# 3. Report

3.1. The description of your tagger, including the architecture, the hyperparameters, etc.

For LSTM tagger, the architecture of the model is similar to the one introduced on PyTorch website (https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html). During the data pre-processing, each sentence and its respecting tags is tokenized into two lists. After tokenization, every word and tag are then represented by an index. Therefore, we have 2 2-d lists for indexed words and indexed tags respectively. For the training part, indexed words are put into the LSTM model as input, while indexed tags are taken as the output.

Architecture of the LSTM model:

- embedding layer – 64-dimensional embedding
- output layer – 0.1 dropout, 32-dimensional dense, relu activation, 0.1 dropout, 9-dimensional dense, softmax activation
- optimizer = SGD
- learning rate = 0.5
- no. of epoch = 30
- loss function = NLL loss

For transformer model, it is adapted from a keras website (https://keras.io/examples/nlp/ner_transformers/). Just like the LSTM tagger, each sentence and its respecting tags is tokenized into two lists, but in different format from the LSTM task. Then token embedding and positional embedding is done to make each sentence the same size. All words and tags are represented by an index. In order to deal with the low frequency words, only the most common 19998 words can be trained while the other words are taken as “unknown words”. During the training part, the words are treated as input and the tags are treated as output.

Architecture of the transformer model:

- embedding layer – 32-dimensional positional embedding, 32-dimensional token embedding)
- transformer layer – 32-dimensional 4-heads attention, 0.1 dropout, 1e-6 normalization, 32-dimensional dense, relu activation, 32-dimensional dense, 0.1 dropout, 1e-6 normailization
- output layer – 0.1 dropout, 32-dimensional dense, relu activation, 0.1 dropout, 9-dimensional dense, softmax activation
- optimizer = adam
- learning rate = 1e-3
- no. of epoch = 10
- batch size = 32
- loss function = cross entropy loss with modifications that will ignore the loss from padded tokens

3.2. The description of your settings, including the hyperparameters you have tried,

the performance of your tagger on the dev set, etc.

The hyperparameters are designated for this task. The embedding size for both models and the no. of attention heads are large so as to make sure every sentences are embedded. The size for hidden layers of both models are also large so as to match the embedding layer size. Dropout is performed to prevent overfitting and redundant nodes in the layers. Other hyperparameters are set by trying different values and see which one yields the highest testing accuracy. Due to time limitations, not all range of values could be experimented.

3.3. The discussion and table of your results, including the performance of your tagger on the test set, the analysis of the results, etc.

The best accuracy of the LSTM tagger and the transformer tagger on the test data are 46.88% and 91.41% respectively.

The accuracy of the LSTM tagger are 91.17%, 58.17% and 46.88% on training, valid and test dataset respectively. It might indicate that there is overfitting on the training data.

On the other hand, the accuracy of the transformer tagger are 98.48%, 93.58% and 91.17%. It might indicate that the model is well trained. Hence the test.out file is generated from the transformer tagger.