Exercise 4

Refael Shaked Greenfeld 305030868,

Danit Yshaayahu 312434269

# Which paper you chose to implement

Learning Natural Language Inference using Bidirectional LSTM model and Inner-Attention by Yang Liu, Chengjie Sun, Lei Lin and Xiaolong Wang

# Why you chose that particular one.

First, we tried to find a model without many parameters because less parameters would decrease the computational complexity and reduce running time (which is important when you have limited computational resource).

Also, we wished to experience with Attention mechanism because we didn’t have the opportunity to do so in the course. We believe it is fundamental skill that we wished to acquire.

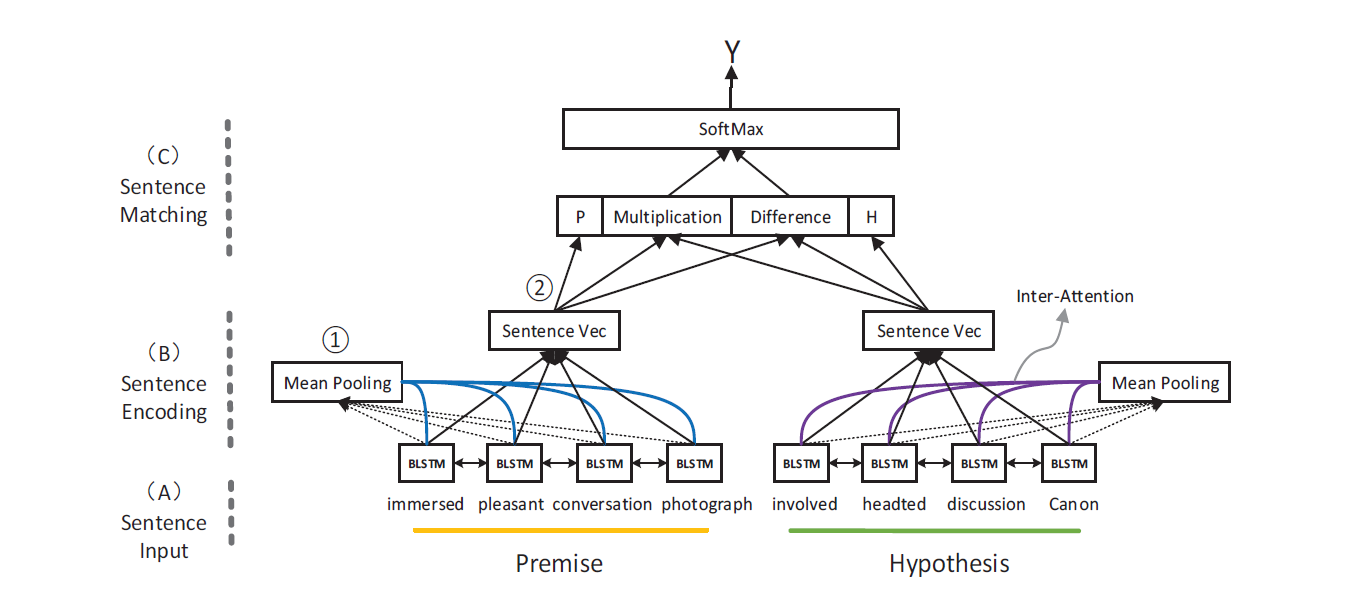
Finally, we felt we can put into action things we learned in the course such as Bi-directional LSTM. In this assignment we had the chance to experience with those things with some different data and with less guidance.

