Report - Assignment 4
Deep Learning for text and Sequences – 89687
Lior Shimon - Lev Levin
341348498 - 342480456

- ***Which paper did we choose to implement?***

  We choose: 200D decomposition attention, published the 6th June 2016 as: "A Decomposable Attention Model for Natural Language Inference" from Parikh et al. The full paper is available at: https://arxiv.org/pdf/1606.01933v1.pdf

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

  In order to catch the features of Natural Language Inference, this paper used a simple attention-based approach: firstly, the model creates the attention in the Attend phase, then evaluates the attention of each subphrases of sentence 1 with those of sentence 2 and vice-versa in the Compare phase, then finally sum the evaluation subphrases of each sentence and concatenate them before feeding a mlp with them.

- ***Why did we choose this model?***

  We decided to implement this particular paper for several reasons. First of all, the model is not memory consuming, with approximately 380k parameters. We also noticed the model was relatively simple to manipulate, with only few layers (8 layers not including embedding) and is eventually parallelizable easily. Finally, this paper shows that a relatively simple neural network can reach at least as good result as much more complex and slower networks (LSTM, BILSTM) for Natural Language Inference: Using an attention-based approach, it shows that pairwise comparisons are more important than global sentence-level approach.

- ***Description of our implementation:***

  We decided to work with Pytorch, and used Google colab as computing resource. Our code is composed as follow:

1) Preprocessing the SNLI dataset

Before properly working on the model, we needed to treat the data:

- extracting the gold label, sentence 1 and sentence 2 from each line.
- mapping each word to a specific index, and the labels to their index.
- invoking the Glove 300d-42B embedding and 100 randomly initialize vectors.
- shuffle the examples
- batching the examples in a way it optimizes the padding (sentence of +/- same size goes in a same batch)

This preprocessing part was coded in the dataloader.py script and in the beginning of main.py just before building the model, and we used torchtext library for implementing it.

2) Building the model

As requested by the paper, after converting each word of the 2 sentences to their embedding vectors, the forward propagation of this Approach was composed of 3 main parts:

Attend - the attention creation by soft alignment, Compare- comparing each aligned subphrases between the two sentences, and Aggregate- summing the two sets of comparison vectors, and concatenate them before treating them.

Each of those three parts uses the same mlp – layers: 2 linear layers with a Relu activation and a 0.2 dropout mask.

3) Training the model

Here again, we followed the instruction ordered by the paper, using Adagrad optimizer, learning rate of 0.05, etc.

We trained the model and saved the accuracy and the loss of the training set and the validation set on the run.

This section is not the most interesting part in our work, since we implemented it the same way we did in the previous assignments.

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

After 300 epochs, using TensorFlow and10 asynchronous gradient-update threads, the result reported in the paper were:

An Overall of 89.5% on the Train set and 86.3% accuracy on the Test set.

83.6% accuracy on Neutral examples on the Development set.

91.3% accuracy on the Entailment examples on the Development set.

85.8% accuracy on contradiction examples on the Development set.

- ***Parameters of our models that were not specified by the paper:***
    - We used the least square normalization.
    - We reached better result with the Glove-300d 42B comparing to Glove-300d 6B.
    - Regarding the 100 randomly initialized embedding vectors we had to add to the Glove Embedding, since any initialization type was specified, we tested some variations and choose random embedding with mean 0 and deviation 1.
    - We used the same mlp layers for the Attend Compare and Aggregate parts, composed of 2 linear layers of 200 neurons, the dropout of 0.2 before each of them and a Relu activation function only after the first linear layer.
    - We added an extra final linear layer at the end of our model, with 3 neurons.

- ***What were our result and our performance on the dataset?***

  Training the network with exactly the same parameters that were mentioned in the paper, the model didn't succeed to learn.

  Training the network with the same parameters that were mentioned in the paper but with additions that were mentioned in a previous paragraph, and with batch size 4, without regularization, after 40 epochs (about 7 hours of continuous training) we had reached the following results:

Epoch [40]:

Train_Loss: 0.820, Train_Accuracy: 66.68%

Dev Loss: 0.689, Dev_Accuracy: 70.99%

However, after we trained with the same parameters, but with batch size 16 and regularization l2 with weight_decay 1e-5m, after 170 epochs (about 7 hours of continuous training) we had reached the following results:

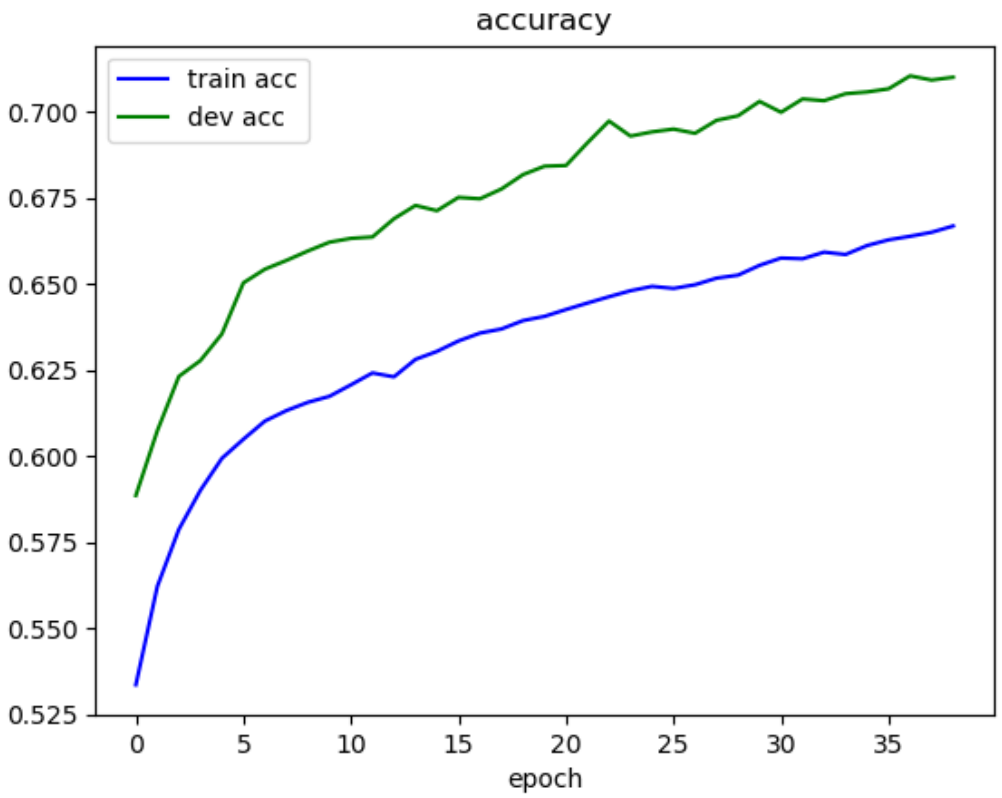
Epoch [170]:

Train_Loss: 0.698, Train_Accuracy: 73.79%

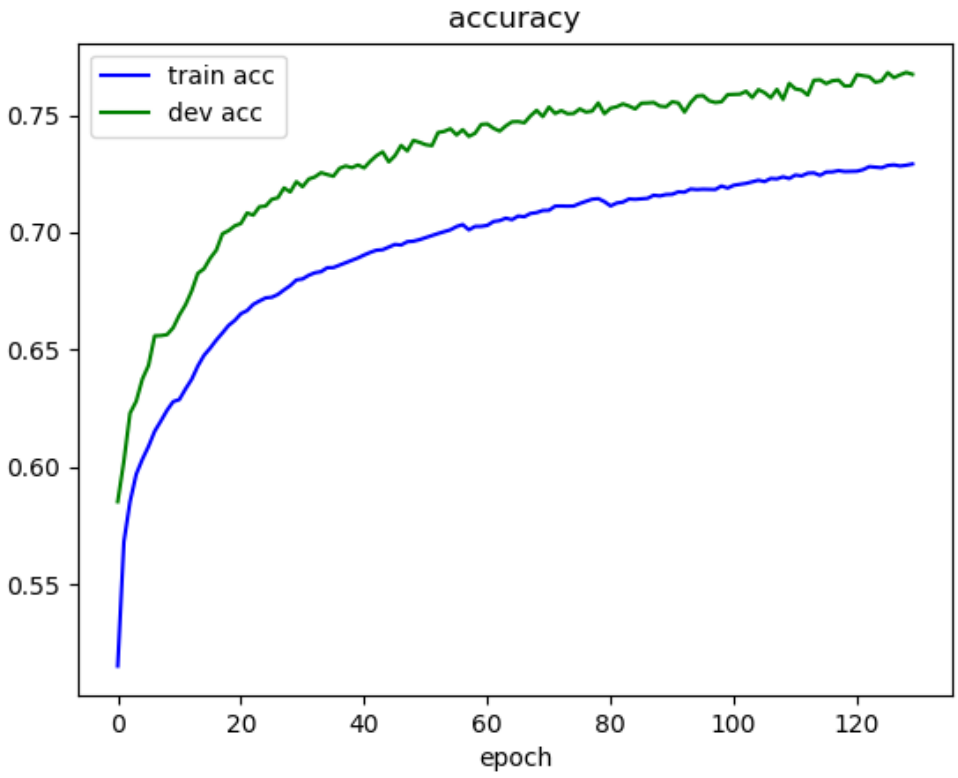
Dev Loss: 0.566, Dev_Accuracy: 77.71%

- ***curve graphs***

  graph of accuracy as a function of number of iterations (batch size 4, without l2 regularization:



graph of accuracy as a function of number of iterations (batch size 16, with l2 regularization 1e-5):



- ***Which problem did we met during the phase of training ?***

  As expected, we met small problem before starting to properly train the model:

  - RELU Problem: as we understood by reading the paper, we tried to use the Relu activation function after each linear layer. Too many values were then set to 0.

    We decided to remove the Relu after the second linear layer in the mlp.

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

  We think that the algorithm can be improved by adding the next function in the 'compare phase':

  To each word's vector representation, concat the vector/number that represents part-of-speech tag for the current word, and then feed the concated vector to G feed forward vector.

  It could improve the model because we add more accurate information about each word that may help the network to avoid to learn from illogical comparisons. For example, if we compare two words 'the' and 'beautiful', then the result of comparison doesn't bring us a useful information for natural language inference because 'the' it's determiner and 'beautiful' it's adjective and possible may only distract the model(noise). However, if we compare word such as 'night' and 'day' where both of them are nouns, the concated value 'noun' to both words can help the network to understand faster that there is a possible contradiction between the two sentences.

  Note that adding such a feature doesn’t impact the speed of the learning process.