

Assignment 1 – Part 5

Submitters:

Ori Braverman 318917010

Elie nedjar 336140116

Introduction

In Part 5 of the assignment, we integrated a CNN to achieve some knowledge by incorporating sub-word units. We followed the work of Ma and Hovy (2016) to improve our POS and NER results.

The input for the CNN

Character Embeddings

Each character in a word is represented as a one-hot vector or an ordinal number (ASCII) indicating its position in the character set.

So, to represent all the possible characters we decided to use the size 256 to the character Embeddings (we know the article provided 100, but we wanted to make sure no letter missing)

We also initialized it uniformly according to the article.

News (Mikolov et al., 2013). To test the effectiveness of pretrained word embeddings, we experimented with randomly initialized embeddings with 100 dimensions, where embeddings are uniformly sampled from range $[-\sqrt{\frac{3}{dim}}, +\sqrt{\frac{3}{dim}}]$

The processing steps:

1. Each word was converted into a sequence of characters and mapped to its corresponding embedding.
2. Words are padded to a uniform length ($\text{max_word_length} + 2 * \text{pad_size}$) to ensure consistent input size for the neural network.
3. The padded character sequences are reshaped into $[\text{batch_size}, 1, \text{max_word_length}, \text{embedding_dim}]$ to match the input format required by the convolutional layers.

Model Architecture

CharCNN Module:

First, there is a convolutional layer over the character embeddings.

Then, ReLU activation function followed by max pooling to extract essential features.

Outputs a feature vector representing character-level information for each word.

Tagger4 Model:

The difference from the other taggers is that now the CNN is concatenated into the pretrained word embeddings.

Hyper-parameters Exploration & Analysis

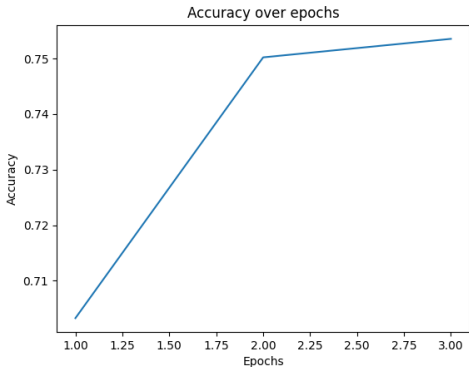
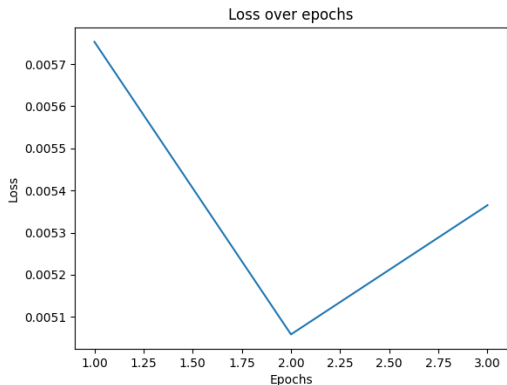
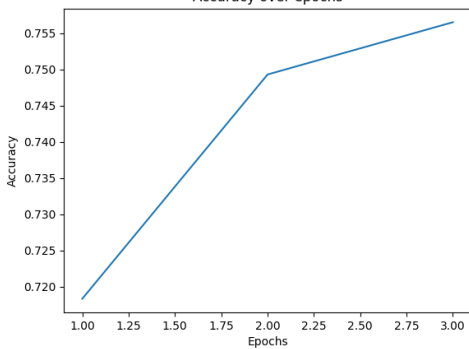
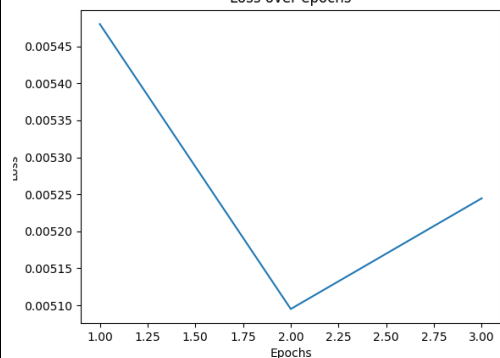
Questions

