Assignment 2

Part 4

Course: Deep learning for texts and sequences.

Students: Lev Levin, id: 342480456

Lior Shimon, id: 341348498

## Parameters for ner model:

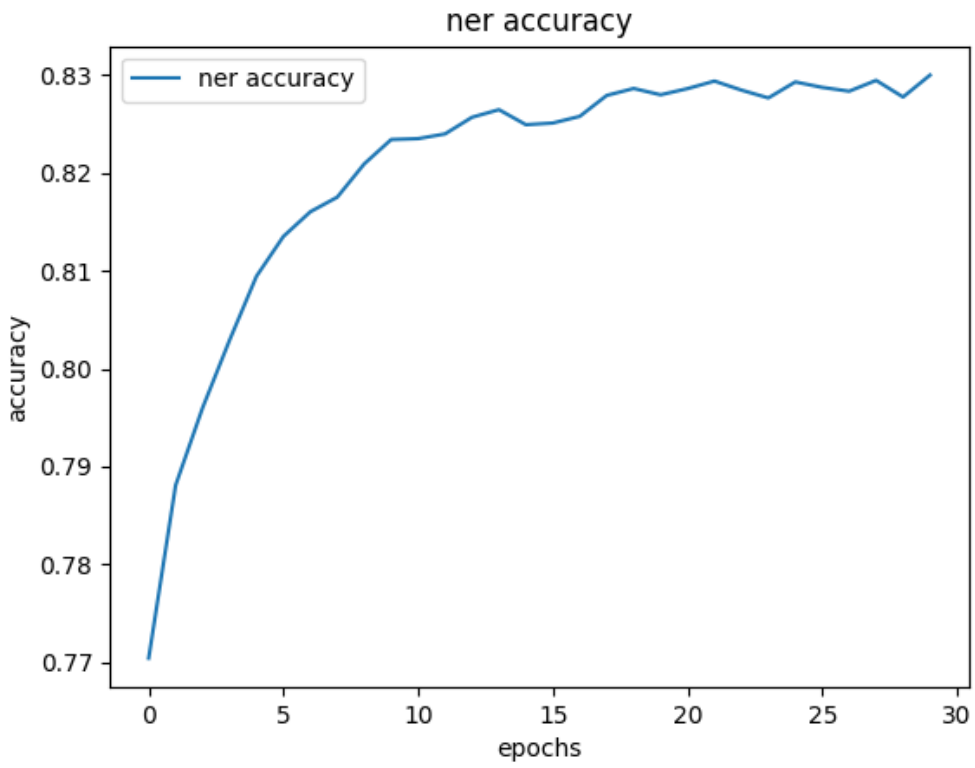
### From Pre-train Embbeding layer:

