. The best BLEU score was observed at the end of 10th epoch after processing 80,000 samples, after which the BLEU score started to reduce due to over-fitting. The BLEU scores were generated separately from the training implementation. The generation was done using greedy search. Following is the summary of the results obtained for the model trained without mini-batching.

- BLEU on validation set, greedy search = **14.33, (48.3/21.4/10.3/5.3)**
- BLEU on test set, greedy search = **15.8, (48.9/22.5/11.6/6.3)**

5 Issues Faced

- Debugging Dynet was a bit of a challenge since the python bindings did not have proper debug statements.
- I hoped to report better BLEU scores if the mini-batching implementation along with the modifications of dropout and `unk` was working on time. The translations suggest that the model is not learning the representations properly. I hope to debug the issue and check the translations accuracy for this model.
- One other issue faced was that training on CPU was very slow. There were some initial issues with GPU and the Dynet binding for GPU (_gdynet) and hence the setup took some time. The installation instructions outlined in the tutorial miss a parameter (−`DCUDA_TOOLKIT_ROOT_DIR=/usr/local/cuda-x.x` while building the GPU version of Dynet (`DBACKEND=cuda`).

6 Conclusion

Neural Machine Translations models have achieved state of the art performances. The attention model implemented here learned good representations for translating German sentences to English. While BLEU does not take into account the semantics of the translated sentences, manually looking at some of the low scoring BLEU sentences seem to make similar sense to the reference translations. It would be interesting to see how different evaluation measures are used in translation models which capture this semantic sense of the generated sentences.

References

`http://dynet.readthedocs.io/en/latest/index.html`

`https://github.com/neubig/nmt-tips`

`https://github.com/clab/dynet/tree/master/examples/python`

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *"Neural machine translation by jointly learning to align and translate."* In Proceedings of the International Conference on Learning Representations (ICLR), 2015.

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