# What was the result reported in the paper?

|  |  |  |
| --- | --- | --- |
|  | Train | Test |
| Accuracy | 84.5 | 84.2 |

# What method was used in the paper?

The paper used a method called “Inner-Attention” on top of a mean pooling of a Bi-LSTM. The inner-Attention supposed to emphasize the important words in each sentence (premise and hypothesis) and afterwards concatenate the results of the two plus the difference and their multiplication, When the Network is trained in a Siamese network form (same network compute the hypothesis and the premises sentences) – not mentioned if only the LSTM is shared or also the Inner-Attention. The architecture described above looks as follow:



The inner attention equation () is:

When:

1. is the output of the words form the Bi-LSTM
2. is the mean pooled vector from the Bi-LSTM output.
3. is the unite matrix[[1]](#footnote-1)
4. (all parameters of the model)

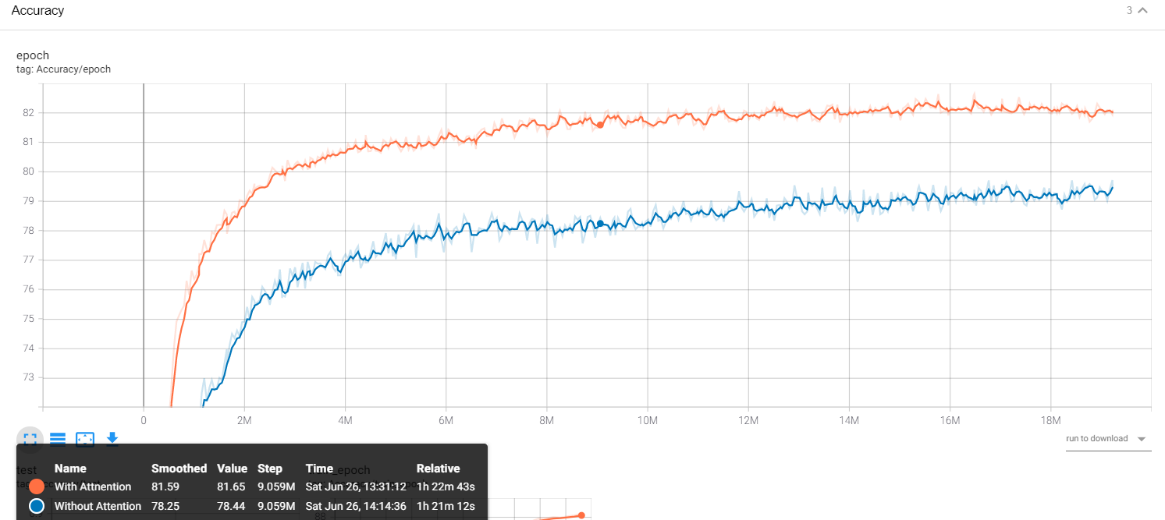
# Did your code manage to replicate this result?

Yes, our code manage to get very similar results (with a minor difference of 0.1 a point reduction that we believe is not very significant)

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

|  |  |  |
| --- | --- | --- |
|  | Original paper | OUR MODEL |
| TEST Accuracy | 84.2% | 83.7% |
| tRAIN aCCURACY | 84.5% | 89.8% |

