EE 599 Deep Learning – Named Entity Recognition with pretrained BERT

Wenjing Lin, Jiaqi Liu

May 8th, 2020

Abstraction

These days, Natural language processing, in brief NLP, becomes more and more popular. In this project, we first summarize several techniques in NLP, including old and basic word embedding methods like word2vec[1] and GloVe[2], Cove[3], RNN, attention mechanism, bidirectional models and state-of-art transfer methods like BERT model. Then we implement a named entity recognition task based on finetune pretrained BERT model.

1. Literature Review on NLP Techniques

1.1 Word2vec

Word2vec is a group of models used to produce word embedding. There are two main techniques in Word2vec, one is Continuous Bag-of-Word (CBOW), the other is skip-gramps.

Word2vec use a context window move in the documents and try to represent word as vector by taking advantage of the target word and surrounding words. CBOW use several surrounding words in context window to predict target word. Skip-grams use target word to predict its surrounding words. The window size suggested by the author is 10 for both CBOW and skip-gram.

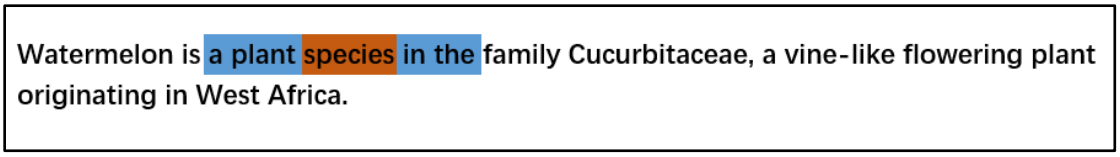


Figure1 Context Window, Size=5

Word2vec typically use two hidden layers, with softmax activation function between hidden layer and output layer. It finally represents word vectors as one hot vector. The dimensionality of word vectors is typically set to be between 100 to 1000. The softmax activation in high dimension is complex, so there are also some techniques to simplify computation like Hierarchical Softmax and Negative Sampling.

The ideal situation of word2vec model is that after training, words that share many common contexts in corpus will are closed to each other in vector space. The performance of word2vec model is related to the vector dimension, training data size, etc. However, word2vec model doesn't consider the order of words in the context windows, so it cannot distinguish like polysemy. Also, word2vec only provide the vector representation of words in the specific corpus. For different downstream tasks with new corpus, it needs to not only train word2vec model from scratch to get word embedding but also train for the task. The computation cost is high.

1.2 GloVe

Glove for Global vector is also a static word embedding model. It is based on the word-word co-occurrence matrix. The entries of co-occurrence matrix, say X_{ij} represents how many times the word j shows in the context of word i. P_{ik} represents the probability of word i occurs at the context of word k.

$$P_{ik} = \frac{X_{ik}}{\sum_{all\ j} X_{jk}}$$

GloVe model utilize the ratio of P_{ik}. The author found if P_{ik} / P_{jk} is close to 1, then we can say word i and j are like noise for predicting word k, because word k neither related to word i nor word j. If word k is related to i but not related to j, then the ratio would be much larger than 1. Otherwise, if k related to j but not i the ratio would be much smaller. The loss function in GloVe model is based on the co-occurrence matrix and the corresponding probability.

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

GloVe model is similar to Word2vec. They are all unsupervised learning, and all produce the word vector embedding. The difference of GloVe and word2vec model is that GloVe model use the global statistic information. The number of co-occurrence is based on the

whole corpus. However, the word2vec model use context window which only contains the local information in a window size.

GloVe model can produce lower dimension vectors compared to word2vec, so it achieves better result in a shorter time. But still, the GloVe model is a static word representation method, for different task and new corpus, every time it needs to train from script.

1.3 RNN

A recurrent neural network (RNN) is a kind of artificial neural networks, which connects nodes form a directed graph along a temporal sequence. Recurrent Neural Networks are for time series data and are commonly used in natural language processing. However, traditional RNN has several limitations, which makes it harder to get better performance.

Firstly, RNN are not very good at processing long sentences. For example, in RNN encoder-decoder model[4], encoder part is used to step through the input time steps and to convert the whole sequence into a fixed length vector. The vector is called context vector. And the decoder part is used to step through the output time step when decode from the context vector.