Start with the initialization, number of filters, and window-size as appears

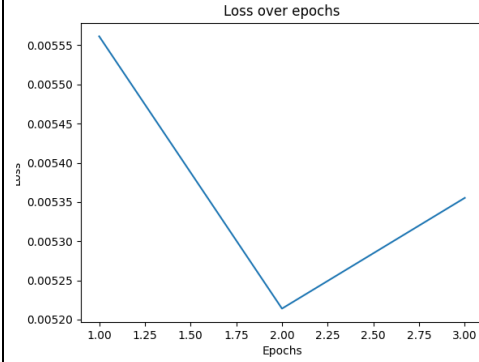
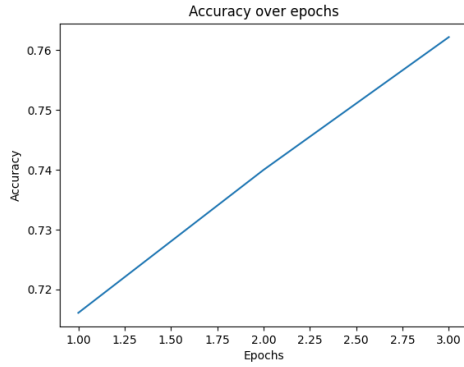
in the Ma and Hovy paper. How does this method compare to what you implemented in Part 4?

What happens when you try fewer filters? When you try

more filters? What happens when you try different window sizes?

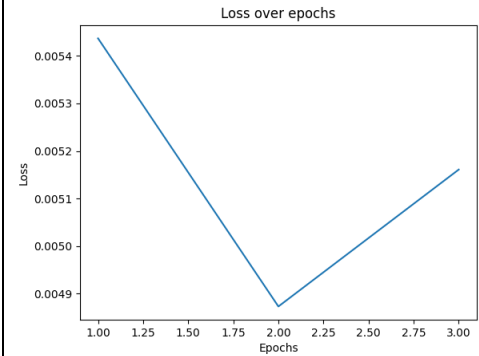
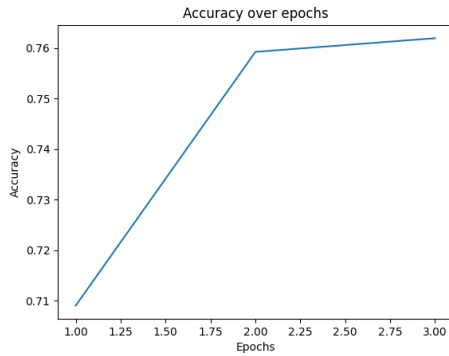
Hyper-params	Dev_ner_accuracy	Dev_ner_loss
'num_filters': 30, 'window_size': 3		
Epoch 3/25 - Avg. Loss: 0.0632 - Train Accuracy: 0.9267 - Dev Accuracy: 0.7536 - Dev Loss: 0.0054		
'num_filters': 10, 'window_size': 3		
Epoch 3/25 - Avg. Loss: 0.0630 - Train Accuracy: 0.9284 - Dev Accuracy: 0.7565 - Dev Loss: 0.0052		

'num_filters': 20,
'window_size': 3



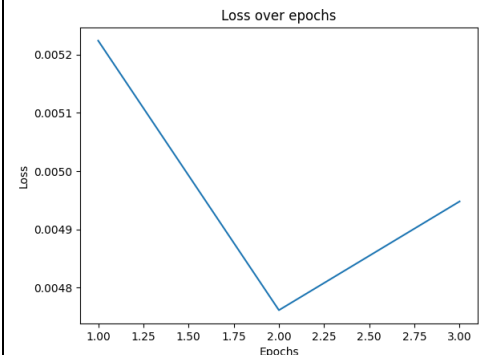
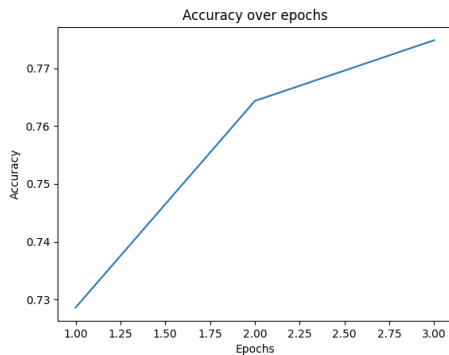
Epoch 3/25 - Avg. Loss: 0.0621 - Train Accuracy: 0.9290 - Dev Accuracy: 0.7622 - Dev Loss: 0.0054

