Itamar Trainin – 315425967
Guy Gispan - 316332444

Assignment 4: Implementing a model from the literature

Introduction

For this assignment we have chosen to implement the ideas presented in the paper: *"Shortcut-Stacked Sentence Encoders for Multi-Domain Inference"* by Yixin Nie and Mohit Bansal. The paper tackles the problem of Natural Language Inference (NLI) using a deep (stacked) Bi-LSTM network with skip connections from the input and all previous layers to each layer in the stack. In addition, fine-tuning of the pre-trained embeddings was used to achieve the best results. The idea behind the paper is to take the 'hypothesis' and 'premise' sentences, encode them using a Bi-LSTM network and then compare the two encoding to decide on the correct label indicating how they relate. This approach is useful since if the two sentences relate then they have similar encoded representations.

We have chosen this paper because we have found the idea of 'skip-connections' to be an interesting and useful approach that improves the basic RNN model that we have learned in class and experimented with in previous exercises. In addition, the paper had a relatively small number of parameters compared to the state-of-the-art approaches as well as that the implementation seemed feasible in the time frame that was available to us. Furthermore, the authors of the paper have provided their implementation of the model, assuring us that we could turn for help in case we fail to implement correctly (we had no use for their code since we were able to easily implement the model ourselves). In action we have realized that the number of parameters (9.7M) was rather very large, especially too large to run on a personal computer.

The Task

In the problem of NLI, a machine is asked to determine whether an "hypothesis" sentence is true (entailment), false (contradiction), or undetermined (neutral) given a "premise" sentence. That is the machine is asked to predict a label ('entailment' / 'natural' / 'contradiction') from two input sentences presented to it.

The paper was evaluated on the benchmark published by *The Stanford Natural Language Inference (SNLI) Corpus* and have achieved accuracy of 85.7% on test set and 89.8% on the train set (see more detailed results in 'The Results' section).

The Data

In the paper, the neural network was trained and tested on the *SNLI* and the *MultiNLI* datasets published by *The Stanford Natural Language Inference (SNLI) Corpus*. The SNLI dataset is a collection of 570k human-written English sentence pairs manually labeled. The MultiNLI dataset is a collection of another 433k labeled sentence pairs, this dataset is modeled on the SNLI corpus but differs in that it covers different genres of spoken and written texts. In addition, this dataset provides both matched and mismatch test sets.

Inspection of the datasets reveled that both are available in '*.jsonl*' format and that both are balanced over the different labels. We learned that the golden label was taken based on the majority vote assigned by the different annotators and that if there was no clear majority vote between the annotators, a '-' label was assigned. In the paper this label was used in the learning process however was ignored in the loss computation. In addition, we treated this label as a 'padding' label. Finally we realized that in order to convert the different words in the sentences to vector representations, we can use the tokenized representation provided in the datasets.

The architecture

The architecture used in the paper includes 5 different parts. Given a pair of sentences the model predicts a label in the following way:

1. <u>Embedding Layer</u>
   Each word in each sentence was converted into a vector representation using GloVe 840B 300d pre-trained embeddings (840B words, each represented as a vector of 300 dimensions). In some of the experiments done in the paper, the pre-trained embeddings were fine-tuned during the training process to potentially better fit this problem. This layer helps the network to better understand the different words in each sentence.

2. Stacked Bi-LSTMs with Shortcut Connections
   After each sentence was converted into an embedded representation, it is passed through multiple layers of bi-LSTMs of varying dimensions. This process creates an encoding of the sentences that could help the network make better decisions. Shortcut skip-connections were introduced to each layer. In some experiments there were no skip-connections, in others there were connections from the input (embedded words) to all the other layers, however the best results were achieved when using skip-connections from the embedded words as well as the outputs of all the other layers computed before the current layer, fed into the current layer. The skip connections are implemented as a concatenation between the different vectors fed as input to the current layer.
   This layer is applied to both 'hypothesis' and 'premise' sentences.
3. Max-Pooling Layer
   The output of the stacked Bi-LSTMs layers is passed through a max-pooling layer which reduces the sequence of vectors into a single output (tensor of the length of the batch). This approach is useful over others, such as 'acceptor-like' approaches, because it helps the network pass information that was seen in earlier time in the sequence and might also be important for the final decision.
4. Entailment Layer
   This layer is intended to help the network make sense of the two encoded sentences. The heuristics used in this layer are meant to effectively capture the relationship between the two encoded vectors in a low complexity approach. First the encoded versions of both sentences are concatenated and passed with no change, as usually done in "Siamese" networks. Secondly the two encodings are subtracted from each other and lastly the product of the two encodings is computed. Those heuristics are in place to help the network exploit the notion of 'similarity'. Intuitively, if they are similar, they should have similar encoded representation and therefore their difference should be close to zero. The result of the different operations is concatenated to provide a single output vector.
5. MLP Layer + Softmax
   The entailed vector is then passed into a multi-layer perceptron. This process is used to classify the best label to output based on a given entailed vector.