This mechanism leads to a problem: when sentence is too long, the corresponding context vector will also be long. This architecture limits the reasonable length of input sequences and results in worse performance for very long input sequences. Take the following picture as example, the longer input sentence is, the harder processing the sentence will be.

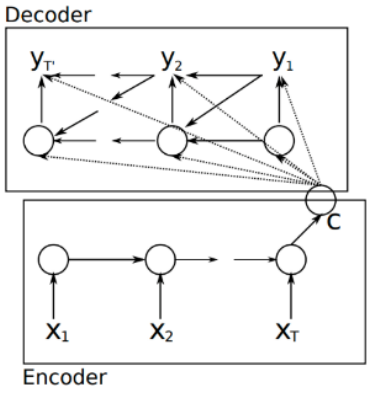


Figure2 RNN Encoder and Decoder

Secondly, traditional RNN architecture only catches dependencies in one direction when processing natural language, which means the meaning of word is only affected by the

word appears before. However, in real world we learn the meaning of a word from the context in both directions.

1.4 Attention Mechanism

As mentioned above, the traditional seq2seq model using RNN has drawbacks that forget the early parts of sequence. The attention mechanism is designed to solve this problem.

There are four steps using attention mechanism. For example, we compute the output at position i. The first step is input the word vectors into LSTM model computing all hidden states. Secondly, calculate the attention weights of hidden states and context vector si (αi in figure). Use Softmax activation for the weights to make sue the weights are between 0 and one and sum all weights get one. If it is the first word in encoder, use the last hidden state as the context vector. Thirdly, calculate the context vector based on the weights, hidden states and previous context vectors. Fourthly, compute the decoder output using the context vectors.

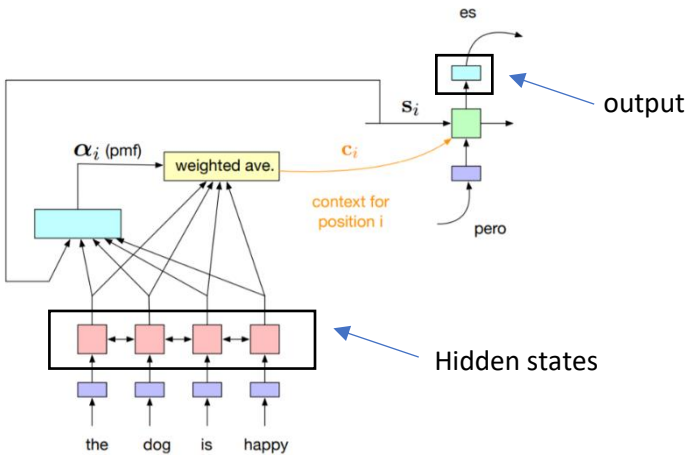


Figure3 Attention Model

Attention mechanism overcomes the drawback that in RNN encoder only produces one vector. In attention mechanism, encoder pay different attention weights on different input and produce a sequence of context vectors.

1.5 Cove

Inspired by successful transfer of CNN in ImageNet to other tasks in computer vision. Cove focus on training word encoder on a large NLP task and transfer that encoder to other NLP transfer. MT as known as machine translation requires a model to encode the

input words in context and decode them to another language. Cove chooses MT dataset for transfer learning, because MT task requires to reproduce the sentence to target sentence without losing information in source language, and there is an abundance of machine translation data.

How dose machine translation model work? Firstly, input the Glove vector in source language to a two layers bidirectional LSTM, calculate the hidden state. Then the decoder computes a vector of attention weight based on the current hidden state and previous target embedding. The decoder produces context-adjusted hidden states using attention weights and bidirectional LSTM. The final output is calculated from context-adjusted hidden states.

Cove so called as context vector is the output of MT-LSTM. Cove model concatenates the GloVe representation of words with the Cove vector.

$$Cove(w) = MT - LSTM\left(GloVe(w)\right)$$

$$W' = \left[GloVe(w),\ Cove(w)\right]$$

, in which w is the word vectors, W' is the new word vector.

