Ass 4 - Report

Dorin Keshales - 313298424
Elad Ben Zaken - 203519731

- **Which paper you chose the implement.**

  The paper I chose to implement from the leaderboard is '600D Residual stacked encoders' by Yixin Nie and Mohit Bansal that was published in November 2017.
  Link to the paper: https://arxiv.org/pdf/1708.02312.pdf

- **Why you chose that particular one.**

  What first got my attention when I passed over the papers in the leaderboard is the use of residual connections to the input of each biLSTM layer, I was intrigued to explore it's impact on the learning process comparing to a typical stacked biLSTM structure, in which the input of the next biLSTM layer is simply the output sequence of the previous biLSTM layer. Additionally, the paper mentions the use of max pooling (on the output of the last biLSTM layer), which we mentioned in class in a context of convolutional networks and I'm familiar with from the computer vision field as learned and was practiced in Machine Learning course. So, I thought it will be a good opportunity to see its effectiveness on text convolutions as well. Moreover, the paper uses an interesting technique in which it combines the vector representations of both the premise and the hypothesis to extract relations between them. As following that, we get a new vector, that captures information from both premise and hypothesis, which we feed into the MLP in order to predict the relationship between the two sentences. This technique got me interested and it made me want exploring its effectiveness for this task.

- **What method was used in the paper?**

  The paper follows the approach of encoding-based models and propose a simple sequential sentence encoder for both SLNI and Multi-NLI problems.

  The model consists of two separate components, a sentence encoder and an entailment classifier. The sentence encoder encodes each of the two sentences -premise and hypothesis - into a vector representation and then the classifier makes a three-way classification based on the representation vectors combination to label the relationship between the premise and the hypothesis as that of entailment, contradiction, or neural.

Now, I will describe each of those components:

****

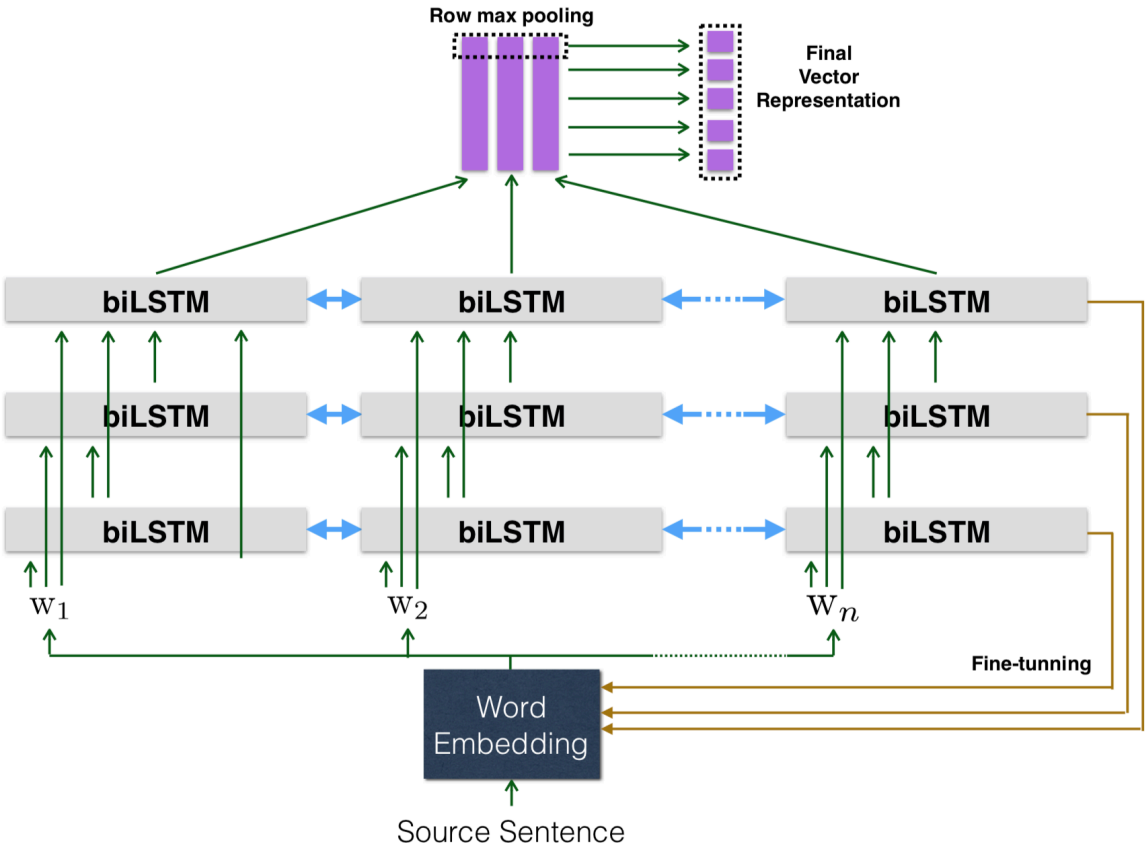
I'll first mention that the paper is mainly focused on presenting a model that has shortcut connections. But, at the last chapter of the paper (6) it was mentioned that in later experiments, they found that a residual connection can achieve similar accuracies with fewer number of parameters, compared to a shortcut connection. They report their results on SNLI, for the fewer-parameter Residual-Stacked Encoder model. Those results are also the ones appear in the leaderboard, next to this paper. Therefore, I tried to replicate the model that uses residual connections ,on which I also base my report.

