Report – ass 4

Eilon Bashari 308576933 **Daniel Greenspan 308243948**

We chose to implement the method described in the paper:

"Recurrent Neural Network-Based Sentence Encoder with Gated Attention for Natural Language Inference"

By: Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, Diana Inkpen.

Link to the paper: https://arxiv.org/pdf/1708.01353.pdf

We decided to choose this paper and implement it because we noticed that the method is using a stacked BiLSTM which interested us after implementing the single BiLSTM in the previous assignment how can it be done in a stacked formation. Furthermore, we saw that this model uses 12m parameters which is reasonable comparing to other models.

The Method

The method presented in the paper composed of the following major components: word embedding, sequence encoder, composition layer, and the top layer classifier:

Word Embedding:

The method for dealing with word embedding in the paper is to concatenate embedding vectors learned at two different levels to represent each word in the sentence: the character composition and holistic word-level embedding.

The character composition feeds all characters of each word into a convolutional neural network (CNN) with max-pooling.

For the holistic word-level embedding we use pre-trained GloVe-6B-300D vectors for each word as holistic word-level embedding (in the paper they use GloVe-840B-300D, we change that due to low capabilities of our computers, and still it was too much for them).

The concatenation of the character-composition vector and word-level embedding results in a word: $e = ([c_1; w_1], \dots, [c_l; w_l])$.

Sequence Encoder:

In order to represent words and their context in a premise and hypothesis, sentence pairs are fed into sentence encoders which is a stacked shortcut BiLSTM to obtain hidden vectors (h p and h h). The authors used shortcut connections, which concatenate word embeddings and input hidden states at each layer in the stacked BiLSTM except for the bottom layer (input layer). We get as a result:

$$h^p = \text{BiLSTM}(e^p) \in \mathbb{R}^{n \times 2d}$$
$$h^h = \text{BiLSTM}(e^h) \in \mathbb{R}^{m \times 2d}$$

(where d is the dimension of hidden states of LSTMs)

As we know BiLSTM output is a concatenation of the result of unidirectional LTSM. Hidden states of unidirectional LSTM ($\overrightarrow{h_t}$ or $\overleftarrow{h_t}$) are calculated as follows:

$$\begin{bmatrix} i_t \\ f_t \\ u_t \\ o_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \tanh \\ \sigma \end{bmatrix} (W x_t + U h_{t-1} + b)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot u_t$$
$$h_t = o_t \odot \tanh(c_t)$$

(where σ is the sigmoid function, is the elementwise multiplication of two vectors, and W ∈ R $4d \times w_d$, U ∈R^{4dxd}, b ∈ R^{4dx1} are weight matrices to be learned)

BiLSTM calculation:

For each input vector x_t at time step t, LSTM applies a set of gating functions the input gate it, forget gate f_t, and output gate o_t, together with a memory cell c_t, to control message flow and track long-distance information and generate a hidden state h_t at each time step to be forwarded to next step.

Composition Layer:

The paper proposes an intra-sentence gated-attention to obtain a fixed-length vector which could be fed to the classification layer. Illustrated by the case of hidden states of premise h p

$$v_g^p = \sum_{t=1}^{n} \frac{\|i_t\|_2}{\sum_{j=1}^{n} \|i_j\|_2} h_t^p$$

$$\text{or } v_g^p = \sum_{t=1}^{n} \frac{\|1 - f_t\|_2}{\sum_{j=1}^{n} \|1 - f_j\|_2} h_t^p$$

$$\text{or } v_g^p = \sum_{t=1}^{n} \frac{\|o_t\|_2}{\sum_{j=1}^{n} \|o_j\|_2} h_t^p$$

(We chose to use the third option)

(where i_t, f_t, o_t are the input gate, forget gate, and output gate in the BiLSTM of the top layer. Note that the gates are concatenated by forward and backward LSTM: $i_t = [\overrightarrow{i_t}; \overleftarrow{i_t}]$, $f_t = [\overrightarrow{f_t}; \overleftarrow{f_t}]$, $o_t = [\overrightarrow{o_t}; \overleftarrow{o_t}]$. $k * k_2$ indicates l_2 norm, which converts vectors to scalars).

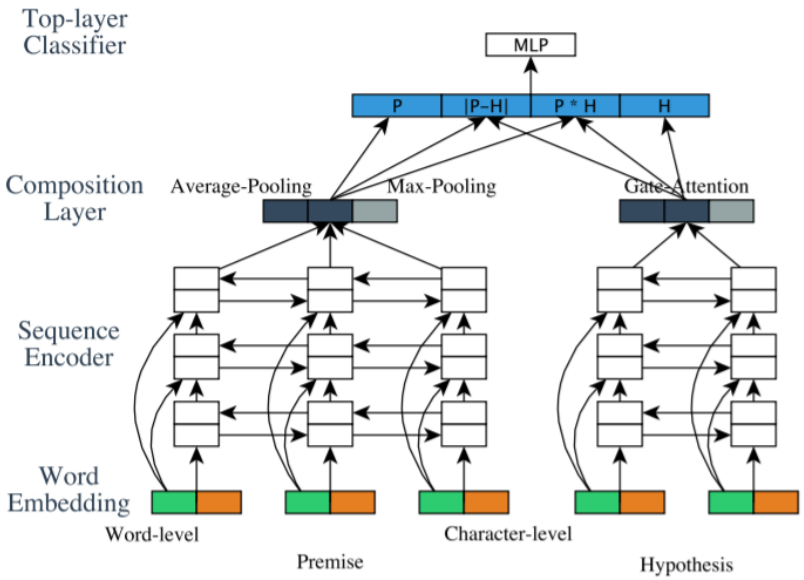
Top Level Classifier:

