ASSIGNMENT 1

Shrimai Prabhumoye
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213
sprabhum@andrew.cmu.edu

1 Introduction

Machine translation means to translate text or speech from one language to another. In the recent years, neural models are used to perform the task of machine translation. In this assignment we will be using an encoder-decoder framework with global attention as described in [2]. Figure 1 [2] shows the computational graph for translation of French (denoted by f) to English (denoted by E). In this assignment, we will do translation of German to English. We will be using data from IWSLT2016 workshop. We are using data tokenized using the tokenizer from NLTK (.tok), and this tokenized data was further lowercased (.low). In addition, a filtered set with sentences with at least one and no more than 50 words was provided (.tok.filt and .low.filt) to us. We will be using those files for our training and experiments.

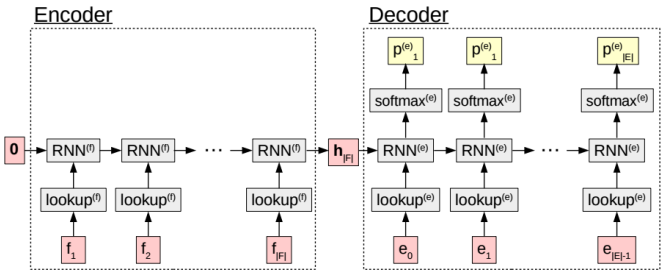

Figure 1: Encoder Decoder for Machine Translation

2 Model

For this assignment, we using bi-directional LSTM for encoder. We first create an embedding vector for each word in the source sentence (in our case German). The embedding size is the same for all the words. We then run RNN for all the words in the source sentence in the left-right and right-left directions. Once we get the forward ($\overrightarrow{h}_j^{(f)}$) and the backward ($\overleftarrow{h}_j^{(f)}$) representations, we concatenate them into a bidirectional representation $\mathbf{h}_j^{(f)}$. For ease of calculations, we then concatenate all the columns of the bidirectional encoder vectors into a matrix. Here, $|F|$ is the length of the source sequence.

$$
\begin{aligned}
\overrightarrow{h}_j^{(f)} &= RNN(embed(f_j), \overrightarrow{h}_{j-1}^{(f)}) \\
\overleftarrow{h}_j^{(f)} &= RNN(embed(f_j), \overleftarrow{h}_{j+1}^{(f)}) \\
h_j^{(f)} &= [\overrightarrow{h}_j^{(f)}; \overleftarrow{h}_j^{(f)}] \\
H^{(f)} &= concat\_col(h_1^{(f)}, ..., h_{|F|}^{(f)})
\end{aligned}
$$

We then calculate the attention vector α_t in the following manner:

$$
\begin{aligned}
\alpha_t &= softmax(a_t) \\
where, a_{t,j} &= attn\_score(h_j^{(f)}, h_t^{(e)}) \\
where, h_t^{(e)} &= decoder\_RNN([embed(e_{t-1}); c_{t-1}], h_{t-1}^{(e)}) \\
where, c_{t-1} &= H^{(f)}\alpha_{t-1}
\end{aligned}
$$

We then get probability distribution of each word in the decoder vocabulary (in our case English) in the following manner:

$$
p_t^{(e)} = softmax(W_{hs}[h_t^{(e)}; c_t] + b_s)
$$

The above equations are all given in [3]. The Figure. 2 is given in [3] and it explains the computation graph of the attention model for german to english machine translation.

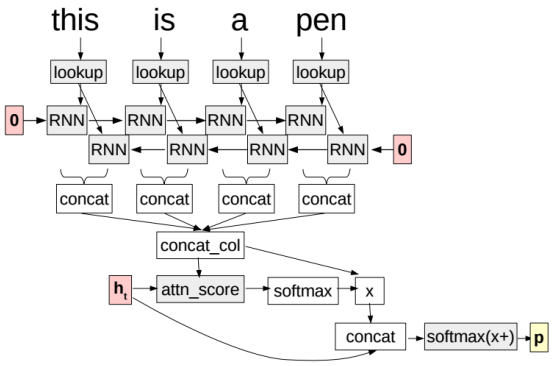

Figure 2: Attention model for Machine Translation

3 Experiments and Results

We have not restricted our vocabulary of German or English. Hence the vocabulary size of German is 83475 and vocabulary size of English is 41590. We have done mini batching with batch size of 32. I tried beam search but it was extremely slow on CPU because of which, I have excluded it from my submission. Hence, the extensions we have tried are:

- Mini Batching
- Dropout

Due to computational constraints and the speed of training, I could only complete training upto a certain epoch. I have mentioned the epoch number for better understanding of results.

