ID: 011862141

# Part 3 networks details

**Part a:** The word embedding vector size is 128. The biLSTM internal state is a 50-vector. This size for the biLSTM is used for all subsequent networks.

**Part b:** The character embedding size is 40. The char LSTM’s hidden state’s size is 50

**Part c:**

* The prefix, suffix and words used 3 different embedding vectors. For example, the ‘per’ in ‘person’ was trained separately from the ‘per’ in ‘upper’ and separately from the word ‘per’.
* Prefixes and suffixes were not extracted from words shorter than 5 letters – due to the assumptions prefixes/suffixes of short words are arbitrary.
* The embedding size for suffix was identical to word embedding size – 128.
* Combining the embeddings during training:
  + The 3 embedding vectors were summed and passed on to the next layer. If the word was shorter than 5 characters, only the word embedding vector was extracted and passed.
* Combining the embeddings during prediction:
  + If the prefix or suffix of a word were not present in the prefix- and suffix- vocabularies collected during training, then it was represented by just its word embedding vector.
  + If the word was not encountered, but either its prefix or suffix were, then it got the work embedding for ‘UNK’ with the prefix/ suffix embeddings added.

**Part d:** The additional linear layer size was 60, so it was represented by a 60x178 matrix and a 60-entry bias vector. The first runs of classifier D were performed without the linear layer; the 178 vector was passed directly to the next stage (the 2-layer biLSTM). When this was detected and fixed, the results of the intended classifier were, in fact, slightly worse (by about 0.4% accuracy) than the initial results (without the linear layer). So the code was reverted. Few facts should be noted regarding this finding:

1. The “best” classifier we arrive at for each class is just one of many possible classifiers with the same topology, due to random decisions taken in the training process (namely initialization and sample ordering). Especially if the scores are very close, working off of a single run might not be “representative” of the topology’s “maximum ability”.
2. The dev set is just one particular challenge; the classifiers may get different scores for different tagging tasks.
3. A more sophisticated classifier does not always perform better than a similar, less sophisticated classifier.

## Treatment of unknown words during training and prediction

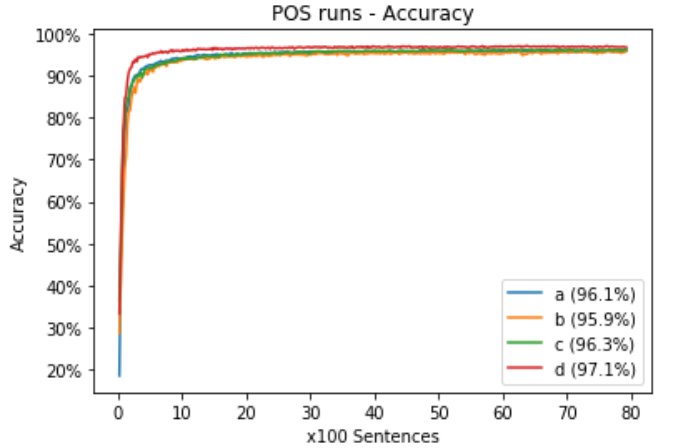
During training, the least common 1000 words were used as ‘UNK’ candidates. When any of these words was encountered, it was dropped with a probability of 20%, in which case it was replaced with the embedding vector corresponding to ‘UNK’, in order for that vector to be trained. 80% of the time, these words were treated normally.

During prediction, any word that was not present in the training vocabulary, used UNK’s embedding.

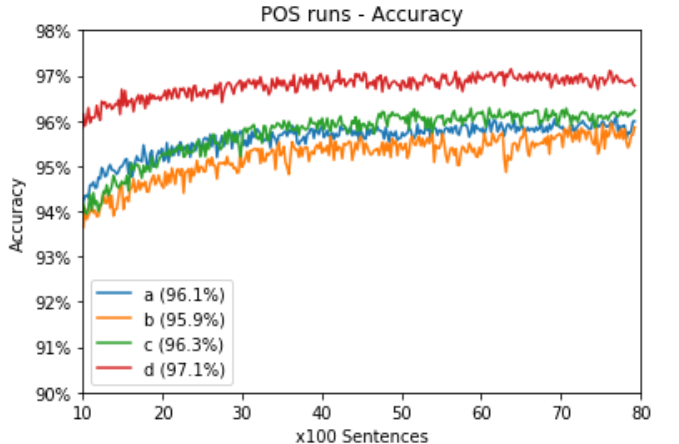
The learning rate was controlled with dynet’s implementation of Adam Scheme, with the default parameters.

# Accuracy graphs

## POS tagging task



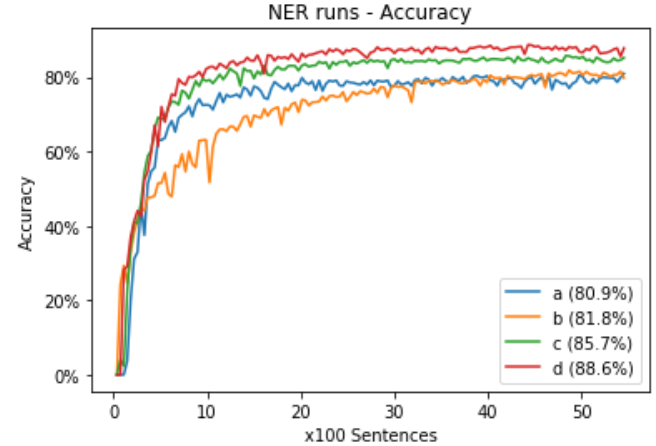
Same graph with “zoom in” for more detailed convergence:



## Notes on POS convergence

Configuration D displays superiority from early on. A dominates B throughout the training process, albite by a small margin; C improvement is more gradual and it ends up located above both A and B.

## Named Entity Recognition task



## NER convergence notes

The NER scores were calculate with “O-tag can only hurt” approach. Again, D gets the best scores from early on. B in this case performs better than A, again displaying the most gradual learning process of the four, probably due to the character LSTMs requiring more training cycles to converge. C is located between B and D.