'num_filters': 50,
'window_size': 3



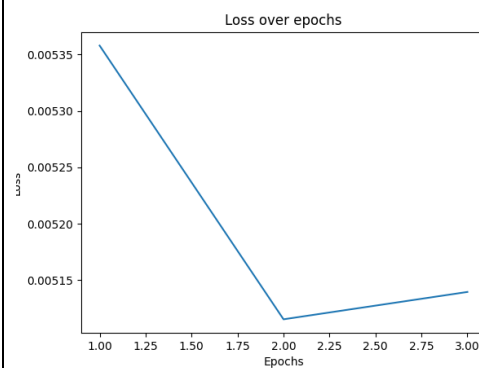
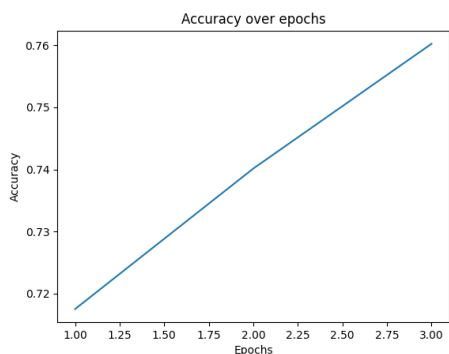
Epoch 3/25 - Avg. Loss: 0.0608 - Train Accuracy: 0.9339 - Dev Accuracy: 0.7620 - Dev Loss: 0.0052

'num_filters':
100,
'window_size': 3

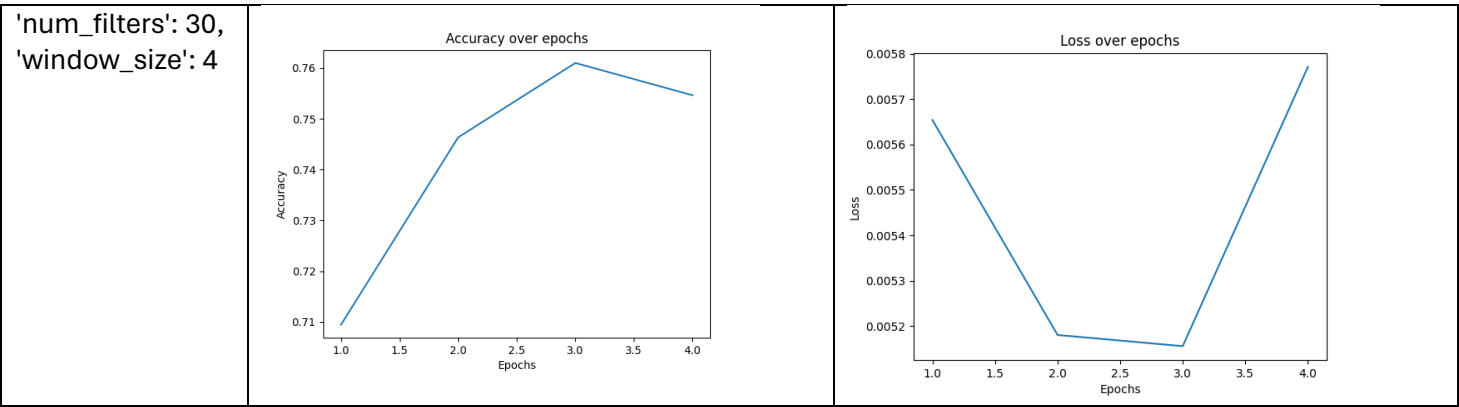


Epoch 3/25 - Avg. Loss: 0.0601 - Train Accuracy: 0.9321 - Dev Accuracy: 0.7749 - Dev Loss: 0.0049

