

# Assignment 2 – Window-based Tagging

Daniel Bazar 314708181

Peleg shefi 316523638

## Part 5 – Convolution-based sub-word units

### Architecture

- Both tasks implement the same Network:
  - CNN
    - Embedding layer
    - 1d Conv
    - Maxpooling
  - MLP with:
    - Embedding- word embedding concat to char embedding (from the CNN).
    - one hidden layer
    - tanh activation function.
- The Convolution-based sub-word units' architecture is the following:
  - each word is represented as a **max-pooled vector** over the vectors resulting from a **convolution over the word's characters**.
  - The pre-trained word-vectors and the character-based CNN vectors are combined via concatenation.
- The network trained with cross-entropy loss.
- We Experimented with several network configurations and chose the best configuration based on the DEV accuracy.
- The optimizer is Adam.
  - It achieved faster run times and better results than the optimizer described at the paper.

### Best parameters

- NER:
  - Hidden layer size: 130
  - Dropout probability: 0.4
  - Batch size: 128
  - Optimizer: Adam (Learning rate: 0.0002)
  - Epochs: 4
  - CNN filters: 30
  - CNN embedding dimension: 30
  - CNN kernel size: 3
  - CNN padding size: 2
  - CNN dropout: 0.5
- POS:
  - Hidden layer size: 130
  - Dropout probability: 0.2
  - Batch size: 2048
  - Optimizer: Adam (Learning rate: 0.0001)
  - Epochs: 8

- CNN filters: 30
- CNN embedding dimension: 30
- CNN kernel size: 3
- CNN padding size: 2
- CNN dropout: 0.5

### Considerations

- The Convolution-based sub-word units' method can be combined with the pretrained embedding so we had to consider all the things in the pretrained part:
  - We handle words that appear in the training file but not in the embedding file as we handle those words in part1 (words that don't appear in dev file), we assign them the UNK token.
  - If we used the pre-trained embedding vectors, we transform the training and the dev data to lower case because the embedding vocabulary being lower-case.
  - Because the embedding vocabulary contains special words like "DGDGDGDG", "DG.DG", "+DG", "NNNUMMM", etc. we treated those words as digits patterns.
  - We padded the sentences with SOS (start of string) and EOS (end of string) at the beginning and end of the sentence.
- The Convolution-based sub-word units cannot be combined with the sub-word unit's method, so we prevent this from happening.
- Compared to what we implemented in Part 4 (sub-word units) this method got much better results which were quite close to our naïve tagger with the Xavier init.

Testing different CNN parameters on the NER task:

Window size	Number of filters	embed dim	result
3	10	30	81.98
3	30	30	82.19
5	30	20	81.15
5	50	30	81.91

\*MLP parameters: parameters: {'hidden\_layer': 130, 'dropout\_p': 0.4, 'batch\_size': 128, 'lr': 0.002}

### Results

MLP parameters: {'hidden\_layer': 130, 'dropout\_p': 0.4, 'batch\_size': 128, 'lr': 0.001}

CNN parameters: {number of filters: 10, embed dim: 20, window size =3}

- NER:
  - Loss validation: 0.127
  - Accuracy validation: 81.98%
- POS:
  - Loss validation: 0.22
  - Accuracy: 94.71%

Brief analysis filters & Embedding layers.

Just by looking at the filters and the embedding layers we couldn't find interesting patterns.

We tried finding similarities between embedding vectors to another, and between filters.

In addition, to show it in a visual way we can do some kind of dimension reduction like PCA.

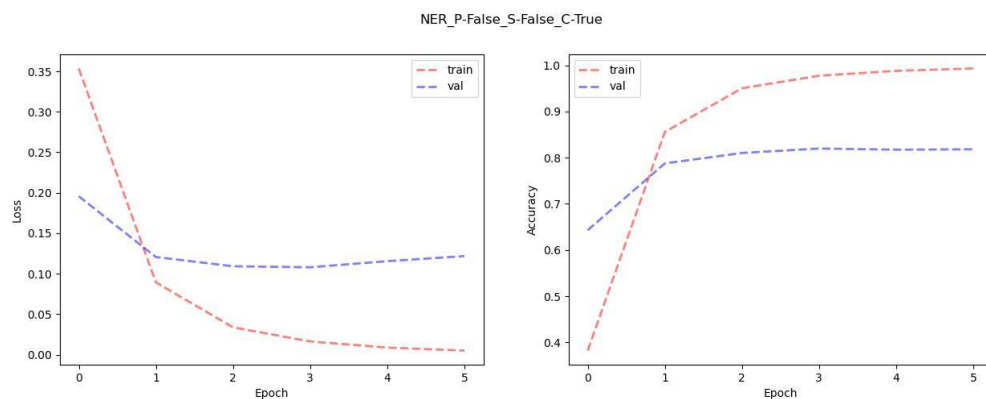
The filters capture local patterns and dependencies within the input data. Each filter acts as a feature detector, scanning the input representations of subword units to identify relevant patterns or structures.

**In our model the filters are designed to capture character-level patterns that are indicative.** For example, a filter may learn to recognize combinations of character n-grams like "ing" or "tion" that frequently appear in verbs or nouns, respectively. It may also learn to detect specifically **prefixes or suffixes** (as we try to detect in part 4) that provide useful information. For instance, a filter might focus on the suffix "-ment" to identify noun forms.

The learned filters for POS tagging and NER can be both similar and different. **Some of them are similar and some are different.** The filters share the objective of capturing local linguistic patterns and dependencies, such as prefixes, suffixes, and character-level patterns. However, there are some differences between the filters used for POS tagging and NER. The filters for POS tagging pay more attention to things like grammar and sentence structure, while the filters for NER pay more attention to finding and understanding named entities.

Graphs

## NER



## POS