The Results

In our attempts to replicate the results, we have **failed** to match the results achieved by the authors of the paper. We believe that we were not able to do so for the following reasons:

1. We were only able to use 75% of the training data. We were not able to pass the whole training data through the model because the memory that was available for us was limited.
2. Similarly to (1), we were only able to upload 130 examples from the total of 10,000 examples available in the dev set. We preferred to upload more training data instead of dev data.
3. In our longest experiment, we run our code for only 25 epochs which took more than two days. It can be seen from the learning curves that this was not enough processing time to achieve good results.
4. The suggested model required training of 9.7M parameters and therefore it took long time to converge.

Attempts done in-order to reproduce the results in the paper:
Since we failed to replicate the results in the paper, we performed a series of experiments in attempt to achieve the best possible results and overcome the gaps in resources found between the authors and ourselves. To do so we have taken different approaches and performed multiple experiments. The following explains our different approaches and notes the results achieved and conclusions we came to.

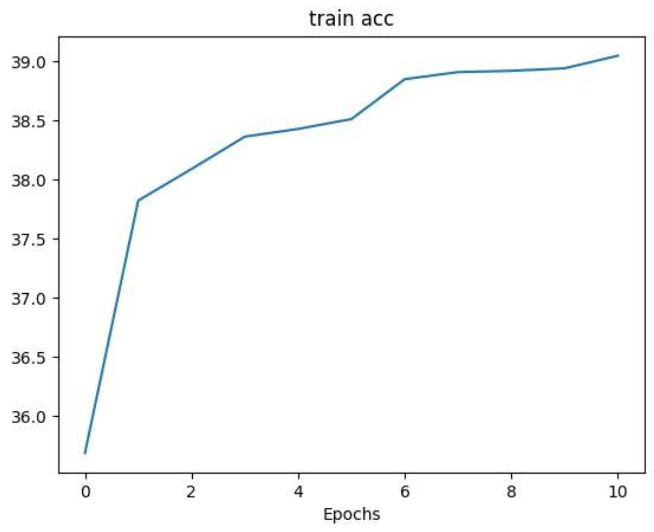
1. **Overcoming the memory issue**
   Once we have decided to limit our available data, it was clear for us that we will not be able to replicate the results in the paper. Therefore, we first tried to find some other platform that has large memory availability and could replace our personal computer. We naturally turned to the popular web services. We have tried to run our code on *Google Colab*, *AWS* and *Microsoft Azure*. However, we were not successful in using those services in the time that was available for us. As for *Google Colab* the free cloud storage was still not enough. As for *AWS* and *Microsoft Azure* we were not able to set up a

machine with a GPU in the time that was available for us (we did not get approval for adding vCPUs in time).

2. **Tinkering with the hyper-parameters**
Accepting that we have less data than was originally used in the paper we kept on trying to get some good results that are at least close to the results found in the paper. To do so we had to play with several hyper-parameters such as the learning rate, the number of layers, the dimensions of the different layers, the optimizing method and the train and dev data ratios (more training data required us to use less development data and vice versa). In the following we will discuss our different attempts:

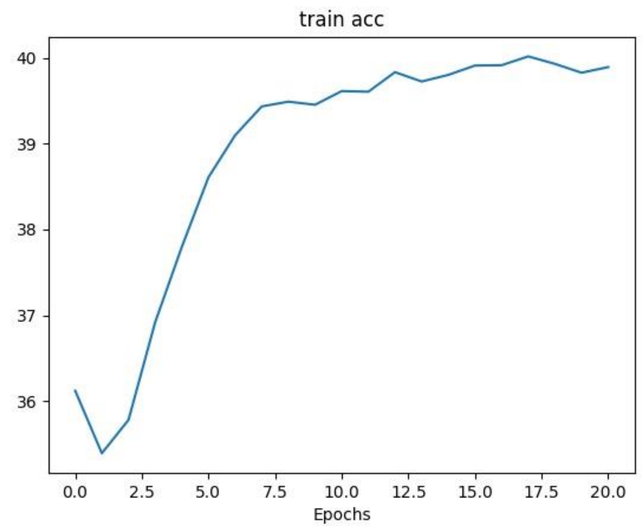
   a. Tinkering with the train and dev ratios: small train set
   First, we have tried to upload only 1,000 training examples. The idea was that we might prefer many epochs on a small train set (with an optimistic hope that 1000 examples will be enough for learning this assignment). The computation of each epoch was very fast, however our model continued to perform poorly. We were only able to achieve about 30% accuracy on the dev set and about 50% accuracy on the train set.
   We have tried to use Adam (as was used in the paper) and SGD as optimizers. They both gave us similar results. We concluded from the experiment that for this assignment, we must use as much train data as possible.

