Ass 2 – Part 4

Dorin Keshales – 313298424

Model's Parameters

In both models:

- I used dropout, after the activation function, with probability of p=0.5.
- Optimizer: Adam
- Learning rate: 1e-3
- Size of the hidden layer: 150

With random embeddings initialization

POS:

- I used batches of size 128 for both training and dev sets.
- Number of epochs: 6

NER:

- I used batches of size 16 for both training and dev sets.
- Number of epochs: 7

With pre-trained embeddings initialization

POS:

- I used batches of size 32 for both training and dev sets.
- Number of epochs: 5

NER:

- I used batches of size 16 for both training and dev sets.
- Number of epochs: 6

Important points regarding the model

- When asked to use sub-words by assigning an embedding vector to each prefix of 3 letters and each suffix of 3 letters, I had to deal also with words that their length is 3 or less. The way I chose to handle it is with taking the whole word as a prefix and as a suffix.

- For each of the conditions (sub word-units, pre-trained) and (sub word-units, no pre-trained) I used 3 embedding matrices – the first for the words in the vocabulary, the second for the prefixes of the words in the vocabulary and the last one for the suffixes of the words in the vocabulary. I have done that in order to compute the sum (= prefix + word + suffix) and pass it as an input to the MLP .

- For unknown words, if their prefix or their suffix is in the prefixes/suffixes vocabulary then I assigned them the corresponding embedding vector to this prefix/suffix.

- For the condition of the sub word-units and random initialization of the embeddings vectors I used the ‘UNK’ token for unknown words (which are not in the vocabulary) and had also an unknown embedding vector for unknown prefixes and an unknown embedding vector for unknown suffixes. Resulting the way I chose to handle with words that their length is 3 and down (described at the end of the first page), the ‘unknown token’ for both unknown prefixes and unknown suffixes is called ‘UNK’ as well. But of course, each 'unknown token' is part of different vocabulary, so by that the identical token name doesn’t represent several situations in the same vocabulary. As I mentioned before, I have 3 embedding matrices and each one corresponds to a different vocabulary.
  There are 3 vocabularies: one for the words themselves, one for the words' prefixes and one for the words' suffixes

- For the condition of the sub word-units and the pre-trained vectors I used the ‘UUUNKKK’ token for unknown words (which are not in the vocabulary).The corresponding 'unknown token' for unknown prefixes was ‘UUU’, because it’s the prefix of the 'unknown token' which belongs to the pre-trained words vocabulary. And similarly, the corresponding 'unknown token' for unknown suffixes was ‘KKK’.

Brief analysis of the results

In my opinion the following pre-trained embeddings are very useful :
start token –‘<s>’, end token – ‘</s>’, unknown token – ‘UUUNKKK’, num token – ‘NNNUMMM’ and all the variations of patterns of numbers (DG, DG.DG, DG.DGDG and etc.) .
Because they allow generalization in the case of different numbers patterns and allow us to handle words which are not in the vocabulary but in the training/dev/test sets, by giving a similar embedding vector to those unknown words. The start and end tokens allow us to create context window to the first and last of a sequence which there are many sequences in all data sets.

For the sub-word units I think that the following are very useful:

- **Common words which appear almost in every sequence -** 'the', 'of', 'in', 'to', 'and', 'on', 'for', 'was', 'at', 'by', 'is' , 'be', 'has', 'it' , 'as' , 'not', 'he', 'had', 'his', ‘its’, 'an', 'a', 'not', 'non', 'all'.
- **Common prefixes of words -** 'pro', 'con', 'pre', 'int', 'per', 'dis', 'com', 'sub', 'imp', 'for', 'mis', 'out', 'uni', 'hav', 'sai' and etc.
- **Common suffixes of words -** 'ing', 'ion', 'ers', 'ted', 'ons', 'ate', 'ive', 'ies', 'ble', 'ity', 'ous', 'ist', 'aid', 'day', 'ent', 'ill', 'ans', 'ian', "'s",'sed', 'lly', 'any', 'ere', 'uld', 'rom' and etc.
- **Punctuation marks -** '.' , '(' , ',' , ')' , '"' , ':' , '-' , '``'

The common prefixes and common suffixes of words allow us in many cases when we run into a word that is not in the pre-trained vocabulary, to might have it’s prefix/suffix in the prefixes/suffixes vocabularies. And even though the word itself will get the embedding vector corresponding to the unknown token, we still be able to use the embedding vector of the prefix/ suffix (or both if exist in the vocabularies) of the word itself, and then allow the word a probably unique or special sum (= prefix + word + suffix) in which we use as an input to the MLP.

