# Part 4

## Name

Simon Raviv

## Best Parameters

### NER Tagging (for both with/without pretrained embedding)

- Epochs: 16
- Hidden layer size: 64
- Batch size: 64
- Learning rate: 0.001
- Learning rate scheduling gamma: 0.85
- Learning rate scheduling step size: 7
- Dropout probability: 0.2
- Weight decay: 0.0001

### POS Tagging (for both with/without pretrained embedding)

- Epochs: 36
- Hidden layer size: 64
- Batch size: 64
- Learning rate: 0.001
- Learning rate scheduling gamma: 0.95
- Learning rate scheduling step size: 15
- Dropout probability: 0.2
- Weight decay: 0.00001

## Consideration

One decision I had to make is weather to add padding character for prefix/postfix smaller than 3 letters or not. I have chosen not to add the padding, since I didn't see the real need for it. From my perspective it could have harm the training since the padding characters wouldn't appear in the real words.

## Accuracy:

The final accuracy at the end didn't show too much of a difference. The difference was in the training process itself. See Results Analysis section for more information.

### NER Tagging:

The dev accuracy (without considering common tag O) is ~81.7 [%] without pretrained embedding, ~82.3 [%] with pretrained embedding.

### POS Tagging:

The dev accuracy is ~94.7 [%] without pretrained embedding, ~94.9 [%] with pretrained embedding.

# Results Analysis

To get baseline results, I did the training using the sub-words embedding for both without/with pretrained embeddings with the exact same hyper parameters. This is to have first analysis without the impact of the hyperparameters optimization. After the first analysis, I did hyperparameters tuning (manual) using multiple experiments I conducted on bot tasks and got different results, better a bit in the accuracy from previous parts, but also much different in the training procedure.

## Baseline

### Without Pretrained Embedding

#### *NER Tagging*

There was a small difference in the final accuracy, without sub-word embedding it was ~82 [%], with sub-word embedding it was ~79[%] in the dev set. In the train the last several epochs for the baseline were better, more than 99 [%], while for the sub-word embedding it didn't get to 99 [%].

Using the sub-word embedding the network can get higher accuracy faster. The several first epochs got higher accuracy comparing to without sub-word embedding, 13,44,70 vs 28,31,73 percent.

In summary, the overall accuracy didn't improve (even a bit lower), the convergence duration was faster.

#### *POS Tagging*

There was a small difference in the final accuracy, without sub-word embedding it was about 0.5 [%] higher compared to with sub-word embedding.

In the POS tagging, the conversions effect was the opposite. Using the sub-word embedding the network the network convergence slower. The several first epochs got lower accuracy comparing to without sub-word embedding, 85,86,89 vs 83,85,85 percent.

In summary, the overall accuracy was about the same, the convergence duration was slower.

### With Pretrained Embedding

#### *NER Tagging*

The final accuracy for both scenarios is about the same 80 [%] – 81 [%]. In the train the last several epochs for the baseline were better, more than 99 [%], while for the sub-word embedding it get to 99 [%] only on the last epoch.

Using the sub-word embedding the network can get higher accuracy faster. The several first epochs got higher accuracy comparing to without sub-word embedding, 47,61,74 vs 49,71,76 percent.

In summary, the overall accuracy didn't improve, the convergence duration was faster.

#### *POS Tagging*

The same affect was in this case as well, the overall accuracy didn't improve, the convergence duration was slower for the sub-word embedding network.

#### *Summary*

The combination of pretrained embedding with sub-word embedding is better for convergence performance, pretty much the same for the final accuracy, compared to the other cases. This is since the

pretrained embedding reduce the time needed to train the embedding vectors and the sub-word embedding add additional level of information about the same token. It is a variation of CBOW, where we add 2 features the postfix and prefix and sum it to the word. The reason it was better for the NER task versus the POS task, is that when we do sum, we lose the information about the ordering which is not important for NER, but it is important for POS.

## Optimized

### NER Tagging

NER tagging didn't require too much optimization for final accuracy with the same parameters. Changing a bit, the learning rate decay and batch size, got to pretty much the same final accuracy. See Best Parameters section.

### POS Tagging

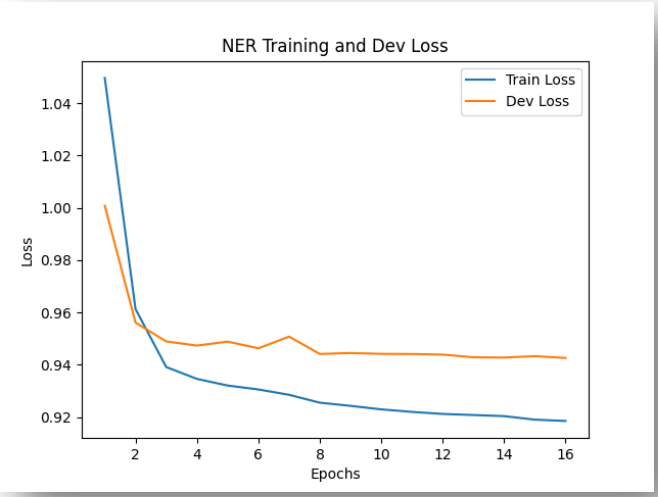
The main difference helped boost optimizing the results for POS task is the learning rate parameter. Reducing it helped by quite a lot for the convergence speed. Reducing from 0.8 to 0.3 to 0.1 helped to get faster convergence with each update. This had the same effect for both with/without pretrained embeddings, using the pre-trained embedding improved the speed as before. I think its due to the fact the pretrained embedding reduces the need to train the embedding vectors.

