CS5740: Project X
Github Repository

Yiling Sun
ys994

1 Introduction (4pt)

In this project, we built a named-entity recognizer for Twitter text. Given a tweet, the tokenizer can identify sub-spans of words that represent named entities: B (beginning of the named entity), I (continuation of a named entity), or O (token that is not part of a named entity). We trained the model with annotated raw tweets data as describe in Section 3.
We built the tokenizer on the BERT (Bidirectional Encoder Representations from Transformers), the state-of-art pre-trained language representation model proved to give faster and better performance on language task.
We experiment on three pretrained model: BERT base, BERT large and RoBERTa. We also experiment on the preprocess of the raw tweets data and fine-tune the learning rate. F1 score is used to measure the performance of model. We achieved the best result 0.63221 on test set and 0.71 on development set with BERT large model and learning rate 5e-5.

2 Task (3pt)

Let $\mathcal{S}$ be the set of all sentences, $\mathcal{X}$ be the set of corresponding named-entity. An tweets $\overline{s} \in \mathcal{S}$ of length $|\overline{s}|$ is a sequence of tokens $\langle\ s_1,\ ...\ s_{|\overline{s}|}\ \rangle$. Each sentence also a corresponding $\overline{x} \in \mathcal{X}$ that is also a sequence of token $\langle x_1,\ ...\ x_{|\overline{x}|}\rangle$. $\overline{x}$ and $\overline{s}$ are one-to-one that $|\overline{x}| = |\overline{s}|$. Tokens of $\mathcal{X} \in \{\text{B,O,I}\}$, which stands for beginning of the named entity, continuation of a named entity, or token that is not part of a named entity respectively. Given a sentence $\langle\ s_1,\ ...\ s_{|\overline{s}|}\ \rangle$, our goal is to generate the sequence s named-entity tags $\langle x_1,\ ...\ x_{|\overline{x}|}\rangle$.

3 Data (4pt)

We have three set of data for training, development and testing. Each dataset contains tweets separated as tokens; each token has a corresponding named-entity. Table 1 gives a shallow statistics of three dataset. For the pre-processing, we handle the username, link and hashtags. All the tokens of username started with was treated as USER. All the tokens of links started with *http* and *www* was treated as www. For hashtags, we experiment on two methods to deal with it: 1. remove the # and keep the words. 2.only keep '#'. The experiment result are shown in 8 We keep the casing of words because it contains information for classification. The emoji was treated as ⟨UNK⟩ in the training. For the unknown words, we handle with WordPiece, the subword tokenizer used by BERT.

Data Size	Train	Dev	Test
Num. of Tweets	2394	959	2377
Num. of Tokens	46469	13360	33354
Tweets Vocab. size	10586	5168	10548
Ave. Tweets Length	19.4	13.9	14

Table 1: Shallow statistics of our three datasets.

4 Model (18pt)

The named-entity recognizer was built on the pre-trained language representation mode BERT introduced by Devlin et al.[1]. BERT is conceptually simple and empirically power. It obtains new state-of-the-art results on eleven natural language processing tasks. The architecture of BERT is a multi-layer bidirectional Transformer encoder with self-attention based on the original implementation described in [2]. The baseline model used is $BERT_{BASE}$ with 12 layers and hidden size 768, and 12 self-attention. We also experiment on other advanced BERT model based the base like $BERT_{LARGE}$ and $RoBERTA$. The performance of each model is listed in section 8. To fine-tune BERT, we simply plug the input tweets and output tags as described in Section 3 into BERT and fine-tune all the parameters end-to-end. Before adding to the model, we adjust our data to fit int the format of model required by adding the special token [CLS] and [SEP] to the front of every

input and between each sentence of input.

We fine-tune the model with a token classification head on the top (a linear layer on the top of the hidden-states output). So, as the model produce the last layer of the hidden layer, it would be classified to the classes (B,I,O). In the $BERT_{BASE}$ model the final hidden layer has size of 768 and 3 out features. Figure 1 shows the architecture of the model.