Cove model use pretrained MT model to produce w' and use w' as input for specific down-streaming task. The conclusion in Cove paper (Learned in Translation: Contextualized Word Vectors) shows that the performance of using contextualized word vectors is better than random initialized word vectors, GloVe word embedding and character n-gram embedding.

1.6 ELMo

ELMo as known as embedding from language model is introduced by Allen Institute for Artificial Intelligence in 2018. It learns from a large text corpus to get the representation of words.

Specifically, ELMo can learn from the internal states of bidirectional LSTM. Bidirectional LSTM means forward LSTM and backward LSTM. Given a sequence of N tokens, $(t1,\ t2,\ \ldots tn)$, forward LSTM maximize the probability of $P\left(t_k \mid t_{k-1},\ t_{k-2},\ \ldots\ t_0\right)$. This means use words before the k^{th} word to predict the k^{th} word and maximize the probably of correct prediction. Backward LSTM maximize the probability of $P\left(tk \mid tk+1,\ tk+2,\ \ldots\ tN\right)$. This means use words after the k^{th} word to

predict the k[th] word and maximize the probably of correct prediction. Bidirectional LSTM combines forward and backward LSTM by maximizing the loglikelihood of forward and backward direction.

$$\sum_{k=1}^{N}(\log p(t_k \mid t_1,\cdots t_{k-1};\theta_x,\vec{\theta}_{LSTM},\theta_s)+\log p(t_k \mid t_{k+1},\cdots t_N;\theta_x,\overleftarrow{\theta}_{LSTM},\theta_s))$$

ELMo learns the weights of intermediate layers in biLSTM. Say there are L layers. Each layer has two directions. In the expression, xkLM is the k[th] token, $h_{k,j}^{LM}$ are token layers at two directions. So, a for each token x_k ELMo compute a set of 2L+1 parameter, in which 2L is L bidirectional LSTM layer, 1 is the original token layer.

$$R_k=\left\{x_k^{LM},\vec{h}_{k,j}^{LM},\overleftarrow{h}_{k,j}^{LM}\mid j=1,\cdots .L\right\}=\left\{h_{k,j}^{LM}\mid j=0,\cdots L\right\}$$

Word embedding is always used as the input of bidirectional RNN. However, ElMo also perform well at the output of RNN. Note that consider each bidirectional layer has different distribution, in some cases it needs to apply layer normalization.

1.7 BERT

The full name of BERT is Bidirectional Encoder Representations from Transformers. BERT is a language representation model with impressive accuracy in many tasks. Bidirectional Encoder Representations from Transformers is a technique for Natural Language Processing pre-training. BERT was developed, created and published by Google in 2018.

The following picture is overall pre-training and fine-tuning procedures for BERT.

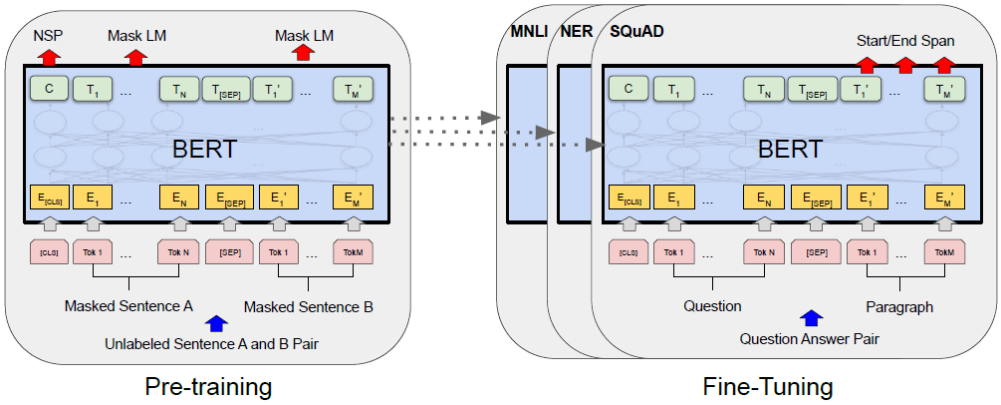


Figure4 Overall Pre-training and Fine-tuning Procedures for BERT

BERT framework can be divided into 2 parts, which are pre-training part and fine-tuning part. In pre-training part, train the model with unlabeled data in various pre-trained tasks. In finetuning part, initialize the BERT model with the pre-trained parameters, and fine-tune all parameters with labeled data from tasks. BERT alleviates the unidirectionality constraint by using a "masked language model" (MLM) pre-training objective[5].