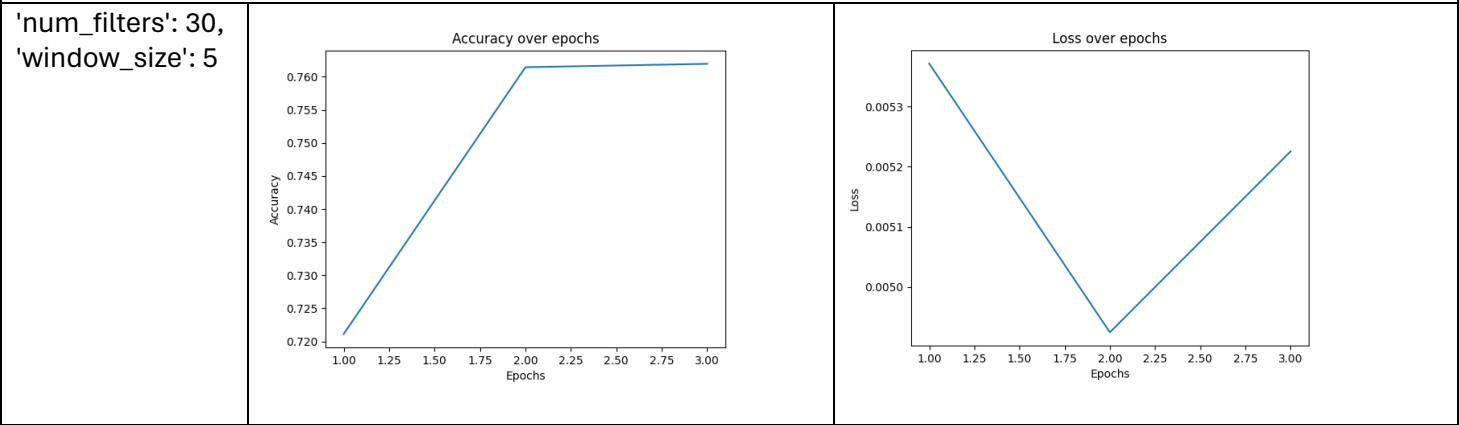
'num_filters': 30,
'window_size': 2



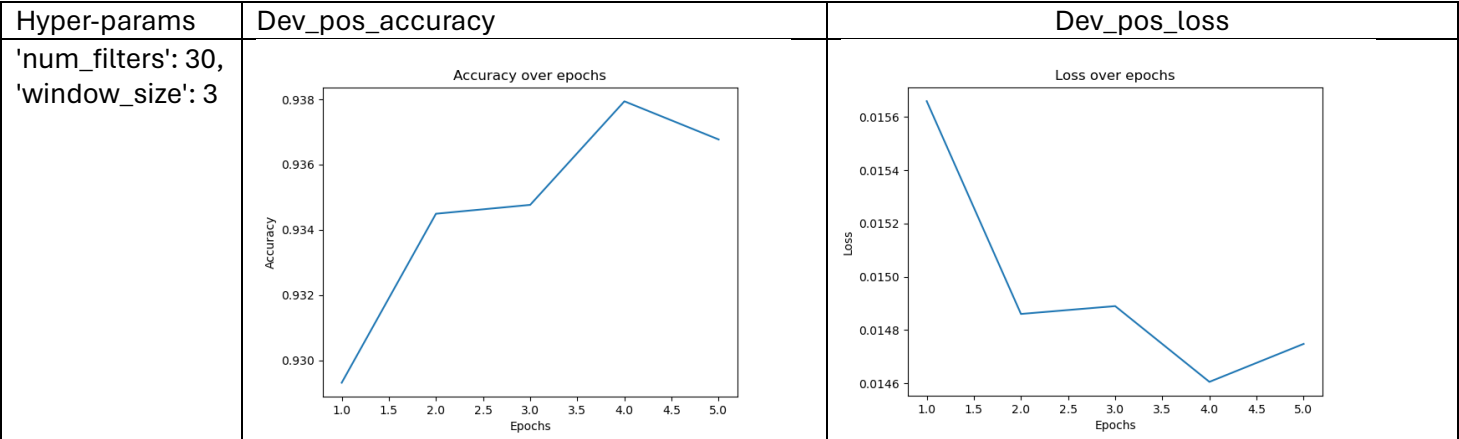
Epoch 3/25 - Avg. Loss: 0.0597 - Train Accuracy: 0.9326 - Dev Accuracy: 0.7602 - Dev Loss: 0.0051