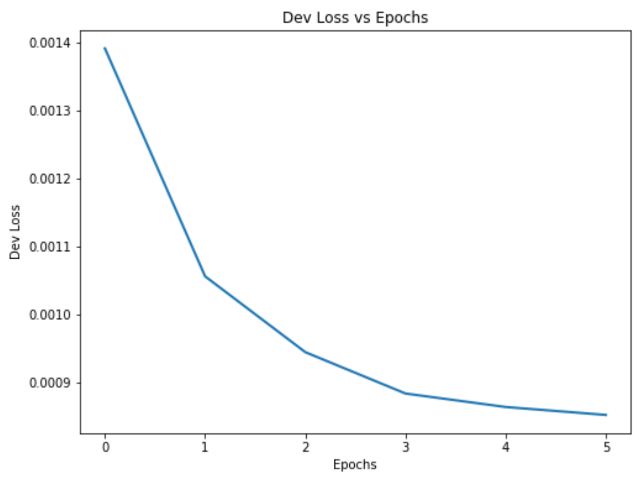
- As you can see in the next page, in both conditions the use of sub-word units increased the accuracy of my model on the dev set. When used sub-word units with random initialization of the embedding matrix you can see a significant accuracy improvement. However, with the sub-word units and the pre-trained embeddings, there is a slight of accuracy improvement.
- My conclusion from the results I got is that the combination of random initialization to the embedding matrix and the use of sub-word units will make my model better on those tasks.

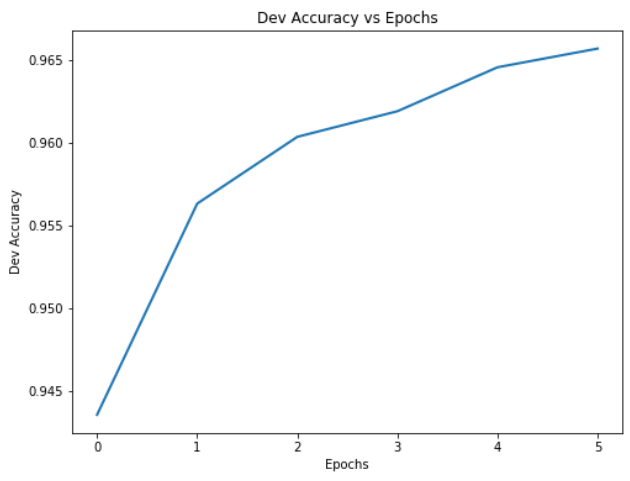
It’s consistent with the accuracy of tagger1 in compared to the accuracy of tagger2 - I got better accuracy with the random embedding vectors. As I wrote in part2.pdf, it might happened because of small overlap between the pre-trained vocabulary and the training and dev sets OR the task in which the pre-trained embeddings were learned in, does not fit to the pos and ner tasks. Therefore, the pre-trained embeddings can be irrelevant to these tasks.

My model’s results:

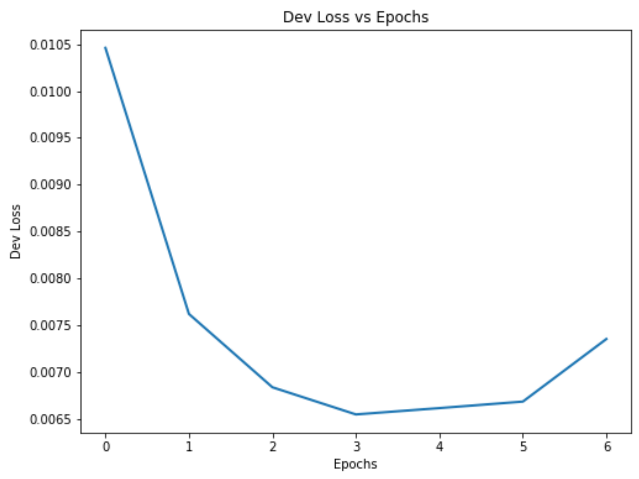
With random embeddings initialization

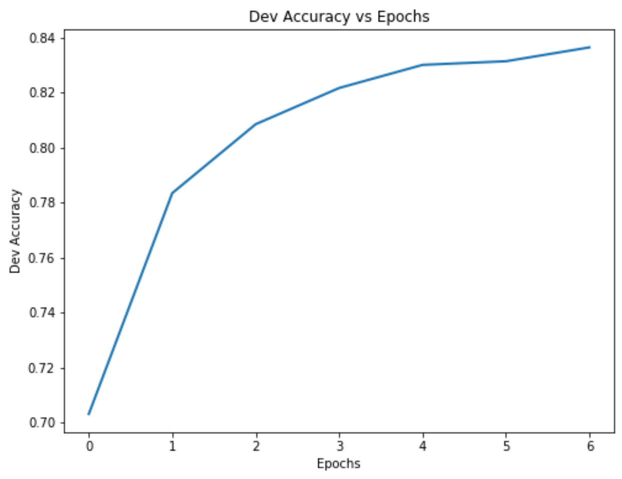
POS: Loss - 0. 00085 Accuracy – **96.567%**