****

<u>The sentence encoder:</u>

The sentence encoder is composed of multiple stacked bidirectional LSTM layers with residual connections (feeding the summation of all the previous layers' output sequences as well as the word embedding sequence to each layer) followed by a max pooling layer. The word embeddings are fine-tuned as well.

The encoder encodes each sentence into a fixed length vector without any information or function based on the other sentence.



* It's worth mentioning that the same encoder is applied to both the premise and the hypothesis with shared parameters projecting them into the same space.

Let $bilstm^i$ represent the ith biLSTM layer, which is defined as:

$$(1) \qquad \boldsymbol{h_t^i = bilstm^i\left(x_t^i, t\right),\ \forall t \in [1, 2, \ldots, n]}$$

where h_t^i is the output of the ith biLSTM layer at time t over input sequence $(x_1^i, x_2^i, \ldots, x_n^i)$.

The input sequences for the ith biLSTM layer is the concatenation of the original word embedding sequence and the summation of outputs of all the previous layers .

Let $W = (w_1, w_2, \ldots, w_n)$ represent the words in the source sentence. We assume w_i is a word embedding. Then, the input of the ith biLSTM layer at time t is defined as:

$$(2) \qquad \boldsymbol{x_t^1 = w_t}$$

$$(3) \qquad \boldsymbol{x_t^i = [w_t, h_t^{i-1} + h_t^{i-2} + \ldots + h_t^1]} \qquad \boldsymbol{(for\ i > 1)}$$

where [ ] represents vector concatenation.

Then, assuming we have m layers of biLSTM, the final vector representation will be obtained by applying row-max-pool over the output of the last biLSTM layer.

The final layer is defined as:

$$(4) \qquad \boldsymbol{H^m = (h_1^m, h_2^m, \ldots, h_n^m)}$$

$$(5) \qquad \boldsymbol{v = max(H^m)}$$

where v is the final vector representation for the source sentence.

The Entailment Classifier

Once the vector representations for both premise v_p and hypothesis v_h are generated, 3 matching methods are applied to extract relations between v_p and v_h :

(i) concatenation of the two representations (v_p, v_h)

(ii) element-wise distance | $v_p - v_h$ |

(iii) element-wise product $v_p * v_h$

The matching vector is then defined as:

$$(6) \qquad \boldsymbol{m = [v_p, v_h, |v_p - v_h|, v_p \otimes v_h]}$$

This final concatenated result m, which captures information from both the premise and the hypothesis, is fed into an MLP layer culminating in a Softmax layer to make final classification.

- **What was the result reported in the paper.**

  The paper used the following parameter settings for the model based on the residual stacked encoder:

  - Pre-trained 300D Glove 840B vectors to initialize the word embeddings .
  - Cross Entropy loss.
  - Adam-based optimization.
  - Batch size of 32.
  - Starting learning rate of 0.0002 with half decay every two epochs.
  - The encoder composed of 3 stacked biLSTM layers each with hidden dim of 600.
  - The MLP has one hidden layer.
  - The hidden layer size – 800.
  - Activation function – ReLU.
  - Dropout layer applied on the output of the MLP layer, with dropout rate set to 0.1.
  - The model was trained with the word embeddings being fine-tuned.

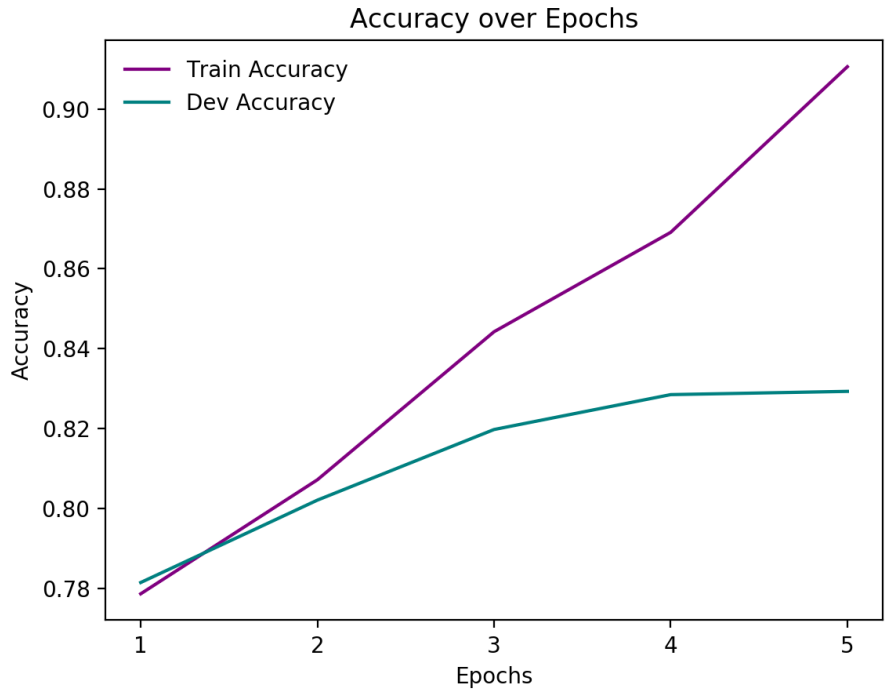
  In the paper, the reported results were 91% accuracy on the training set, 87% accuracy on the dev set and 86% accuracy on the test set.