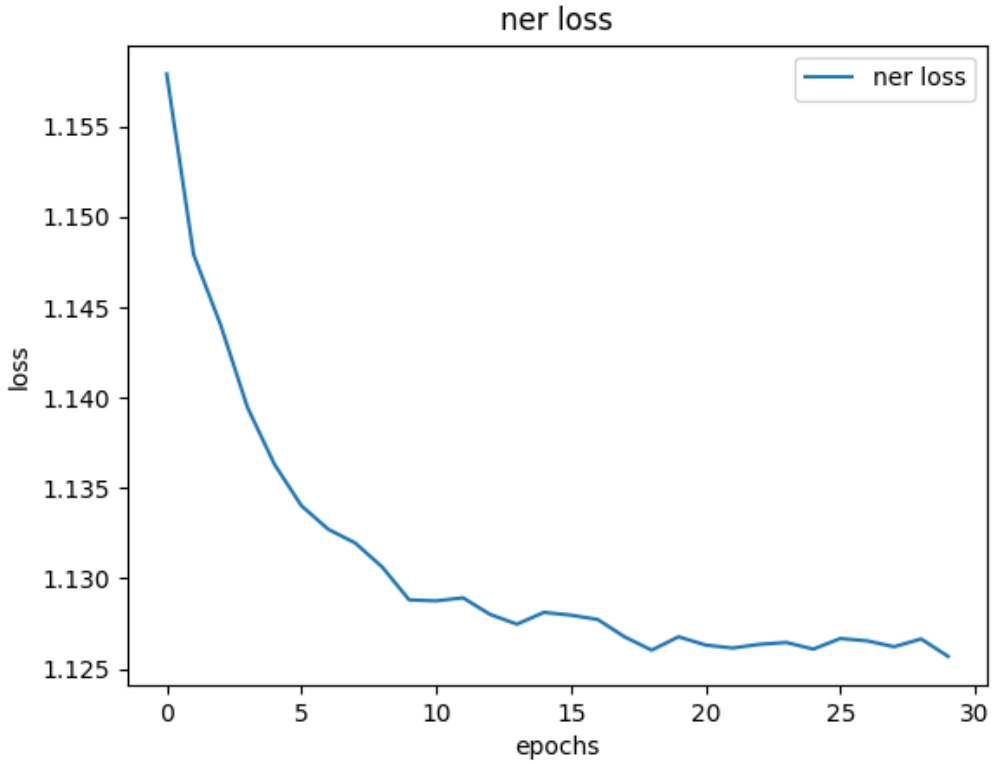
```
epochs = 30
hidden_size = 8
l_r = 0.01
torch.manual_seed(1)
batch_size = 1000
L2 regularization with coefficient: 1e-3
Optimizer: Adam
Random initialization of the Embedding layer: Uniform[-1:1]
```

With those parameters we reached:

```
Epoch [30/30], Train_Loss: 1.0551, Train_Accuracy: 98.03%
Epoch [30/30] ,Dev_Loss:1.1257,  Dev_Accuracy:83.00%
```

Graphs:






---

From randomly initialized Embbeding layer:

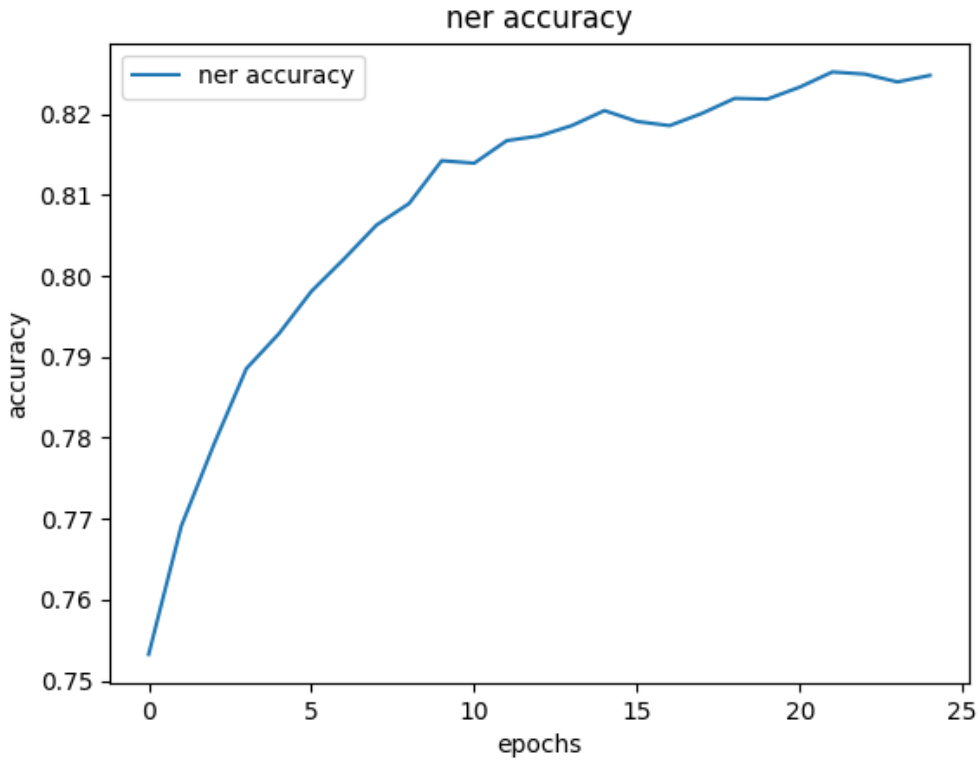
```
epochs = 25
hidden_size = 6
l_r = 0.01
torch.manual_seed(1)
batch_size = 1000
L2 regularization with coefficient: 1e-3
```