   b. Tinkering with the number of layers
   Next, we decided to take 350,000 examples from the MultiNLI dataset (that is about 75%) in addition to the 15% SNLI examples as specified in the paper. This forced us to use less development data. Therefore, we limited the dev set to include only 130 examples.
   In this experiment, we chose to test the effect of the number of layers in the Bi-LSTM and MLP parts. We generated our model with two Bi-LSTM layers of sizes [512, 1024] and two linear classifier layers [1600, 200]. Note that the additional linear layer was not used in the paper however it was a recommended approach by the authors.
   We made multiple attempts on this configuration, while using different learning rates and number of iterations in each such attempt. In the first attempt we used learning rate of 0.0002 (as suggested by the authors) and ran the model for 10 epochs. The training accuracy graph is shown below:



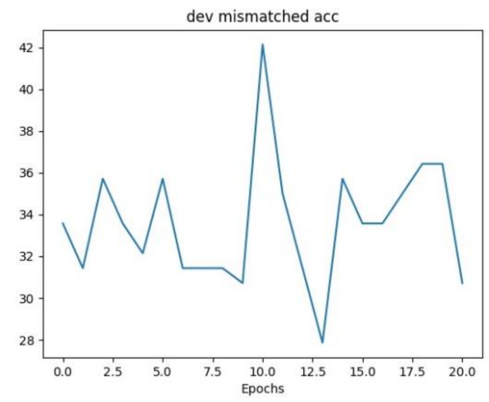
   It is clear to see from the graph that we have achieved an accuracy of 39% on the train set on the 10th epoch.
   This result was not promising, which lead us to question the quality of the learning rate. Therefore, we performed another experiment setting the learning rate to 0.002 in hope for faster convergence of the model. The next graph shows that the change helped. As can be

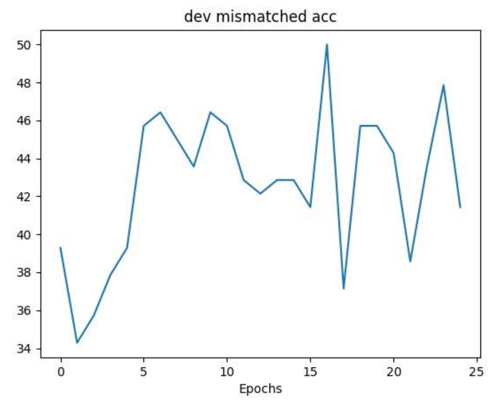
seen easily, we have crossed the 39% mark as early as the 6th epoch in comparison to the 10th epoch in the previous experiment.

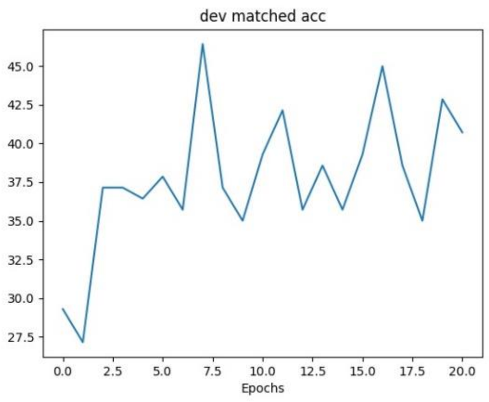


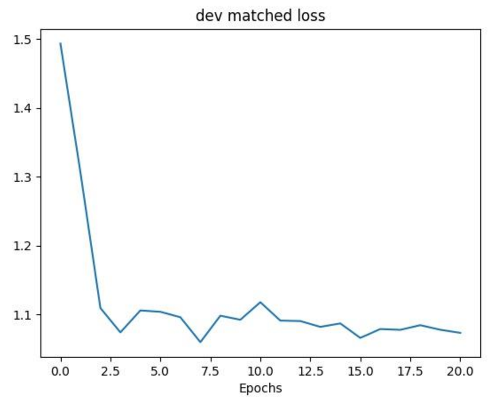
Note, that in this experiment, we noticed that the matched-dev accuracy was higher than the mismatched-dev accuracy. Furthermore, the graphs based on the dev set are a little noisy. This is first because they are computed based on a really small dataset (130 examples) and secondly, because the parameters were updated based on the train examples and therefore not every update was helpful for the accuracy on the dev set.