My Attempt

- **Did your code manage to replicate this result?**

  Unfortunately, despite all my efforts (that will be detailed next page) I didn't manage to replicate the paper results, but I think I got close enough so, it's still a success for me ☺

- **What was your performance on that dataset (how does your report compare to theirs)?**



Accuracy on the **training set**:
91.0584%

Accuracy on the dev set:
82.9303%

Accuracy on the **test set**:
83.0415%

- **What was involved in replicating the result?**

  + *If you tried several hyperparameters, or several approaches, describe them, and what was the result of each.*

    Briefly my attempts included:

    - Different number of epochs 3-6 (till convergence).
    - Shuffle experiments - with and without shuffle on the train and the dev set.
    - Keeping the learning rate steady during the training i.e. not decay the learning rate every 2 epochs.
    - Using different learning rates such as 0.0001, 0.0003 and 0.0004.
    - Using higher dropout rate – 0.2.
    - Defining weight decay to the optimizer of 1e-5 or 1e-8 .
    - Different seed values such as 14, 16 and 1111.

    Description of my experiments in detail:

    At the paper, the number of epochs needed their model to converge wasn't mentioned. So, at first I tried using 3 epochs and got to 82.3% accuracy on the dev set. It was a nice start, but I was convinced that my model could do better . Therefore, I tried using up to 6 epochs, the results showed that the model converged on the fifth epoch with 83.316% dev set accuracy. So, I kept this result aside and tried to make the accuracy better even more. Till now, both training examples and dev examples were shuffled at each epoch. So, this time I tried to test the effect of shuffling on both training set and dev set. I ran the model 3 more times – once without shuffle at all and got 82.4%, the second time with no shuffle only on the training set examples and got 83.33%, and the third with no shuffle only on the dev set examples and got 83.32%. The next experiment I did was to not decay the learning rate every 2 epochs (as opposed to the paper) and then got 83.24% (with 6 epochs). Then, I tried to combine some of the experiments I did earlier to check if I can get any improvement of the accuracy but got around the same accuracy as before. (The next runs were with no shuffle over the dev examples). My next experiment included using a higher learning rate such as 0.0003 and 0.0004 (with the learning rate decay as it says in the paper) but got 83.04% and 83.07%, respectively. I also tried using those learning rates with a change of the seed value to 14 and 1111, but those runs too didn't get any improvement. I even checked a combination of learning rate of 0.0001 with seed 1111 and no learning rate decay which resulted with 83.15%. Next, I thought of using a weight decay of 1e-5 to the optimizer with setting the dropout rate to 0.2% (to not overfit the training set as I saw happens in previous runs). That's when I got a slight improvement to 83.428%. At the end, I have tried using seed 16 with 0.0003 learning rate and got to 82.93%, but then I noticed that this combination got me to **83.0415%** on the test set, which is the highest accuracy I got on the test set from all the experiments I have done, so I stuck with that.

- **What worked straightforward out of the box? what didn't work?**

  When I used the pre-trained 300D Glove 840B vectors to initialize the word embeddings, I couldn't train the model with GPU because the embedding matrix (which following the

paper should be fine-tuned during the training) contained more than 2 million pre-trained vectors which in practice were summed up to 658,805,100 parameters of the total model parameters. Therefore, the model just couldn't feet into the 11GB of GPU I have. Obviously, it goes without saying, that training a model with a total of 687,157,103 parameters, without GPU was pretty much impossible, time wise. Therefore, in order to be able to train the model with GPU I had to initialize the word embeddings with the pre-trained 300D Glove **6B** vectors (which contained only 400 K pre-trained vectors) instead.

The rest of the implementation went pretty much smooth, since I was already experienced, from the last assignment, with obtaining biLSTM layers and preparing the data - using padding - in order to be able to run it through the biLSTM layers using batches.

- **Are there any improvements to the algorithm you can think about?**

  In the paper they only experimented with up to 3 biLSTM layers with 600 dimension each, so in my opinion the model has potential to improve the result further with a larger dimension and more biLSTM layers.