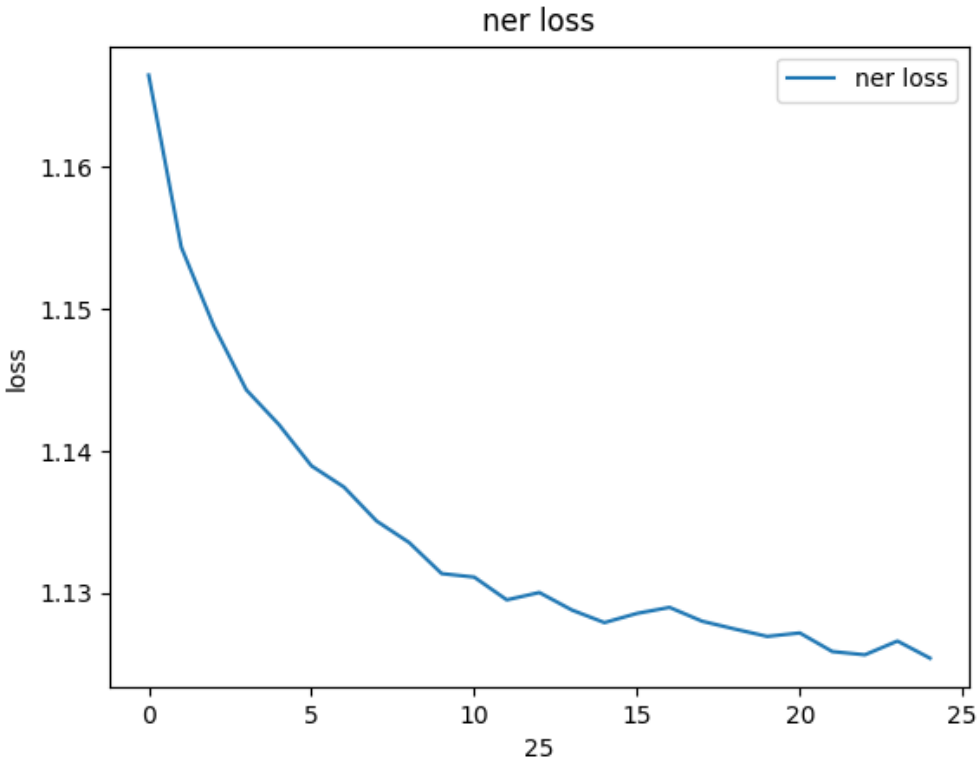
Optimizer: Adam
Random initialization of the Embedding layer: Uniform Distribution on [-1:1]

With those parameters we reached:

```
Epoch [25/25], Train_Loss: 1.0562, Train_Accuracy: 97.75%
Epoch [25/25] ,Dev_Loss:1.1254,  Dev_Accuracy:82.48%
```

Graphs:




Parameters for pos model:

From Pre-train Embbeding layer:

Number of epochs : 24

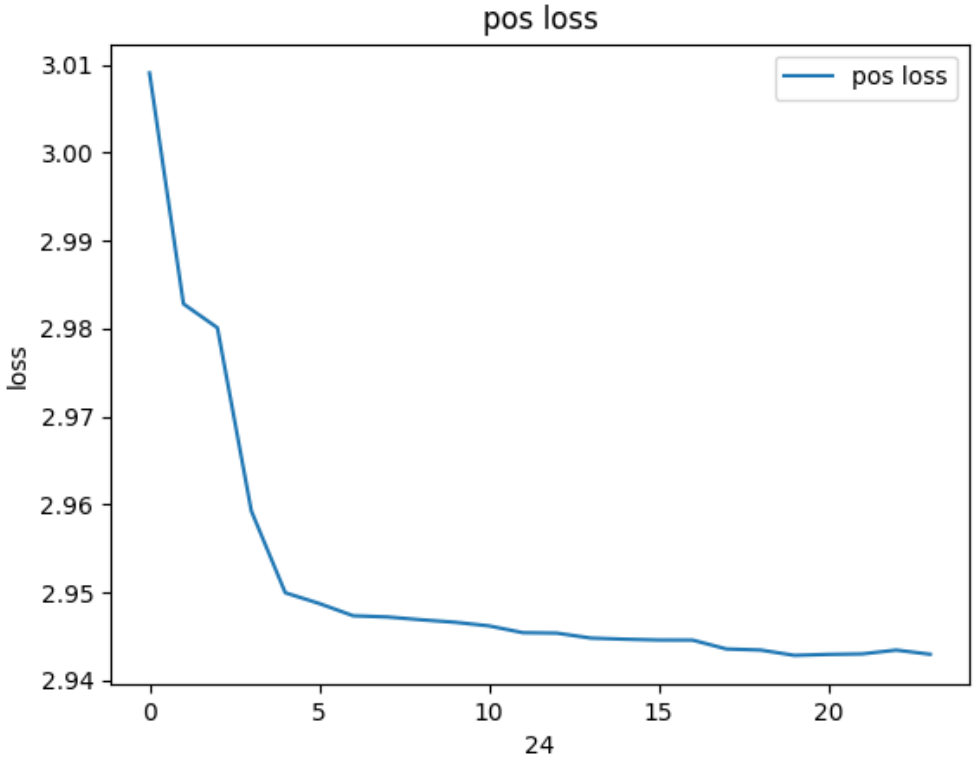
Size of the hidden layer : 80

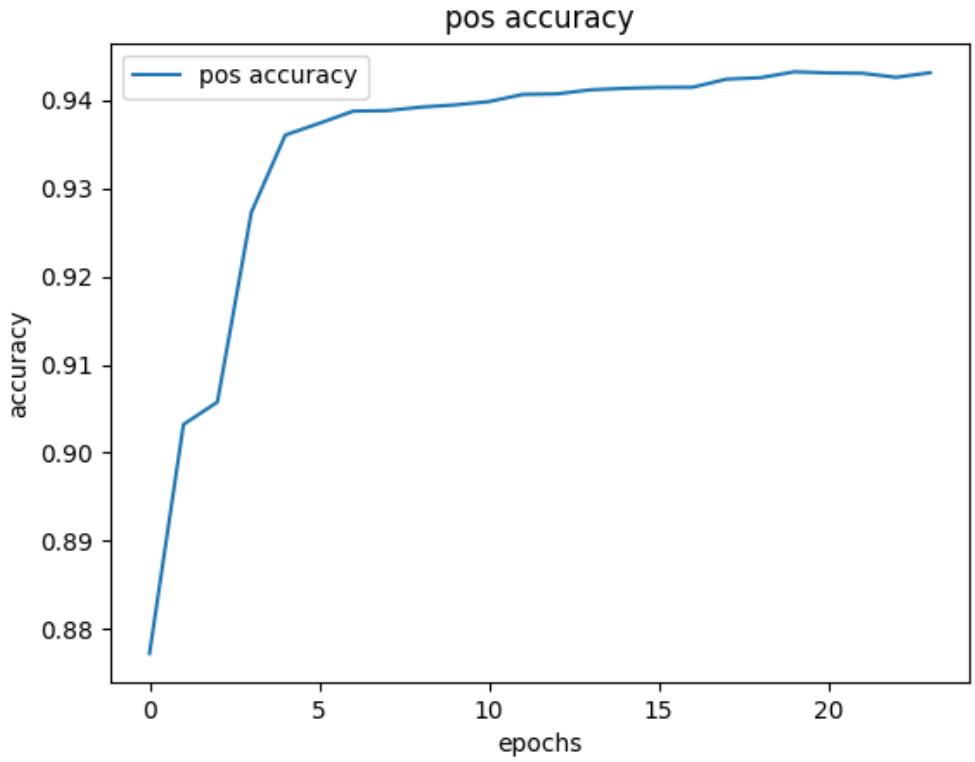
Learning rate : 0.01

Batch size: 1000
Optimizer: Adam

With those parameters we reached:

```
Epoch [24/24], Train_Loss: 2.9150, Train_Accuracy: 97.11%
Epoch [24/24] ,Dev_Loss:2.9430,  Dev_Accuracy:94.31%
```


pos loss
pos loss
3.01
3.00
2.99
2.98
2.97
2.96
2.95
2.94
loss
0
5
10
15
20
24


pos accuracy
pos accuracy
0.94
0.93
0.92
0.91
0.90
0.89
0.88
accuracy
0
5
10
15
20
epochs

From randomly initialized Embbeding layer:

Number of epochs : 15

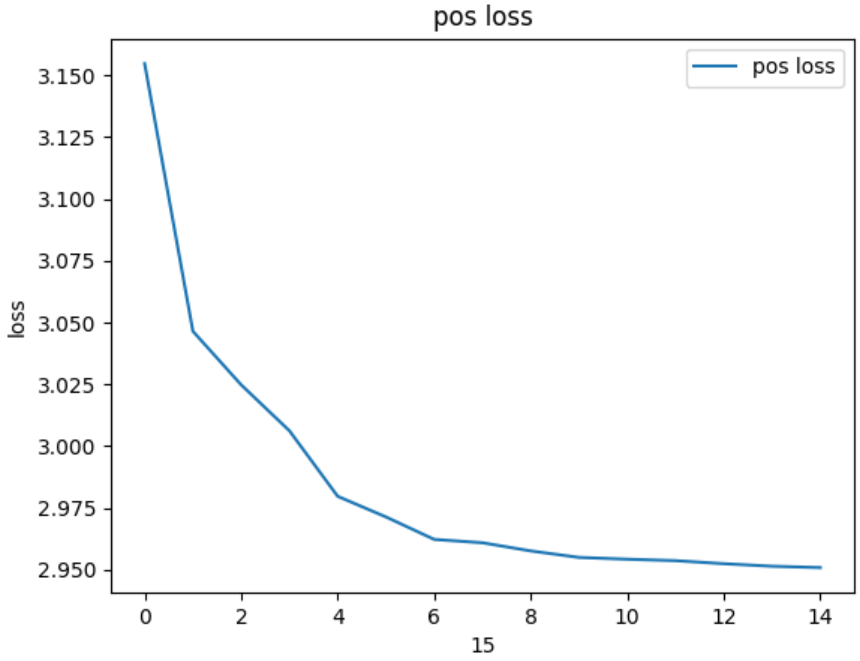
Size of the hidden layer : 60

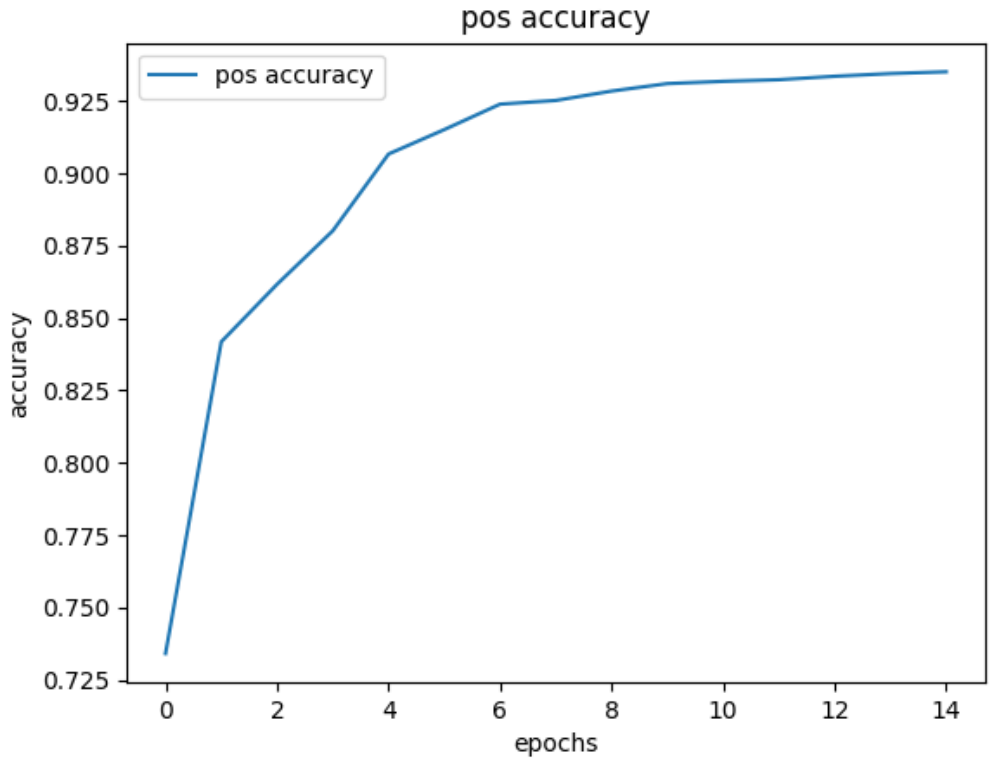
Learning rate : 0.001

Batch size: 1000
Optimizer: Adam
Random initialization of the Embedding layer: Uniform Distribution on [-1:1]

With those parameters we reached:

```
Epoch [15/15], Train_Loss: 2.9218, Train_Accuracy: 96.41%
Epoch [15/15] ,Dev_Loss:2.9509,  Dev_Accuracy:93.51%
```




Our thought and our choice for the sub-words units representation:

First off all, we notice that prefixes and suffixes of all the window surrounding the word can also give information to the network about this word we want to predict.

(for example in POS tags, in the sentence ”I am doing sport” after the suffix -ing- the next word is probably a Name NN).

That’s the reason why we decided to choose to represent all the prefixes and the suffixes of the words in the Embedding matrix of 50 length vectors.

Thus, for each window of 5 words, we concatenated the vector representation of each words together, and concatenated the vector representation of the prefixes of each word and concatenated the vector representation of the suffixes of each words as well, then we summed those 3 freshly created vector, and give them as input to our network.

Note:

About the vector representation of the sub-words unit, we thought about using only the prefix and suffix of the word we want to tag.
The first option was to represent this prefix and suffix as 50 vector length representation as well, and concatenate them with 0s so the tensor size will all be of length 250. But in this case, summing a “regular tensor” with 2 tensors of the form 00..0xxxx00…0 was not natural and would loss a lot of information we want to give to the net.

Another option a 250-length vector each, but this would create 2 different sizes of embedding matrix. So we did not realized this representation neither.

For words that are less than 3, we added special suffixes <suf> and prefixes <pref> that indicate that there are no prefix/suffix in size of 3 and we allocate random vectors for them. It helps to handle cases with words such "a," , "on", which still contain some useful information.

**Analysis about the results :**

There were slight increase in accuracy that resulted from adding sub units (in ner tagging by 1.5% average improvement) . We have concluded from it, that sub units(prefixes and suffixes) give useful information for named entity recognition tag and pos tagging.. Also, we have noticed that the accuracy increase significantly quicker (see the graphs) . According to our opinion: there are two reasons for faster convergence:

First one is that whenever we used pre trained vectors, the model converged quicker. It happens because model start learn from some good starting point. If we use randomly initialized vectors, it have to relearn the same things, therefore it takes more time to reach the same accuracy.

The second reason is a using of suffixes and prefixes. Model gets more useful information about a word per one training example. That's helps to it to correct weights more correctly per one example.

Between the pre-Embedding layer and the sub-word units, we didn’t find a complementarity between them since there are no better result.