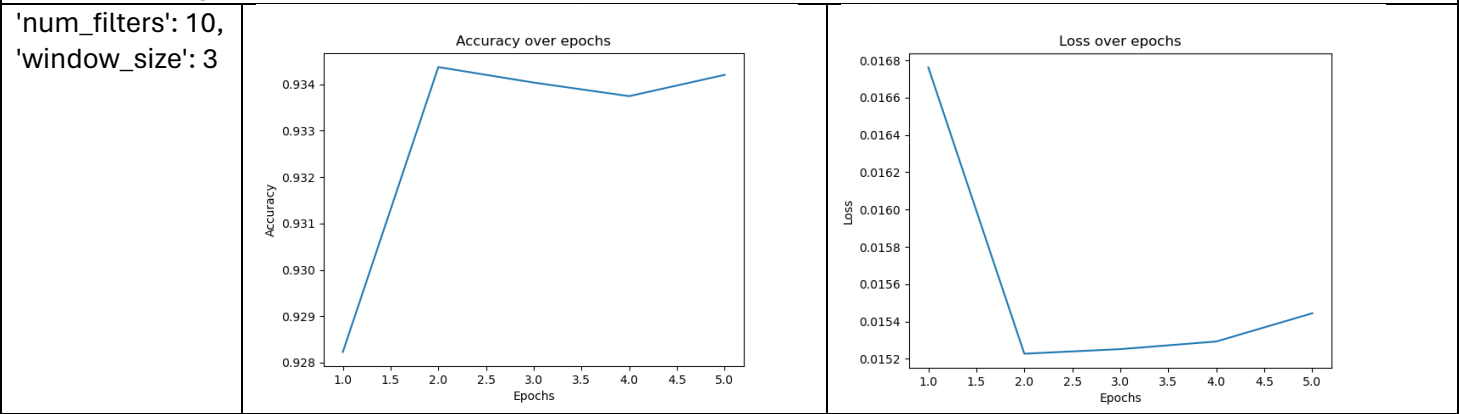
Epoch 4/25 - Avg. Loss: 0.0464 - Train Accuracy: 0.9449 - Dev Accuracy: 0.7546 - Dev Loss: 0.0058



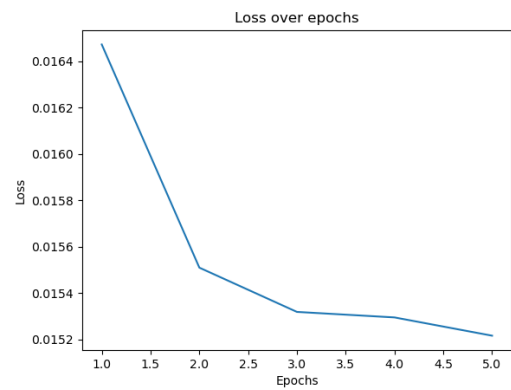
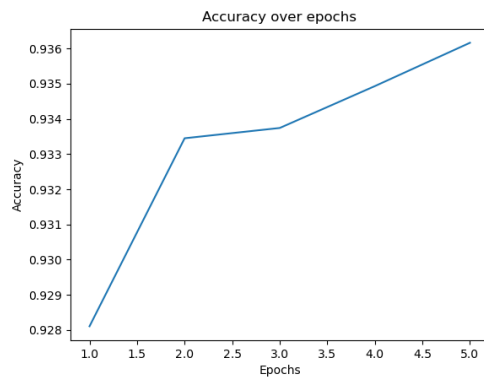
Epoch 3/25 - Avg. Loss: 0.0614 - Train Accuracy: 0.9311 - Dev Accuracy: 0.7619 - Dev Loss: 0.0052



Epoch 5/5 - Avg. Loss: 0.1582 - Train Accuracy: 0.9620 - Dev Accuracy: 0.9368 - Dev Loss: 0.0147

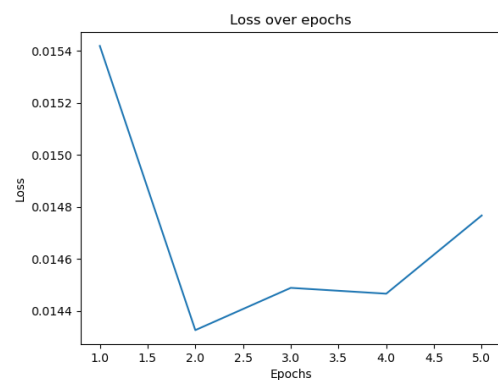
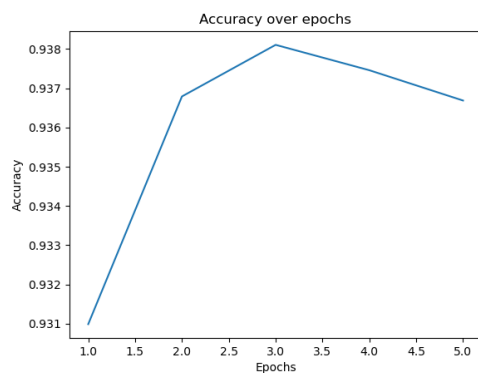