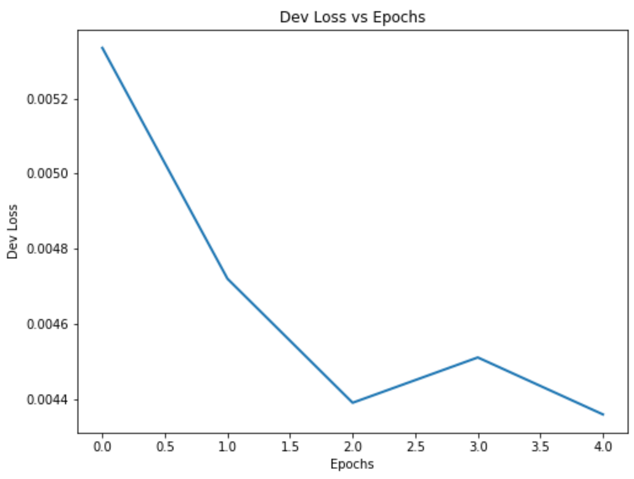
NER: Loss - 0.00735 Accuracy – **83.647%**

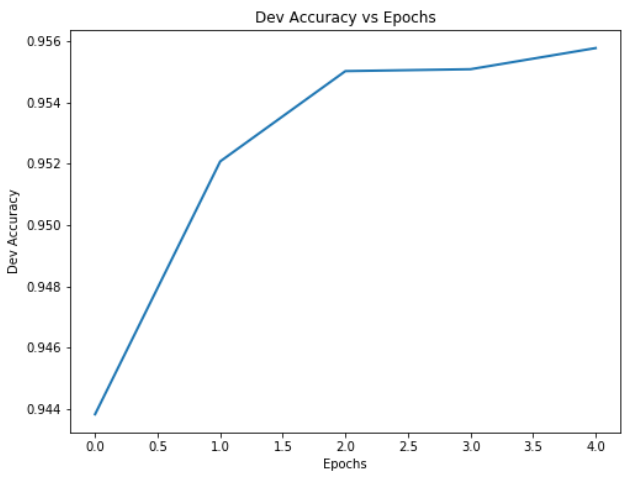




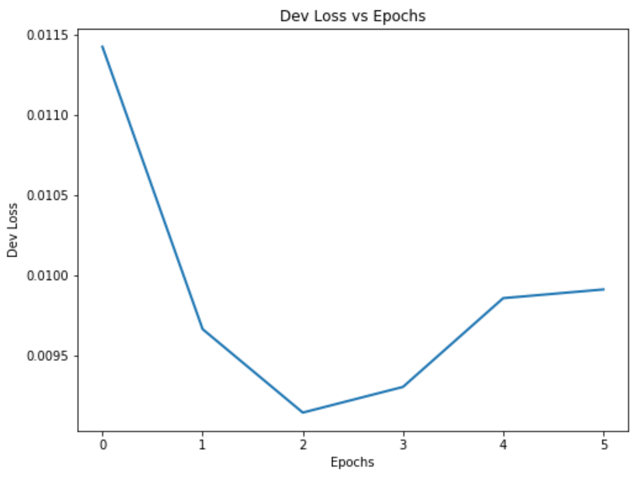
With pre-trained embeddings initialization

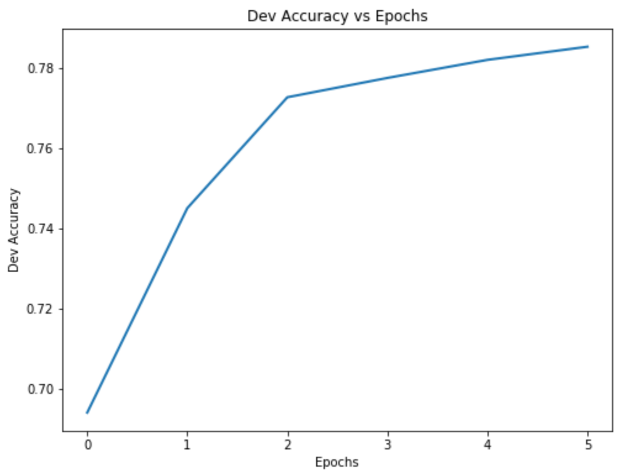
POS: Loss - 0. 00435 Accuracy – **95.577%**





NER: Loss - 0.00991 Accuracy – **78.546%**


Dev Loss vs Epochs
Dev Loss
Epochs


Dev Accuracy vs Epochs
Dev Accuracy
Epochs