11-731 Machine Translation and Sequence to Sequence Models Assignment 1

Varsha Embar

***Abstract*— The goal of this assignment was to develop a neural attention model for machine translation. The model described in this report was trained to translate from German to English. Dynet, a neural network toolkit was used to build this model. The proposed encoder decoder model with attention and greedy decoding with controlled window size performs with a BLEU score of 17.79 at the end of 14 epochs of training.**

I. INTRODUCTION

Neural Machine Translation (NMT) is an approach to Machine Translation by training neural networks using deep learning techniques. These models are trained end-to-end to maximize translation performance. Sutskever et al have successfully applied basic sequence to sequence models for this task.

A basic sequence-to-sequence model consists of two recurrent neural networks (RNN) such as Long-Short Term Memory Networks (LSTMs): an encoder and a decoder. The encoder encodes a sequence of words or characters in a source language into a fixed length vector representation and then the decoder decodes from that representation using another RNN in the target language.

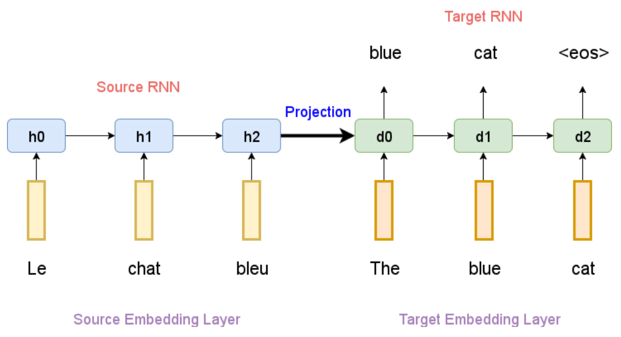

Fig. 1. A basic encoder-decoder model for Machine Translation

II. ATTENTION MODEL

An extension to this model is the encoder-decoder model with attention. Bahdanau et al extended sequence to sequence models that incorporate an attention mechanism that uses information from the RNN hidden states in the source language at each time step in the decoder RNN. This attention mechanism significantly improves performance on tasks like machine translation.

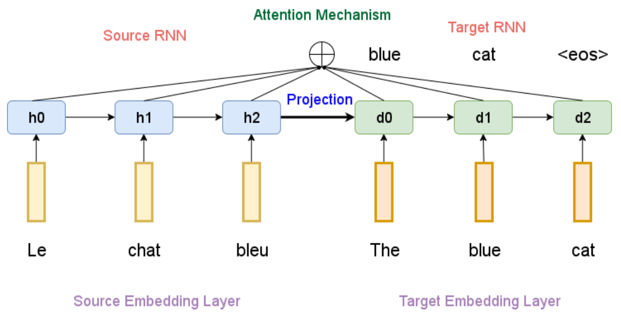

Fig. 2. A encoder-decoder model with attention

In the experiments described in this report, the model used is described in the following subsections. The model was built in dynet using the code at https://github.com/clab/dynet/blob/master/examples/python/attention.py as reference.

A. Encoder

Given the source language sentence, we calculate a representation for every word using a bi-directional LSTM.

$$\overrightarrow{h}_j^{(s)} = LSTM(s_j, \overrightarrow{h}_{j-1}^{(s)})$$
$$\overleftarrow{h}_j^{(s)} = LSTM(s_j, \overleftarrow{h}_{j+1}^{(s)})$$

The final representation of the source vector is given by the concatenation of these two vectors:

$$h_j^{(s)} = [\overleftarrow{h}_j^{(s)}, \overrightarrow{h}_j^{(s)}]$$

B. Attention

We calculate a context vector c_t using the attention score vector α_t and the encoded representation. The attention vector tells us the relevance of the source word at each step of decoding. The equation to calculate the context vector is given by

$$H^{(s)} = ConcatenateColumns(h_1^(s), ...h_s^{(s)})$$
$$c_t = H^{(s)} \alpha_t$$

C. Decoder

We update the hidden representation at the decoder based on the representation of the word and the context vector of the previous time step. We then calculate the attention score using the source encoding and the updated hidden representation. Normalization is done by taking a softmax over the attention scores to get α_t.

$$h_t^{(d)} = LSTM([embed(d_{t-1}); c_{t-1}], h_{t-1}^{(d)})$$
$$\alpha_t = softmax(AttentionScore(h_j^{(s)}, h_t^{(d)}))$$

Our attention score is calculated as follows:

$$AttentionScore(h_j^{(s)}, h_t^{(d)}) =$$
$$w_{a2}^T tanh(W_{a1}[h_t^{(d)}; h_j^{(s)}])$$

Now, given the context vector and the hidden representation, we predict the next word by performing a softmax over the affine transformation of the concatenation of the two vectors.

$$p_t^{(d)} = softmax(W[h_t^{(d)}; c_t] + b)$$

III. Experiments

The following experiments were performed for the training: mini-batching, for generation: greedy search and greedy search with different window sizes.

The dataset consists of IWSLT data, a parallel translation corpus from German to English. The model was trained with 98132 samples. The validation set consists of 887 samples and the test set consists of 1565 examples.

The model parameters are as follows: the embedding size was set to 500 and the the hidden layer size to 500. The attention vector size was set to 200. During generation, the window size was set to the length of the source sentence. The BLEU scores at different epochs are shown in Table 1. At the time of writing the report, the model had trained for 14 epochs.

Epochs	Validation	Test
5	9.95	10.72
10	14.09	15.47
14	15.94	17.52

TABLE I
Evaluation at different epochs

As we can see, the model performs significantly well as the training progresses. Experiments on different window sizes while generation were also performed. Table 2 shows the changes in the BLEU scores based on unigram, bigram, trigram, 4-gram and the average. slen represents the length of the source sentence. An additional window size of 5 more than the length of the source sentence seems to perform the best.

Length	1-gram	2-gram	3-gram	4-gram	BLEU
slen	51.5	24.3	12.7	6.9	17.52
2 x slen	50.8	23.6	12.3	6.6	17.65
slen - 5	53.2	25.2	13.1	7.1	14.27
slen + 5	51.0	23.8	12.4	6.7	17.79
slen + 10	50.8	23.6	12.3	6.6	17.68

TABLE II
Translation with different window sizes

IV. Difficulties Faced

Perhaps the most time consuming part of the assignment for me was the mini batching. Mini batching using dynet was not very straight forward. Not being able to print the shape of the intermediate variables added to the difficulty. Furthermore, installing dynet for a GPU machine was interlaced with several issues.

Training on CPU for an epoch took about 6 hours for 1 epoch as compared to approximately 1 hour 30 minutes on a GPU.

ACKNOWLEDGMENT

I had general discussions on implementation in Dynet with Shrimai, Dheeraj and Samridhi. I thank them for their help; especially Shrimai - for lending her server to run experiments.

REFERENCES

- Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. ”Sequence to sequence learning with neural networks.” Advances in neural information processing systems. 2014.
- https://www.tensorflow.org/tutorials/seq2seq
- http://www.phontron.com/class/mtandseq2seq2017/mt-spring2017.chapter8.pdf
- https://github.com/MaximumEntropy/Seq2Seq-PyTorch
- https://github.com/clab/dynet/blob/master/examples/python/attention.py
- Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. ”Neural machine translation by jointly learning to align and translate.” arXiv preprint arXiv:1409.0473 (2014).
- 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. ”DyNet: The Dynamic Neural Network Toolkit.” arXiv preprint arXiv:1701.03980. 2017.