'num_filters': 20,
'window_size': 3



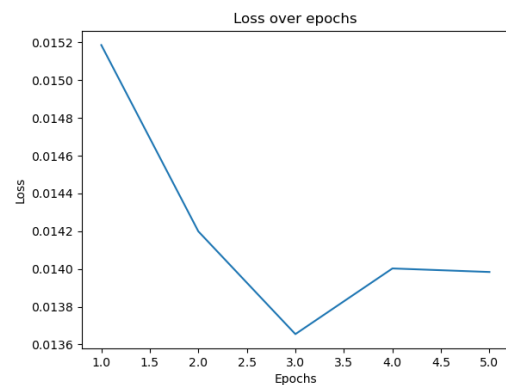
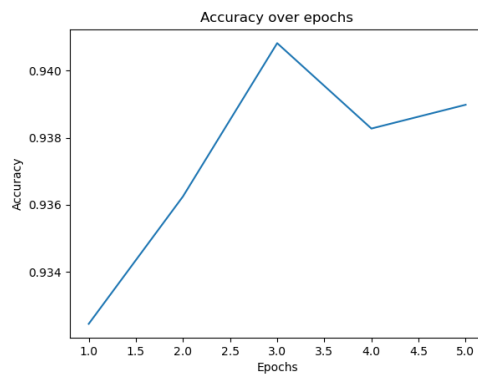
Epoch 5/5 - Avg. Loss: 0.1621 - Train Accuracy: 0.9609 - Dev Accuracy: 0.9342 - Dev Loss: 0.0154

'num_filters': 50,
'window_size': 3



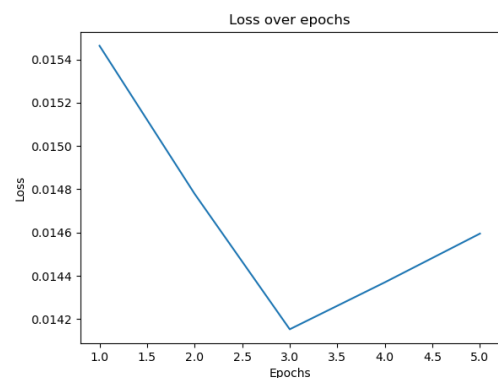
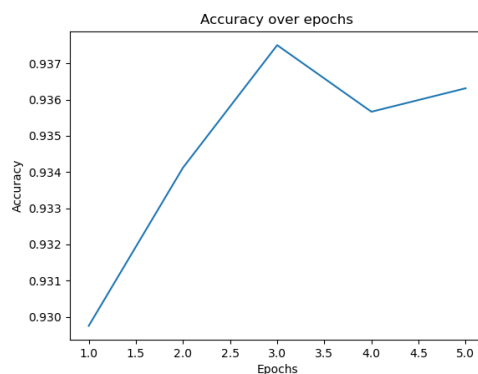
Epoch 5/5 - Avg. Loss: 0.1621 - Train Accuracy: 0.9611 - Dev Accuracy: 0.9362 - Dev Loss: 0.0152