BERT's model architecture is a multi-layer bidirectional Transformer encoder based on the original implementation described in Attention is all you need[6]. The attention mechanism, which could help model process long sentences, solves the drawback in traditional RNN. Attention is the idea to free the encoder-decoder architecture from the fixed-length internal representation. To achieve this, attention mechanism keeps the intermediate outputs from the encoder from each step of the input sequence and then train the model. Different from traditional RNN, attention mechanism tries to learn to pay selective attention to these inputs and connect selected part in input sequences to items in the output sequence. An attention process can be described as mapping a query and a set of key-value pairs to an output, in which the query, keys, values, and output are all vectors. The output is a weighted sum of the values, in which the weight assigned to each value is computed by a compatibility function of the query with the corresponding key[w3].

Attention mechanism is shown in the following picture.

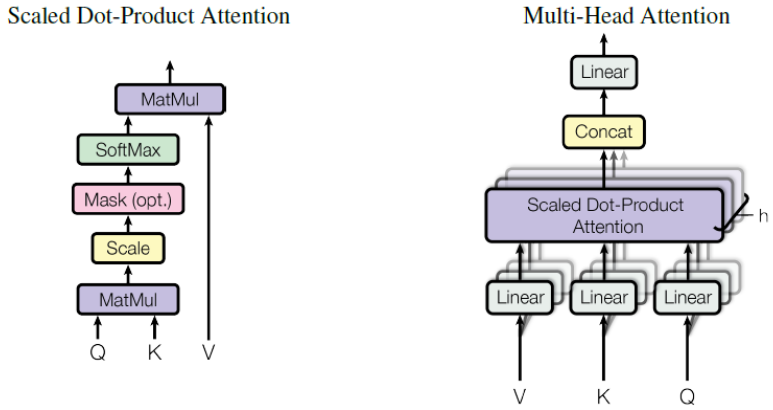


Figure5 (left) Scaled Dot-Product Attention

(right) Multi-Head Attention Consists of Several Attention Layers Running in Parallel

The Transformer follows this overall architecture by using stacked self-attention and pointwise, fully connected layers for both the encoder and decoder[w3].

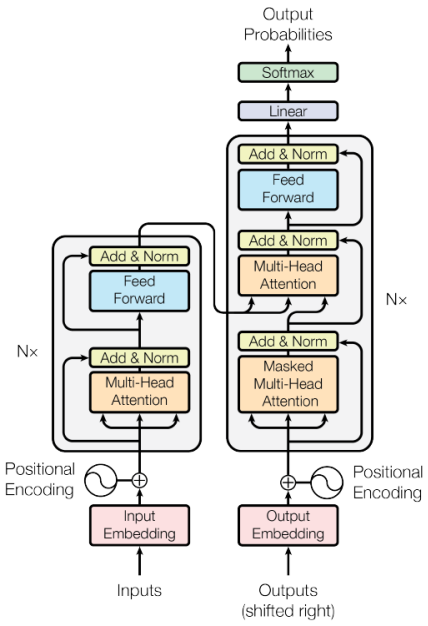


Figure6 The Transformer - Model Architecture.

Note that when use transformer, we need to add position information to make use of the order of sequences. For this goal, we add positional encoding into the input sequences embedding and store them at the bottom of encoder and decoder stacks. Dimension of positional encoding and embedding are the same, which makes positional encoding and embedding addable. Thus, by using Transformer, we get rid of sequential nature of RNN but still capitalize on the time ordering. From the above we could know that BERT solve all limitation of RNN.

1.8 Summary on NLP Techniques

In NLP area, there are two important improvement directions in recent years. The first one is how to learn the meaning of word. The initial methods learning word from the statistic (Word2Vec, GloVe), then some methods learn from unidirectional LSTM or shallow bidirectional LSTM, now there are state-of-art methods leaning from deep bidirectional context using MLM (masked language model).