Configurations	Validation BLEU Score	Test BLEU Score	Dropout Value
Embedding Size = 500 Hidden Size = 500 Attention Size = 200 Epoch = 21 Name: Model 1	Avg: **17.43** Unigram: 51.3 Bigram 24.3: Trigram: 12.8 4-gram: 6.9	Avg: **18.65** Unigram:52.1 Bigram: 25.2 Trigram: 13.6 4-gram: 7.6	0
Embedding Size = 500 Hidden Size = 500 Attention Size = 200 Epoch = 15 Name: Model 2	Avg: **16.42** Unigram: 51.1 Bigram 23.2: Trigram: 11.7 4-gram: 6.3	Avg: **17.83** Unigram:51.9 Bigram: 24.2 Trigram: 13.1 4-gram: 7.1	0.2
Embedding Size = 500 Hidden Size = 500 Attention Size = 200 Epoch = 15 Name: Model 1	Avg: 16.31 Unigram: 48.3 Bigram: 21.9 Trigram: 11.2 4-gram: 6.0	Avg: 16.99 Unigram: 49.3 Bigram: 22.8 Trigram: 11.8 4-gram: 6.3	0.0
Embedding Size = 1000 Hidden Size = 1000 Attention Size = 350 Epoch = 15 Name: Model 3	Avg: 15.78 Unigram: 50.9 Bigram: 22.9 Trigram: 11.5 4-gram: 5.9	Avg: 17.56 Unigram: 51.4 Bigram: 24.2 Trigram: 12.8 4-gram: 6.8	0.0

For generation process, I tried restricting the length of the output sequence from $input\_sequence\_length * 2$ to $input\_sequence\_length$. But this did not have any effect on my generation.

Note that, I have submitted Model 1 generations as my primary results. I have also submitted Model 3 generated sentences for epoch 20 as additional files marked with '.model3.' .

4 Discussion

The first model in the above table i.e Model 1 performed the best because, I could train the model for 30 epochs. After the 21st epoch, the model was overfitting and this was checked using validation perplexity. The BLEU score was the best for this epoch as reported in the table.

Unfortunately, I could not run the dropout model i.e Model 2 for more than 15 epochs. For a fair comparison, I have also shown the BLEU scores for Model 1 at 15th epoch. This comparison tells us that dropout learns much faster.

I have also shown comparison with the model having higher configuration of hidden and embedding size of 1000 with no dropout i.e Model 3. This model is much slower to train but it learns good representations. As we can see from the table, this Model 3 performs much better than Model 1 at epoch 15 but it is still not as good as Model 2 at epch 15.

Error Analysis: When I was going through the generated hypothesis of the models, I noticed that some of the hypothesis had the same meaning as that of the gold sentences. The semantics of the sentence is not captured by BLEU, and hence it would give low BLEU score for such sentences.

Hypothesis: one language is an expression of the human mind .
Gold Sentences: a language is a flash of the human spirit .

Hypothesis: you see , we 're going to see the first time in the web as if it 's really a network , not just from the side , but much more .
Gold Sentences: so you see , we 're navigating the web for the first time as if it 's actually a web , not from page-to-page , but at a higher level of abstraction .

Mini batching helped in speeding up the training of the models. It was very hard to implement mini batching in dynet. One of the main reasons for this was that dynet gives C errors and does not tell the line number on which the program is failing. Hence, I had to manually write debug statements at each candidate point of failure and this was very time consuming. Also, it is not straight forward to find the size of a dynet vector when it throws dimension mismatch error. This makes it hard to debug.

5 Acknowledgements

I have referred to the tutorial code given in [1]. I have also discussed the understanding of attention mode and especially mini batching with Dheeraj Rajagopal, Samridhi Shree and Varsha Embar.

References

[1] Dynet attention tutorial. https://github.com/clab/dynet/blob/master/examples/python/attention.py. Accessed: 2017-02-25.

[2] Neural mt 1: Encoder-decoder models. http://phontron.com/class/mtandseq2seq2017/mt-spring2017.chapter7.pdf. Accessed: 2017-02-25.

[3] Neural mt 2: Attentional models. http://phontron.com/class/mtandseq2seq2017/mt-spring2017.chapter8.pdf. Accessed: 2017-02-25.