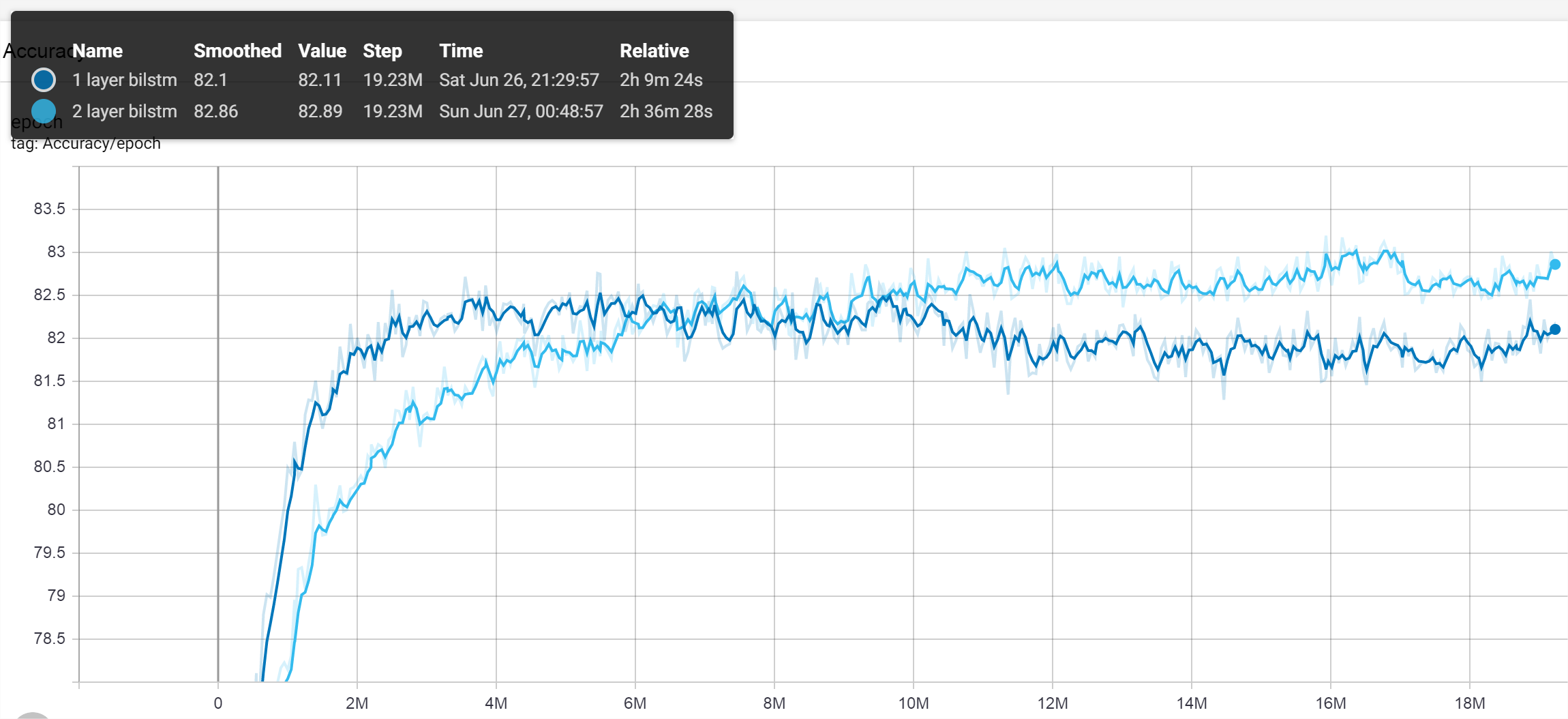
# What was involved in replicating the result?

1. First, in our vanilla implantation there were some unclear points. Such as the hidden dimension size, number of layers and finally what turned to be very important, the final “**nonlinear projection”** (it was mentioned in this way in the paper and we couldn’t understand what it really meant until the last moments)
2. We tried to run the net with and without the attention and we saw a significant increase in both the dev accuracy and in the test accuracy, so we conclude that our implementation of attention was indeed correct. The accuracy graph for test set on two runs with the same exact parameters when one is with the inner-attention and the other is without:

The attention added 2.71 points to the test score (81.54 vs 78.83). As so, we believe we don’t have a problem with the attention implementation.

1. We also tried to check whether the attention supposed to be shared across the premise and hypothesis sentences, but we didn’t notice any significant difference (a difference of less than 0.01 in test on the same parameters) and because it increased in a significant way the number of parameters and also not intuitive (at list without mentioning such a thing in the paper) we have decided to drop it.

1. Playing with the dropout – they did mention the p of the dropout but it isn't clear if they used it only in the end of the net or maybe in more places. So we added the location of dropout as a hyper parameter. The options was Boolean on the use after the attention and between the layers when we always used it after the LSTM output.
   1. Also, we did try to play with Dropout percentage on the values [0.2, 0**.25** (original value), 0.3. 0.35]
2. Although Number of layers in the bi-lstm was seem to be 1. We tried to play with the number of layers - 2 layers indeed added to the accuracy of the dev set, but then the parameters did not fit the reported number (3.5 vs 2.8)



