Named-Entity Recognition on Social Media Data

`github.com/bw-tech/`

Bailey Wei
bw489@cornell.edu

1 Introduction (6pt)

The goal of this project is to build a named-entity recognizer for Twitter text. In particular, we are advised to focus mainly on high accuracy and speed. The data supplied to us aggregated from Twitter tweets and has matching labels for named-entities. There are tokens B for the start of a named-entity, I for continuation of a. named-entity, and O for any other token.

For this project, I implemented two different approaches. The first approach was a through using only PyTorch's LSTM infrastructure to create a neural network to classify the 3 different tokens. This first approach was motivated by focusing on speed, the network was fast to create. The second approach was through and implementation of HuggingFace's *BertForTokenClassification*. I was motivated to chose this method to focus on obtaining high accuracy.

For both experiments, I stuck with the basic data we were supplied. I ran multiple experiments with both models. The first model I experimented with different network structures, embedding dimensions and methods, loss functions, and methods to deal with unknowns words. For the second approach I experimented with different padding lengths, batch sizes, epochs, and loss functions. The F1 test scores for the first and second models are 0.42124 and 0.62284 respectively.

2 Task (5pt)

Let T be the set of all possible tweets, t. For each tweet, t, it has a length of N with a span of tokens from x_0 to x_N. Furthermore, there is a tag matching each token, y_0 to y_N. We must accurately and quickly match a token, x_i, with its corresponding tag, y_i, in the set of $Y = (B, I, O)$. More specifically, we have to match an example tweet, *'Cornell Tech is great'*, to *B,I,O,O*.

3 Data (4pt)

We utilized Twitter tweet data supplied by the assignment only. Table 1 breaks this down. Furthermore, vocabulary size, and document length is detailed in Table 2

Data	Size
Training	2394 Tweets
Development	959 Tweets
Test	2377 Tweets

Table 1: Data Sizes

Data	Size
Vocabulary Size	10,586 Unique Words
Document Length	Not Relevant

Table 2: Vocabulary and Document Sizes

These statistics were computed while pre-processing and cleaning the data. In more detail, I converted hashtag, twitter mentions, punctuation, hyperlinks, digits, and rare/unknown words to their own tokens. Furthermore, for the HuggingFace implementation, I had to add padding to ensure each tweet is the same length. As a logical move, I set this to be 140 words long, the maximum number of characters for a tweet being 140 characters in the past. Following this, the BERT model requires a $[CLS]$ and $[SEP]$ tag for each sequence. After pre-processing I reached the sizes mentioned in Table 3

Data	Length of Vocab
Vocab.	8301

Table 3: Post Pre-processing Sizes

4 Model (25pt)

For the first model, it is a neural network model with an initial embedding layer, followed by three individual LSTM layers, and then two fully connected linear layers. This approach was built to be a relatively quick to run model to obtain respectable results. Further experimentation was done to add a max pooling layer and implementing drop out. However, these additions resulted in

worse results. The model infrastructure is located below:

(embedding): Embedding(8301, 4000)
(lstm1): LSTM(4000, 2000)
(lstm2): LSTM(1000, 500)
(lstm3): LSTM(500, 200)
(fc1): Linear(200, 100, bias=True)
(fc2): Linear(100, 3, bias=True)

For the second model, we fully utilized HuggingFace's *BertForTokenClassification*. This is a BERT transformer with a final linear layer added onto it for output calculations. The architecture for this pretrained model is much more sophisticated than the first one. There are 12 attention heads, 12 layers, and 768 embedding dims. Below is more detailed infographic of the model structure.

Inference The inference process is very standard. First we process the test data in the same methods as our development data, then replace both the tokens and tags with their respective vocabulary index values, created from a dictionary. Following this, the data is sent to the embedding layer of our model to then return a final score. This highest score is converted to the most likely prediction. This methodology is very similar to both models. However, the second *BertForTokenClassification* model requires addition process sing in the form of marking the each sentence. Furthermore similarly to the first model, we have to pass the outputs into a final soft max layer.

Unknowns and Rare Words We handle unknown words in the first model by first processing all sub-groupings of unique twitter information. Every hashtag, hyperlink, digit, punctuation, and mention are all classified with their own special token. Furthermore, anytime a word appears infrequently, less than twice, then it would be given a rare tag. This process gives significant performance boost while being very fast to run. For instance, I trial run of a substring checker and synset resulted in several hours being required to process.

5 Learning (10pt)

For our learning, I utilized Cross Entropy Loss with experiments of either ADAM or SGD. This is essentially the soft max equation. For unknown or rare tokens, they are specified in one of the earlier define classes. If a word appears to match one of the specified classes, they are then changed to become so. If not, then the word is forced into the catch-all rare tag. This approach is a great balance between information loss and performance because the model has the ability to now learn when and why a named entity would become one of these tags. Furthermore, with the reduction to vocabulary, our model is able to run much quicker. As an example, given the tweet, *@CornellTech is a great NYC*, our model would be able to translate this into *-MENTION-, is, a, great, -HASHTAG-*, which ends up as *O, O, O, O*.

