# 11731 Machine Translation Homework-1

Dheeraj Rajagopal (drajagop) github: dheeraj091

February 25, 2017

**Abstract**

## 1 Introduction

Current state-of-the-art performances in Machine Translation are achieved by Neural Machine Translation(NMT) systems. These NMT systems usually have an Encoder-Decoder approach, which uses an RNN language model for source and destination. A source "encoder" uses a function $F$ to encode the information as a real-valued vector and uses this information to predict the target sentence.

Current improvements in the Sequence-to-Sequence Models is to include multiple RNN layers to get the hidden representation and "attention", where we learn additional parameters, that encodes the similarity information between individual words in the source to individual words in the target sentence.

Our model uses a bi-directional LSTM network with MLP attention for the task of German to English translation of IWSLT corpus. We use a Titan X GPU to run our experiments and to speed up the time for experiments, we implemented mini-batching and we observed about approximately 3x-5x improvements in speed. We got a best BLEU score of 15.33 in our experiments.

## 2 Model

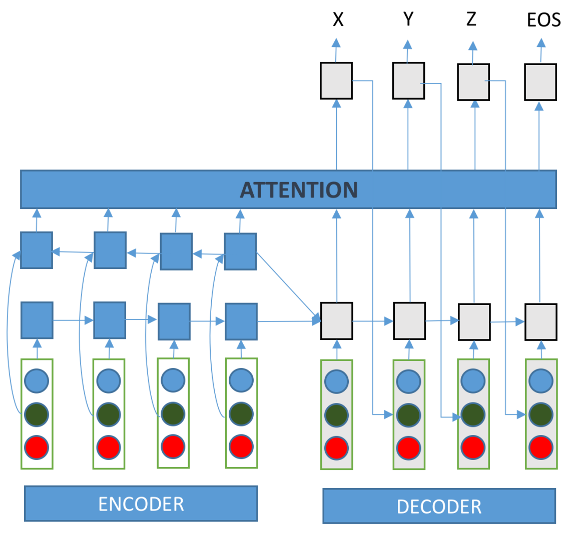

Figure 1: Proposed Model

The proposed model is shown in figure 1.

**Encoder**: We use a bidirectional LSTM Encoder for our model. Suppose the input source sentence $x$ is represented words $w_1^s, w_2^s, ..w_n^s$ and each of the LSTM hidden state is represented as $\vec{h}_{i=1..n}$.

$$\vec{h}_i = LSTM(embed(f_j), \vec{h}^{(f)}_{j-1}) \quad (1)$$

$$\overleftarrow{h}_i = LSTM(embed(f_j), \overleftarrow{h}^{(f)}_{j+1}) \quad (2)$$

We use the concatenation of both of the above vectors to represent the hidden state $h^{(f)}_j = [\vec{h}^{(f)}_j; \overleftarrow{h}^{(f)}_j]$ and we further concatenate them to a matrix $H*(f) = [h^{(f)}_i, ...., h^{(f)}_n]$

**Context Vector :** We then calculate the context vector $c_t = H^{(f)}\alpha_t$

**Attention :** In our model, we use a variation of the Multilayer Perceptron [BCB14] to calculate the attention matrix. Instead of calculating the attention over the hidden layer of the decoder, we use all the states in the two-layer LSTM weights to calculate our attention.

**Decoder**

For the decoding part, we use the softmax function and the categorical cross entropy error to get the loss. This loss is then backpropagated for the network to tune the weights.

### 2.1 Extensions Implemented

For our experiments, we also implemented mini-batching and dropout. Although, we notice 3x-5x speedup in the runtime of each epoch, the network did not learn substantially.

## 3 Experiments

1. We used the full vocabulary to train our model
2. **Hyperparameters :** embedding size = 512, hidden state size = 512, attention size = 256, batch size = 32, dropout = 0.2
3. We used the SGD optimization (Adam Optimizer overfit the data very fast and often converges to a comparatively bad local minima).
4. We use the BLEU metric to evaluate our results
5. At the end, we use the greedy decoding to select the sentence candidate.

We achieved a BLEU score of 15.33 on the test set, without mini-batching after 12 epochs. We were unable to debug the mini-batched model on time to run the model to get reasonable output. The output from the mini-batched output indicates that the model hasn't learned to generate a reasonable representation. We are still in the process of analyzing the code for bugs.

## 4 Implementation Issues

Some of the issues we faced when using Dynet. We wanted to indicate the problems so that it could be rectified.

1. In the installation instructions for GPU, it was required to add an additional parameter -DCUDA_TOOLKIT_ROOT_DIR=/usr/local/cuda-x.x to compile it correctly. Not sure if its a common issue.
2. The errors were not very friendly to non-C++ programmers.
3. It would have been easier to debug if the errors mentioned what line in the python code they came from. There was significant delay in debugging code because of this.

## References

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.