To deal with the unknown words, we used the

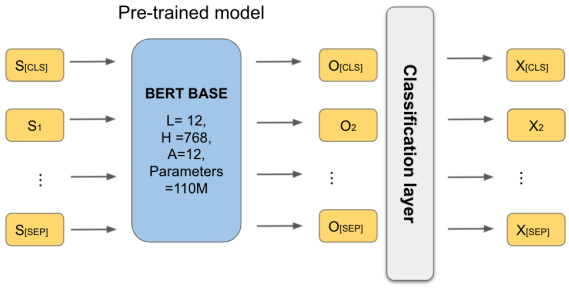

Figure 1: Architecture of Model

WordPiece, the subword tokenization algorithm used for BERT built in Transformer[3]. WordPiece initializes the vocabulary to include every character present in the training data and progressively learns a given number of merge rules. It choose the symbol pairs that maximizes the likelihood of the training data once added to the vocabulary. Since the split of word extend the number of tokens taken by the original token, we added a [MASK] to the token added and decode the output as the offset mapping to avoid the mismatch. For example, if the word HuggingFace was split as ['hugging', 'face']. We would obtain the output tags ['B', [MASK]]. Then we can get rid of the [MASK] for the final output. For the further experiment, RoBERTa has the same architecture as BERT, but uses a byte-level BPE as a tokenizer (same as GPT-2) and uses a different pretraining scheme.

5 Learning (8pt)

The named-entity recognizer was trained as a classification task with loss calculated with **CrossEntropyLoss** and ignored the loss from padding and mask. The objective of task is to minimize the loss. The formal definition of CrossEntropyLoss from PyTorch is [4]

$$loss = \frac{\sum_{i=1}^{N} loss(i, class[i])}{\sum_{i=1}^{N} weight[class[i]]}, \quad (1)$$

where N is the number of class, and weight is assigned to each class. The loss of each class is calculated by

$$loss(x, class) = -log(\frac{exp(x[class]}{\sum_j exp(x[j]}), \quad (2)$$

where x is the probability output across all possible outputs and class is a specific class of output. Then,total loss is calculated by We assign equal weights to all class of output.

6 Implementation Details (3pt)

For the BERT base model, it used 12-layer, 768-hidden, 12-heads, totally 110M parameters. For the BERT large model, it used 24-layer, 1024-hidden, 16-heads, 336M parameters. For RoBERTa, it used 12-layer, 768-hidden, 12-heads, 125M parameters. We are not adding any extra parameters. We used Adam optimizer with **AdamW** proposed by Loshchilov at al. [5], which improve the rugularization in common library of Adamby decoupling the weight decay from the gradient-base updata. The dropout rate is fixed to 0.1. We implemented with the fixed learning rate 3e-5, eps 1e-8, learning rate warm up over the first 100 steps and weight decay of 0.01 for all experiment. To train the model, we ran on GPU, Tesla 4. On average, each epoch can be finished in around 1 minutes, which is acceptable.

7 Experimental Setup (4pt)

The input tweets was preprocessed and plug in the encoder to get the last layer of hidden state. Our baseline model is with the vanilla BERT BASE. We experiment on use BERT LARGE and other pretrained model and adding drop out layer. For all the experiments, we fixed the batch size to be 32. The evaluation metric we used to compare the result is the F1 score as 3, which is a better evaluation metrics than simple accuracy, because it convey the balance between precision and recall.

$$F = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (3)$$

8 Results

Test Results (3pt) The final test results is given by BERT Large with learning rate 5e-5 and treat hashtags as a single token. All other parameters are fixed throughout the experiment as described in Section! 4We used full trainset with 3 epochs. The best result we got from leaderboard is 0.638.

Development Results (6pt) The baseline model is BERT BASE with learning rate 3e-5 and 3 epochs. Each model was trained with two different methods of dealing with hashtags as described in Section 3. We experiment on the BERT LARGE and RoBERTa with the same parameters. We got a significant increase on the F1 score with BERT LARGE. We then experiment on the learning rate of the model. We found that learning rate 5e-5 outperform model with learning rate 3e-5. More specific data are shown in Table 2.