In the models presented in the article, the authors compute the absolute difference and the element-wise product for the tuple $[v^p, v^h]$. The absolute difference and element-wise product are then concatenated with the original vectors v^p and v^h in the following way:

$$v_{\text{inp}} = [v^p; v^h; |v^p - v^h|; v^p \odot v^h]$$

v_{inp} will be the input for our MLP classifier which uses two hidden layers with ReLU activation function and a SoftMax function for the final output.

An overview of the System described above:



Results:

The paper report that on the SNLI problem the model achieved an accuracy of 85.5%, which is best result reported on SNLI, outperforming all previous models when cross-sentence attention is not allowed. We can see in the next table the result for each method on the SNLI problem.

Model	Test
LSTM (Bowman et al., 2015)	80.6
GRU (Vendrov et al., 2015)	81.4
Tree CNN (Mou et al., 2016)	82.1
SPINN-PI (Bowman et al., 2016)	83.2
NTI (Munkhdalai and Yu, 2016b)	83.4
Intra-Att BiLSTM (Liu et al., 2016)	84.2
Self-Att BiLSTM (Lin et al., 2017)	84.2
NSE (Munkhdalai and Yu, 2016a)	84.6
Gated-Att BiLSTM	85.5

Our Attempt:

Unfortunately, we didn't manage to achieve the same results as in the paper.

First, we tried to implement the paper's method in a very 'naive' way (using only one BiLSTM rather than a stacked BiLSTM and not using much helpful code libraries) and run on a single input each time. Due to the fact that the implementation was very simple a run took a lot of time (10+ hours and finished only 15% of the training data) and the model achieved an accuracy of 22% which was reasonable considering it didn't even finish 15% of a single epoch.

Then, we improved out code by using Pytorch and dived into its library functionalities to optimize our code as much as possible. We then added the stacked BiLSTM which caused us a lot of troubles until we got it under control.

After a week or so of bug fixing, we ran our model again on our computers, but it took them eternity and we could not see the results (we both don't have GPU).

We also tried to use the Google Cloud platform and we ordered a machine on cloud, but it was not good enough for our computing and it stopped after the 11th epoch.

There were three main issues we faced while building our model:

1. The biggest issue is the problem not having sufficient means to compute this problem on our laptops. We tried to overcome this problem by using the Google Cloud VM where you can use a much stronger computer services and run python on but still it was not enough.
2. The second issue was trying to optimize our model to run with a stacked BiLSTM. At first we tried to use the arg.layers property of BiLSTM to set the number of BiLSTM layers to 3 which presented much problems with the dropout. We decided to calculate each BiLSTM separately with dropouts = [0.1,0.1,0.0] which solved our problem.
3. At first, we used only word-level embedding for the premise and the hypothesis and treated them the same for our first try. When we tried to concatenate the word-level embedding and the character-level embedding for the premise only (as described in the paper) it caused us a lot of problems since the sizes now did not match. Eventually we managed to concatenate the two types of embeddings to create our embedded premise representation.

As mentioned the run of this program took too much for out laptops and the machine we ordered from the google cloud platform hence we could not see the final results.

We were thinking about how we can improve this algorithm:

1. we noticed that in each of the bi-lstms we concatenate the input with the output of the previous bi-lstm, as we thought that maybe it could be better to skip the middle bi-lstm and let it run without concatenating the first bilstm output and the input to it. We though that in that way we could save some time for the algorithm.

2. as we mentioned above, we at first used the parameter num_layers of the LSTM class in pytorch. This could do the 3 layers of lstm by itself instead of creating 3 instances of the bi-LSTM.

Our parameters:

1) 1st BiLSTM:
   i) input dim: 300
   ii) Output dim: 600
   iii) Dropout: 0.1

2) 2nd BiLSTM:
   i) input dim: 900
   ii) Output dim: 600
   iii) Dropout: 0.1

3) 3rd BiLSTM:
   i) input dim: 900
   ii) Output dim: 600
   iii) Dropout: 0.0

4) Composition layer:
   i) input dim: 900 +900
   ii) Output dim: 1800

5) Final concatenation layer:
   i) input dim: 1800 + 1800
   ii) Output dim: 7200

6) MLP layer:
   i) input dim: 7200
   ii) Hidden layer1: 3000
   iii) Hidden layer2: 1000
   iv) Output dim: 3

7) EPOCHES:
   i) 20

8) Validation rate:
   i) 2000