Another improvement is transformers. In image processing, it is common that feed the new images into model that pretrained by some other images. Without retraining the model with new images, just slightly finetune the model or even don’t change any weights, the output of the model will capture the features of new images. The same language shares a similar corpus among different tasks. So, inspired by train image on

pretrained model, in NLP area, we also want to transfer a model to capture the features of words. Originally, we use GloVe to initialize word vectors which still need a lot computation for new corpus. Later there are methods like Cove and ELMo which utilize the bidirectional LSTM and attention mechanism to better understand the context and get better word representation. Finally, we have the BERT model. Pretrained BERT model works better than other transformers.

2. Implementation

2.1 Software

We use Pandas for data process. We use Tensorflow Keras layers, code and model provided by Google research group for training data. We also use BERT classification model provided by Pytorch and transformers. We run the model at Google Colab and AWS EC2 P3 Instances.

2.2 Dataset description

The whole dataset contains 1048575 words and 57959 sentences. Every word has one corresponding tag. The number of tags is very unbalanced. More than 84% of words with tag ‘O’, which means the word has non special meaning. Among tags, the quantity is also unbalanced. Tags like ‘nat’ and ‘eve’ and ‘art’ are less than 5% of total number.

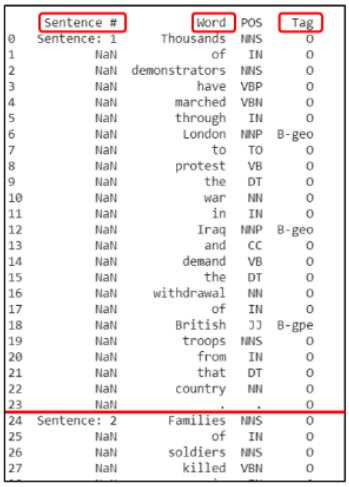
Tag	Amount
O	887908
B-geo	37644
B-tim	20333
B-org	20143
I-per	17251
B-per	16990
I-org	16784
B-gpe	15870
I-geo	7414
I-tim	6528
B-art	402
B-eve	308
I-art	297
I-eve	253
B-nat	201
I-gpe	198
I-nat	51

geo = Geographical Entity
org = Organization
per = Person
gpe = Geopolitical Entity
tim = Time indicator
art = Artifact
eve = Event
nat = Natural Phenomenon
B- = Begin of clump
I- = In the clump

2.3 Data Preprocessing

The dataset contains words, the tag of word and the word position. Below is part of our dataset. For example, ‘Thousands’ is the first word of the first sentence, ‘of’ is the second word of first sentence, ‘Families’ is the first word of the second sentence. ‘Tag’ column represents the tag of corresponding word. Because our input should be a sentence, output should be a sequence of tags, we need to use words in the dataset compose sentences and sequences of tags for training.

Because BERT model needs beginning marker ‘CLS’ and ending marker ‘SEP’ to check where is the next sentence, we also need to add markers into sentence. Below is our first input sentence and corresponding tags. Do preprocessing for all the sentences in the dataset.



	Sentence #	Word	POS	Tag
0	Sentence: 1	Thousands	NNS	O
1	NaN	of	IN	O
2	NaN	demonstrators	NNS	O
3	NaN	have	VBP	O
4	NaN	marched	VBN	O
5	NaN	through	IN	O
6	NaN	London	NNP	B-geo
7	NaN	to	TO	O
8	NaN	protest	VB	O
9	NaN	the	DT	O
10	NaN	war	NN	O
11	NaN	in	IN	O
12	NaN	Iraq	NNP	B-geo
13	NaN	and	CC	O
14	NaN	demand	VB	O
15	NaN	the	DT	O
16	NaN	withdrawal	NN	O
17	NaN	of	IN	O
18	NaN	British	JJ	B-gpe
19	NaN	troops	NNS	O
20	NaN	from	IN	O
21	NaN	that	DT	O
22	NaN	country	NN	O
23	NaN	.	.	O
24	Sentence: 2	Families	NNS	O
25	NaN	of	IN	O
26	NaN	soldiers	NNS	O
27	NaN	killed	VBN	O

```
['[CLS]', 'Thousands', 'of', 'demons',
 '##tra', '##tors', 'have', 'marched',
 'through', 'London', 'to', 'protest',
 'the', 'war', 'in', 'Iraq', 'and',
'demand', 'the', 'withdrawal', 'of',
'British', 'troops', 'from', 'that',
'country', '.', '[SEP]']

['[CLS]', 'O', 'O', 'O', 'X', 'X', 'O',
 'O', 'O', 'B-geo', 'O', 'O', 'O', 'O',
'O', 'B-geo', 'O', 'O', 'O', 'O', 'O',
'B-gpe', 'O', 'O', 'O', 'O', 'O',
'[SEP]']
```