1. Optimizer - In the paper they used mini batch RMProp – but didn’t share all of its parameters, we decided in early point (when we were 2.5 point far away from the reported score) to use AdamW with different learning rates and different weight decay. We sticked to this optimizer in the end because empirically it worked better overall.
2. Using Xavier initialization – we tried some Kaggle tricks to improve score, although it did help, it wasn’t close to the requested score.
3. The paper mention that the last layer is a non-linear projection to the number of classes. The authors didn’t give the exact details about this function and in the beginning we tried to use a linear layer with Relu() activation function (for the non-linearity). But later on, after some thoughts, came into conclusion that Relu() which is linear for every value > 0 is less appropriate for our purpose and we replaced it with Tanh(). Eventually we saw improvements in the test set as follow:

|  |  |  |  |
| --- | --- | --- | --- |
|  | without xavier and without tanh | with xavier and without tanh | with xavier and with tanh |
| TEST Accuracy | 81.63 | 81.34 | 82.14 |

1. nonlinear projection + number of parameters:

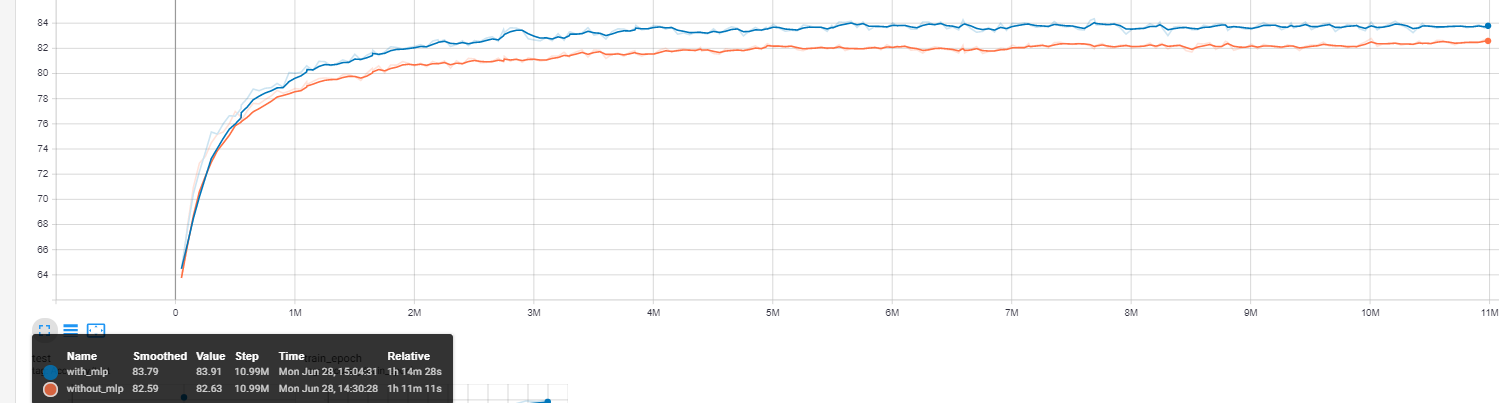
Finally we had a problem - our model yield after parameter tuning 82.55 accuracy (~1.7 less than reported) with more parameters – because we used 2 layer LSTM.

What was “weird” in our model was the final phase – the **nonlinear projection** – at first we just did a simple vector with Tanh() which transform 800 dimensions vector into 3. Which felt too much to handle. In addition, when we moved back to 1 layer LSTM we still left with about 0.5 M parameters short. Than, we decided that this may be the trick – adding an extra MLP with activation function that would first reduce dimension from 800X200 and then prediction vector 200X3. This "trick" reduced model's parameters to 2.8M, as mentioned in the paper, and also improved our accuracy.

This gave us the following result:

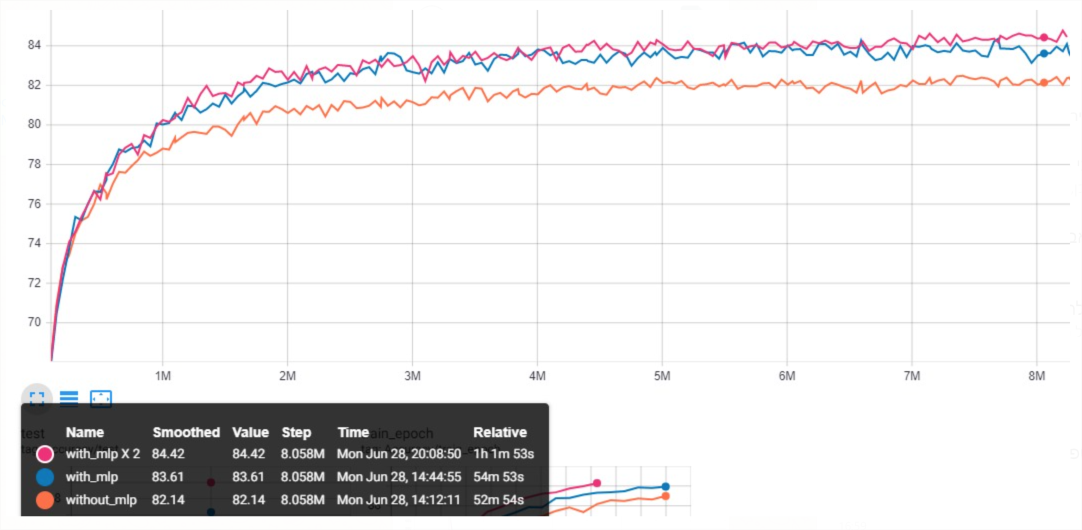
|  |  |  |
| --- | --- | --- |
|  | With MLP | without MLP |
| TEST Accuracy | **83.63** | 82.11 |

Dev accuracy (on same results):