The graphs below show the accuracy and loss on the mismatch and match dev data:

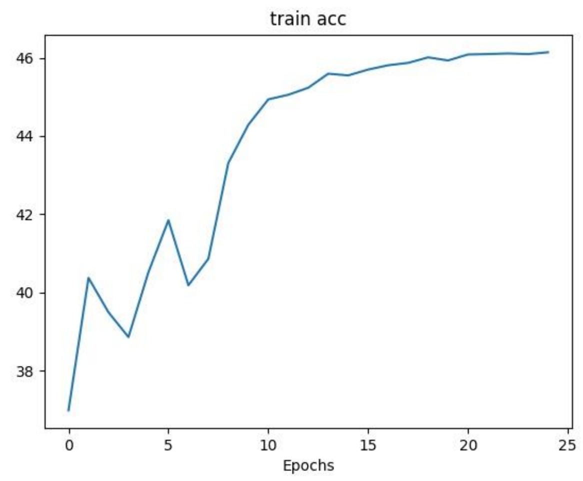


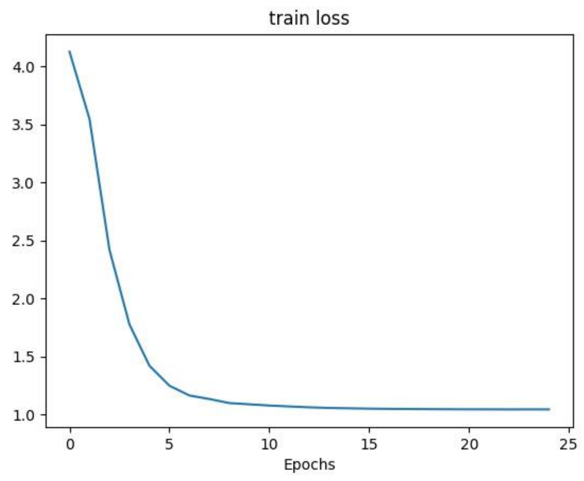




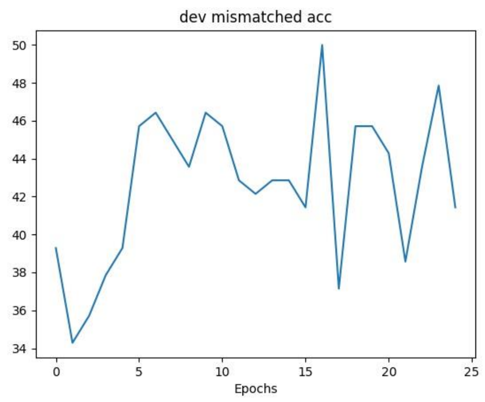


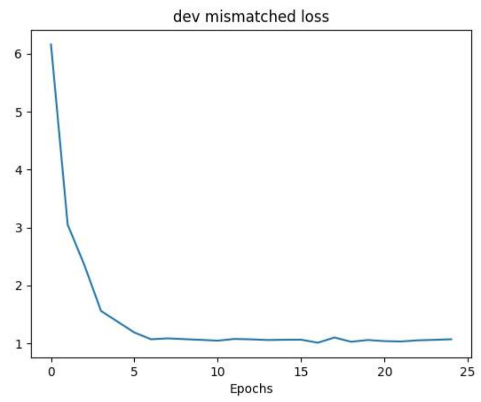
c. <u>Mock the layer structure suggested in the paper</u>
In the last experiment, we have tried to run the model according to the best parameters described in the paper. We used three Bi-LSTM layers of dimensions [512, 1024, 2048] and only one linear layer of dimension [1600]. The only difference was that we have used less data. In addition, we were not able to run the model for more than 25 epochs due to time considerations. We believe that the authors of the paper ran their model for more epochs. Furthermore, we can see from the graphs that the accuracy is constantly improving. This indicates that running for more epochs would of helped for achieving better results.
The graphs below shows the accuracy and loss on the train data:



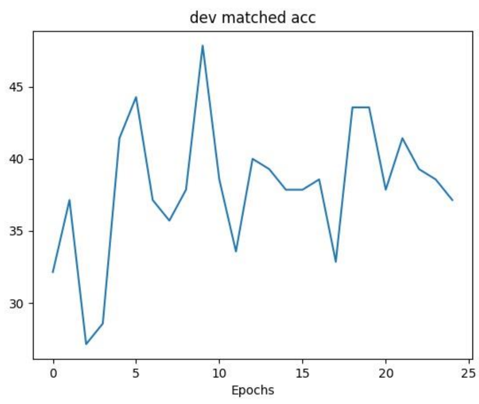


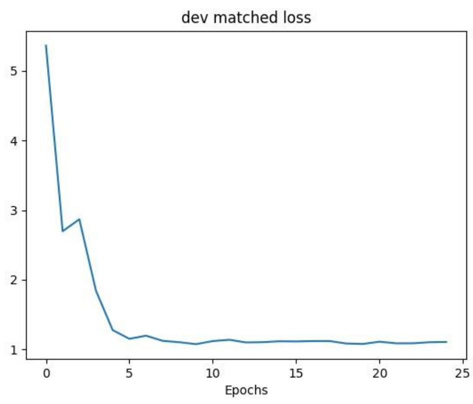
One can notice that the even when we subtract the last linear layer (the paper used only one layer and not two), the addittion of the third Bi-LSTM (layer of size 2024) was absolutely more powerful! Using this configuration, **the network was able to cross the highest accuracy achieved in all previous experiments as soon as on the second epoch** and we were able to achieve an accuracy of 46% on the train set. Although this is still far from the results achieved in the paper, we believe that we are on the right track. We believe that we would of achieved the same results as reported in the paper if we were able to use the whole dataset and would have had more time to train our model.
Note that it is useless to compare this result to the result achieved in experiment (a) since in that experiment we used only 1000 training examples.
Concluding our different experiments, we were able to achieve best dev set results of about 48% accuracy on the mis-matched data and about 45% on the matched data (with early stopping).
The achieved accuracies on the dev set can be seen in the following graphs:





Itamar Trainin – 315425967
Guy Gispan - 316332444

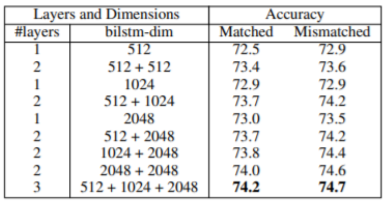




<u>Conclusion and comparison to the paper</u>**:**
The paper's results are shown in the figure to the right. The authors used different configurations examining their different effects, indicating their best results. We conclude that we did not even get close to the results achieved in the paper, however we believe that if we were able to use more data and more running time, our results would have greatly improved.

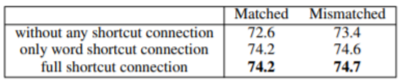
How to Improve

The improvement we are suggesting is to add an Attention mechanism on top of the Bi-LSTM. We believe that better results could have been achieved if the 'hypothesis' and the 'premise' sentences where reduced into a lighter version where only the important words that are relevant for the meaning of the sentence are left. Similar effect can be achieved with our suggestion, since this way the network can learn to give more weight to the important words and less weight to the less important words.



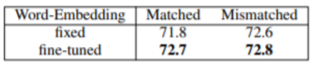
Layers and Dimensions		Accuracy	
#layers	bilstm-dim	Matched	Mismatched
1	512	72.5	72.9
2	512 + 512	73.4	73.6
1	1024	72.9	72.9
2	512 + 1024	73.7	74.2
1	2048	73.0	73.5
2	512 + 2048	73.7	74.2
2	1024 + 2048	73.8	74.4
2	2048 + 2048	74.0	74.6
3	512 + 1024 + 2048	**74.2**	**74.7**

Table 1: Analysis of results for models with different # of biLSTM layers and their hidden state dimensions.



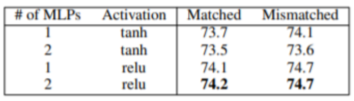
	Matched	Mismatched
without any shortcut connection	72.6	73.4
only word shortcut connection	74.2	74.6
full shortcut connection	**74.2**	**74.7**

Table 2: Ablation results with and without shortcut connections.



Word-Embedding	Matched	Mismatched
fixed	71.8	72.6
fine-tuned	**72.7**	**72.8**

Table 3: Ablation results with and without fine-tuning of word embeddings.



# of MLPs	Activation	Matched	Mismatched
1	tanh	73.7	74.1
2	tanh	73.5	73.6
1	relu	74.1	74.7
2	relu	**74.2**	**74.7**

Table 4: Ablation results for different MLP classifiers.

Tables 1-4 from "Shortcut-Stacked Sentence Encoders for Multi-Domain Inference", Nie et al.