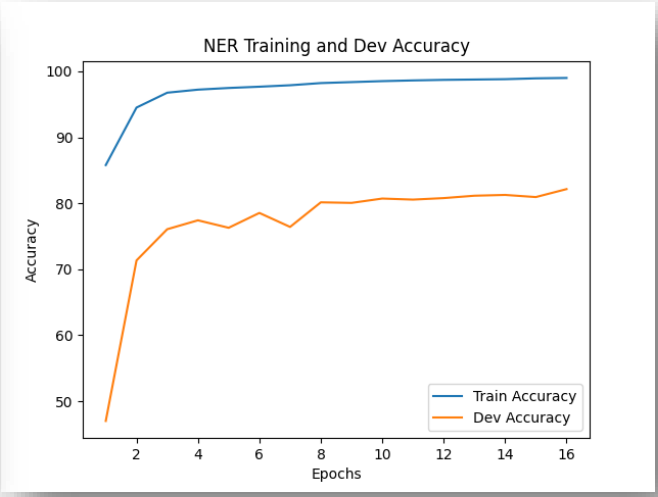
# Training Graphs

## NER Tagging

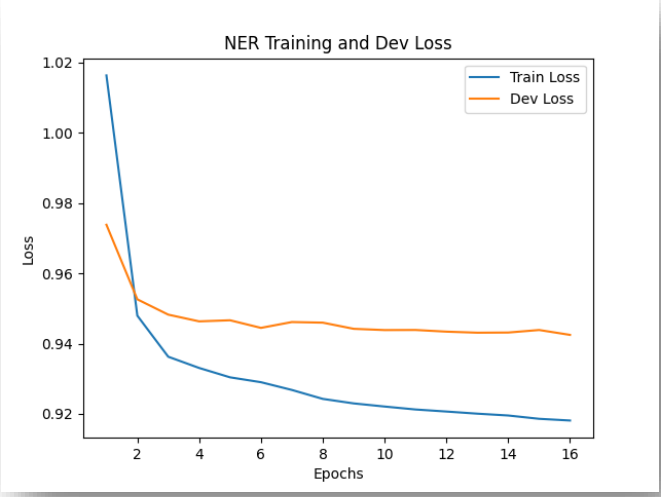
Note: The calculation for the dev done without considering the common label ‘O’ as requested. The calculation for the train done with it.

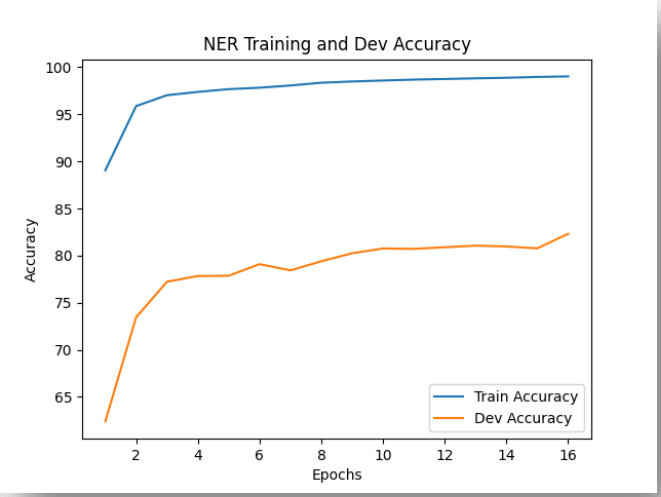
### Without Using Pretrained embedding



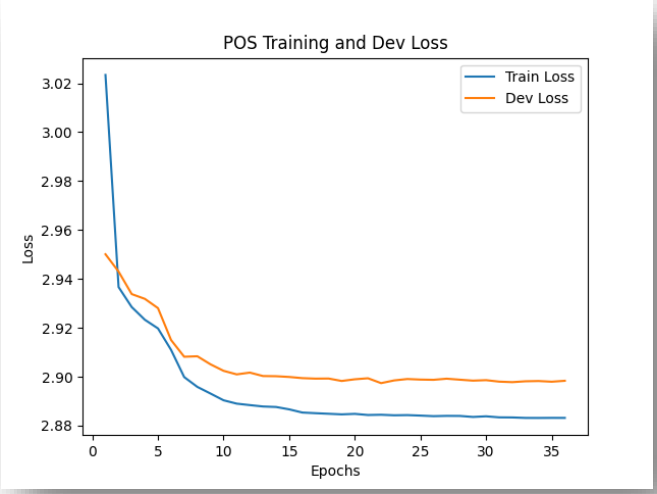

### With Using Pretrained embedding

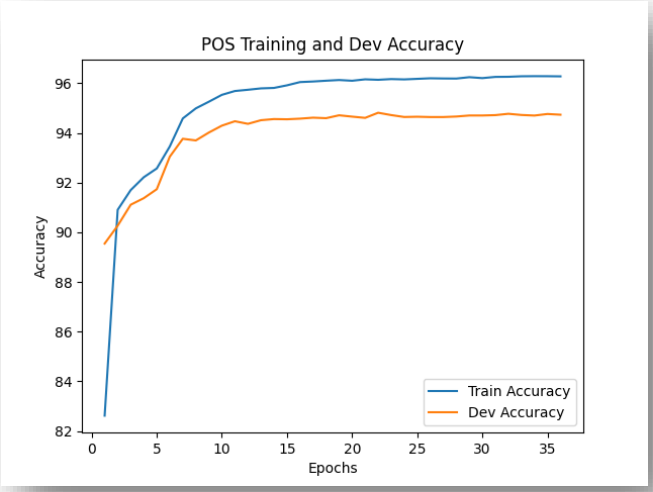


## POS Tagging

### Without Using Pretrained embedding




### With Using Pretrained embedding