6 Implementation Details (3pt)

The first model has the following parameters: embedding layer dim. size, LSTM layer dim. size, learning rate, momentum, and number of epochs. For the second model: maximum sequence length, batch size, number of epochs, and learning rate.

7 Experimental Setup (4pt)

The data for the experiments was split with a training, development and testing set. The train set was used for initial model evaluation. Following this, the development set was used for further hyper-parameter tuning. The test set was used to only submit final results of the model. F1 score along with precision and recall were used for evaluation.

8 Results

Test Results (3pt) The following Table 4 shows the final test results. The first method was a result from utilizing a fast and basic model of 1 LSTM layer and 1 fully connected layer. This model, (1), had shallow dimensions, a 400 dim. embedding layer, 200 dim. LSTM, and 100. dim FC layer. The second model was detailed in the model section, model (5). The final result is detailed in model (8)

Method	F1 Score
Embedding, 1 LSTM, 1 FC, Rare	0.1584
Embedding, 3 LSTM, 2 FC, Rare	0.4212
Bert, Rare Handling	0.6228

Table 4: Test Results

Development Results (7pt) These are the main experiments conducted. I experimented with additional model complexity for the first

method. First I started with small embedding sizes and only 1 layer for both the LSTM and fully connected layer. However, I noticed the information loss from smaller embedding was impact my model very negatively. Thus, I experimented with larger embedding dimensions. For the second method, I mainly experimented with the padding sizes and number of epochs. Table 5 show cases the related development results.

Model 1 (LSTM RNN):

1. Embedding 400, LSTM 200, FC (100, 3) , LR 0.01, Epochs 10
2. Embedding 1000, LSTM1 500, LSTM2 250, FC (100, 3), LR 0.05, Epochs 10
3. Embedding 1000, LSTM1 500, LSTM2 250, LSTM3 125, FC (50, 3), LR 0.001, Epochs 10
4. Embedding 2000, LSTM1 1000, LSTM2 500, LSTM3 250, FC1 125, FC2 (50, 3), LR 0.001, Epochs 10
5. Embedding 4000, LSTM1 2000, LSTM2 1000, LSTM3 500, FC1 200, FC2 (100, 3), LR 0.001 , Epochs 20

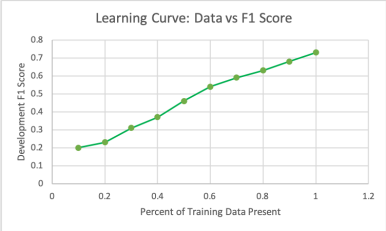
Model 2 (BertForTokenClassification):

1. Max Seq Len 140, Batch Size 20, LR 0.01, Epochs 3
2. Max Seq Len 140, Batch Size 20, LR 0.001, Epochs 5
3. Max Seq Len 140, Batch Size 20, LR 0.001, Epochs 10

Method	F1 Score
1	0.1323
2	0.2013
3	0.2781
4	0.3400
5	0.4203
1 (BERT)	0.6239
2 (BERT)	0.6591
3 (BERT)	0.7287

Table 5: Development Results

Learning Curves (4pt)



From the above graphic, based on the BERT model, we see a steady increase in development F1 scores based on increasing data. From this, we can assume given more data we could potentially reach a better score. However, from the end of the graph, it appears that there the gain in points is slowing.

Speed Analysis (4pt) Detailed in Table 6 are the computation times for the BERT model. Hardware Used: INTEL I5-8600K, NVIDIA GTX1080 8GB, SAMSUNG 500GB 860 EVO SSD also with Google Colab.

Task	Total Time	Per Tweet
LSTM RNN	4:45	9.50 /s
BERT	12:30	29.3 /s

Table 6: Speed Test Results

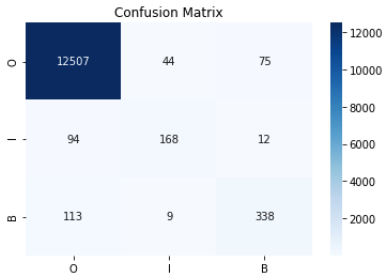
9 Analysis

Error Analysis (7pt) In Table 7 we see the misclassification of 'The Game' as not a named-entity. This is an example of a false negative. The tokens 'The' and 'Game' are both very common words that typically are not named entities. For instance another tweet 'The game we played last night as fun' would have no named entities at all.

Tweet	RT @TheLadBible : The Game showing his love for LFC today! (hyper-link)
True	O O O B I O O O O B O O
Pred.	O O O O O O O O O B O O

Table 7: Error Results

Confusion Matrix (5pt) From the confusion matrix, we see that the model most often misclassified tags are...



10 Conclusion (3pt)

The largest increase in numerical results came from implementing the *BertForTokenClassification*, to reach 0.623 on the test set. Preprocessing gave us better results. Fine-tuning the hyper-parameters within the original LSTM-RNN model, we were also able to reach respectable results of 0.42 on the test set.