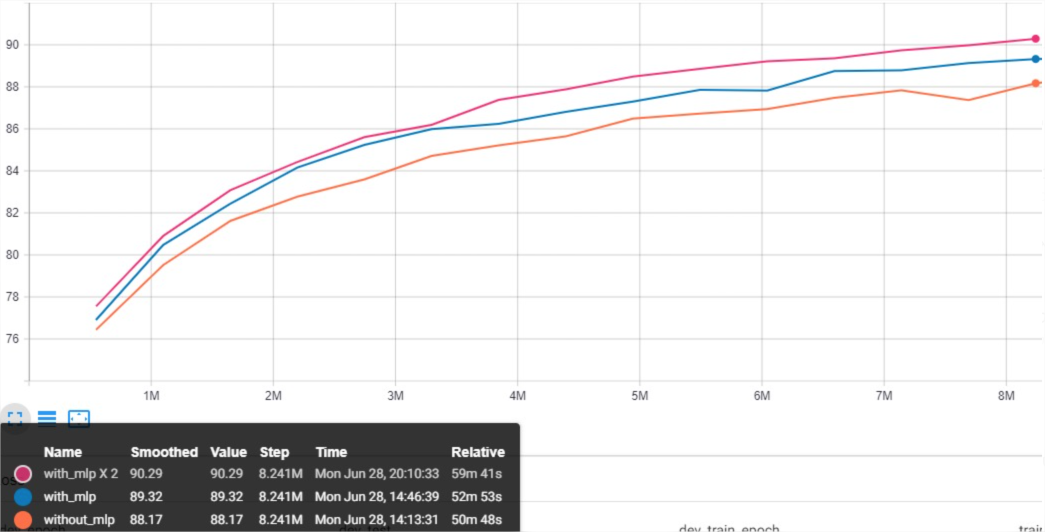


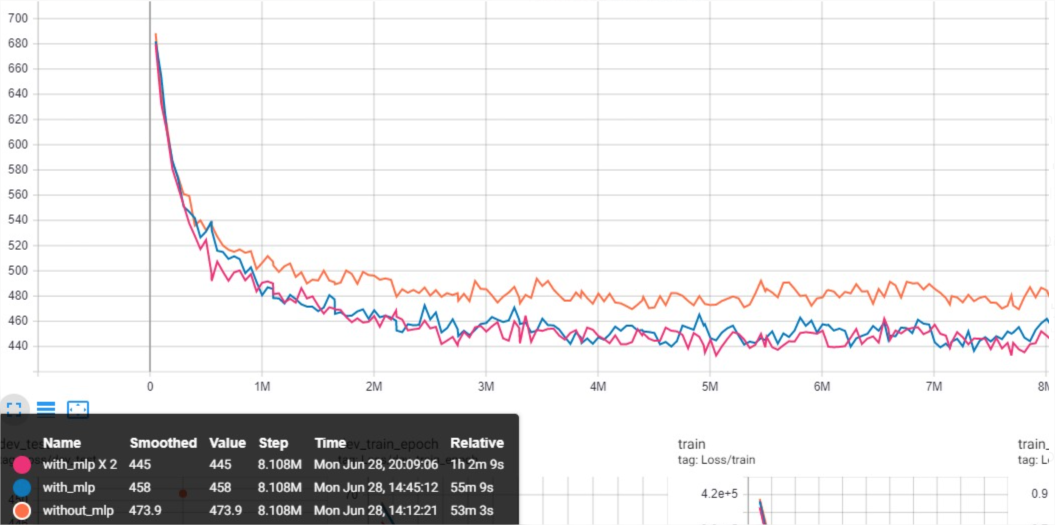
|  |  |
| --- | --- |
|  | With 2 MLPs |
| TEST Accuracy | **84.01** |

1. We understood that the trick is working therefore we decided to add another MLP for smother reduction. Now our first MLP reduces from 200 dimensions to 100 dimensions and the second one from 100 to 3. The final results are the following:  
     
     
     
   Dev accuracy:

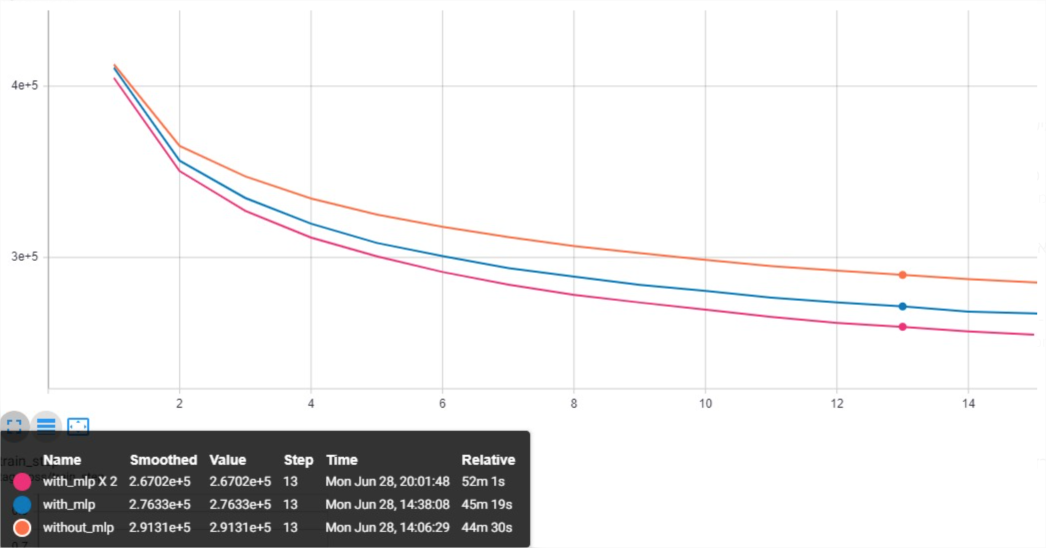


Train accuracy:



Dev loss:

Train loss:



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

The attention layer and the overall architecture seemed to work out of the box except the final layer. We have manage to understand how the main flow works and acts and also managed to write the code accordingly.

What didn’t work was the last layer, we believe that we miss interpreted the term "nonlinear projection" and over simplified it.

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

We would try several things:

1. Adding embedding which sign verbs\nouns\entities in the document and embed them with the glove ones.
2. Adding a third vector which is a concatenation of both inputs that would also run through the network and may replace the multiplication and reduction vectors used in the paper.
3. Add another LSTM/MLP on top of the concatenated vector. We observed a significant jump using the addition in our model – and we believe there is still more room to explore.

1. As far as we understood, we tried to reach out to the writer but they didn’t answer us and we consulted some experts (and literature) who advise us so. [↑](#footnote-ref-1)