'num_filters':
100,
'window_size': 3

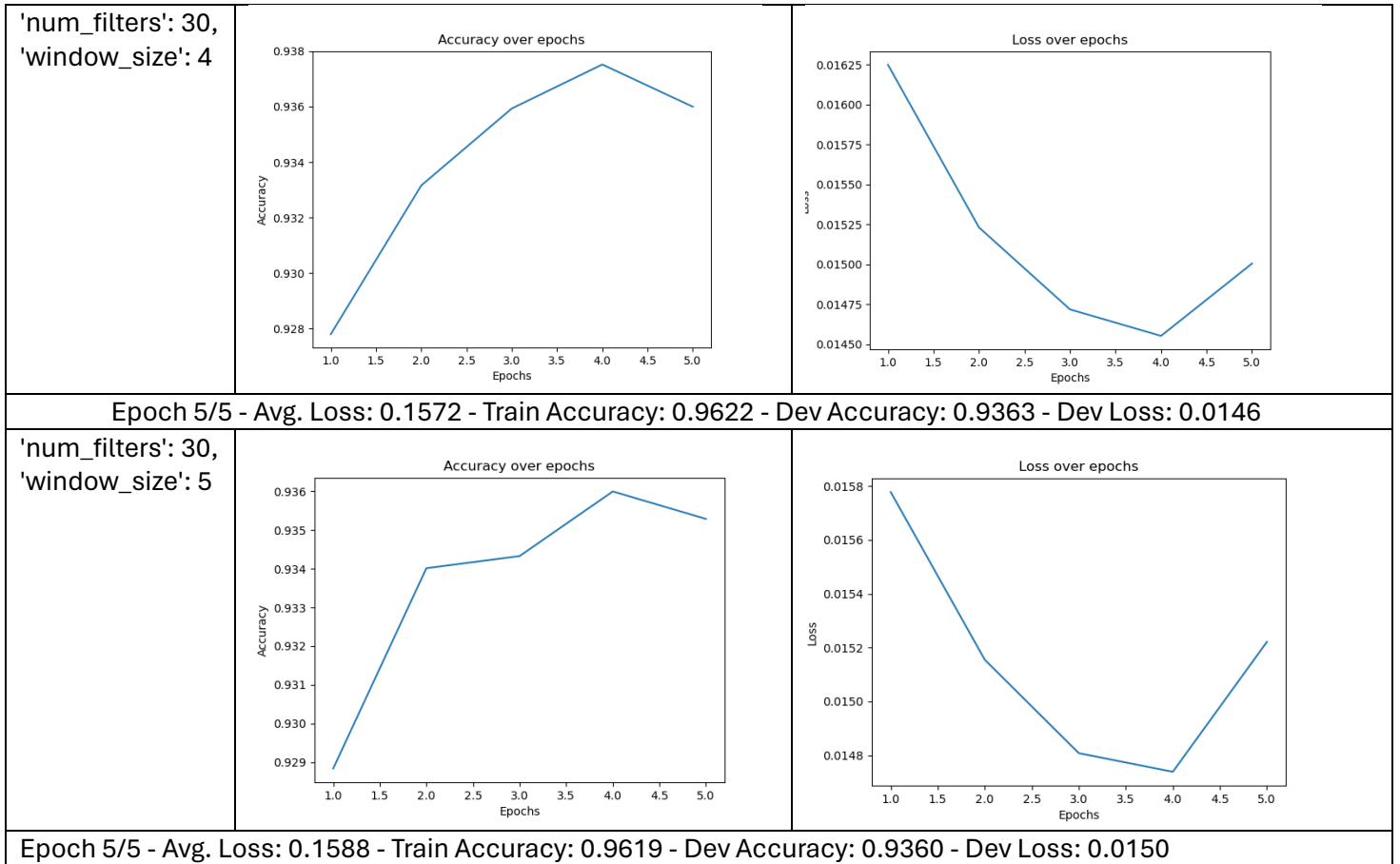


Epoch 5/5 - Avg. Loss: 0.1571 - Train Accuracy: 0.9611 - Dev Accuracy: 0.9367 - Dev Loss: 0.0148

'num_filters': 30,
'window_size': 2



Epoch 5/5 - Avg. Loss: 0.1520 - Train Accuracy: 0.9629 - Dev Accuracy: 0.9390 - Dev Loss: 0.0140



We can learn from the NER results that the most efficient number of filters and the window size was the higher ones, the pos results was similar to each other so there is no preferred settings.

The best accuracy of NER achieved by the model with 'num_filters': 100, 'window_size': 3:

Epoch 3/25 - Avg. Loss: 0.0601 - Train Accuracy: 0.9321 - Dev Accuracy: 0.7749 - Dev Loss: 0.0049

The best accuracy of POS achieved by the model with 'num_filters': 30, 'window_size': 2:

Epoch 5/5 - Avg. Loss: 0.1520 - Train Accuracy: 0.9629 - Dev Accuracy: 0.9390 - Dev Loss: 0.0140