Model	F1-score
BERT base cased, hashtags 1	0.594
BERT base cased, hashtags 2	0.652
BERT base cased, hashtags 1	0.684
BERT base cased, hastags 2	0.706
RoBERTa BASE, hashtags 2	0.688
BERT large with lr 5e-5, hashtags2	0.71

Table 2: Development ablation F1 score

Learning Curves (3pt) As Figure **??**, we found that the performance of model fluctuate. There is no correlation between data size and model performance. The reason may be that we used the pretrained model that large portion of the model was pretraine. As the downstream task, we only train with a small decay rate, so the performance of our model largely depends on the pretrained model, which is expected that with BERT LARGE, the performance is better.

Speed Analysis (3pt) The methods of dealing with hashtags have no influence on the speed, so we only record one for each. The speed of training mainly depends on the number of parameters in the model. As in the table 3. We used GPU Tesla T4 with batch size 32, same as training.

9 Analysis

Error Analysis (6pt) In development data, there is no hashtag (words started by #) was 'O',

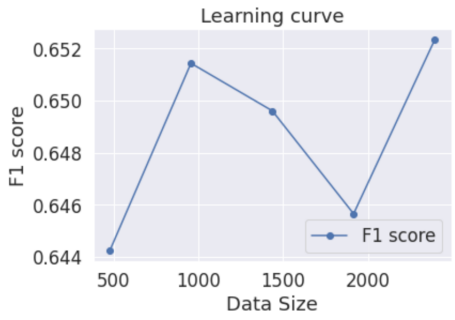

Figure 2: Learning Curve

Train Size	Speed
BERT base cased, hashtags 2	274
BERT large cased, hastags 2	89
RoBERTa BASE, hashtags 2	143
BERT large with lr 5e-5, hashtags2	88.7

Table 3: Speed for inference

while 27 of hashtags were identified as 'B' like ISIS, KASHABI and also some reasonable mistake #Brazil and #Australia. But we believe this is an annotation error that in the training set there are 9 hashtags were 'B' including '#Denver' and '#Dalas'. For the future development, we can add another dataset for common location name to identify the location name in the first place. There are total 140 of words was 'B' but was predicted as 'O'. Among them, there are 28(20%) of words are all capital and 100(70%) words are started with Capital character. In the whole development set, there are 1233 words are all Capital, 295 words are 'O' and 51 are 'B'

Confusion Matrix (3pt) Figure **??** shows the confusion matrix of our final model. We can tell that we have a higher accuracy on predicting 'B' than 'I', while there are $0.017 + 0.044$ misclassified between 'B' and 'I', which could be easily fixed by exchange the neighbor tags.

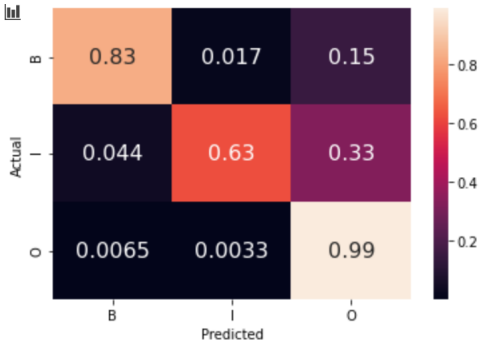

Figure 3: Deveopment Set Confusion Matrix

10 Conclusion (2pt)

As we used the pre-trained BERT model, our performance largely built on the performance of pretrained model. While the tweets contains too much noise and the annotated tweets data are not completely correct, the result can vary from task to task. But generally, the pretrained BERT can give a result around 65%.

References

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv.org*, May 2019.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv.org*, Dec 2017.

[3] "Summary of the tokenizers."

[4] "Crossentropyloss¶."

[5] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv.org*, Jan 2019.