Figure7 Example of Result of Sentences Processing

2.3 Convert Sentences into An Embedding

Below is processing of converting a sentence into an embedding and the corresponding codes.

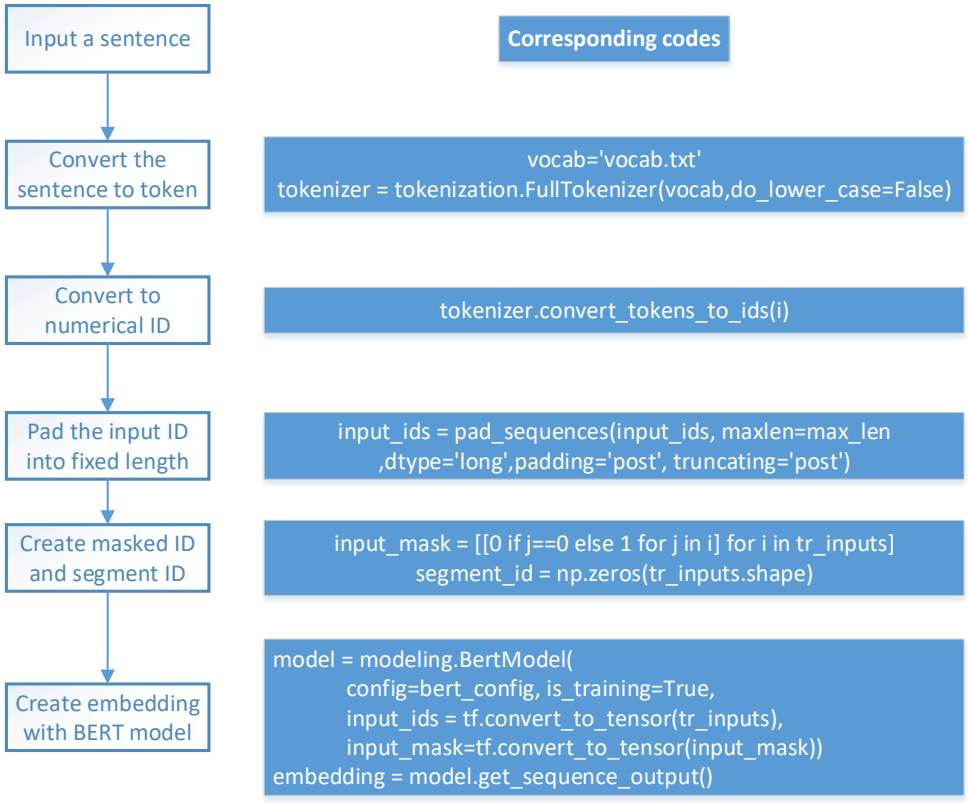


Figure8 Processing of Converting and the Corresponding Codes

By using Tokenizer package and vocab.txt, each word in a sentence will have corresponding token. Take word ‘demonstrators’ as example, this word will be converted to ‘['demons', '##tra', '##tors']’ according to vocab.txt file. Then we convert those tokens into corresponding numerical ID and pad them into a fixed length. After that we create masked ID and segment ID, in which masked ID is used to mask the padding and segment ID is used to avoid errors in BERT model. Finally, feed input ID and masked ID into BERT model, and we can get embedding with model.get_sequence_output() command. Details of BERT model and parameters we used will be described in the section 2.5.

2.4 Process Target Tags

The target tags are characters, deep learning model can’t deal with characters, we need to convert tags to numbers.

We made a dictionary that map every tag to number. Similarly, when we feed sentence into model and get the prediction of tags, we also need to covert the prediction number back to characters. So, another number to tag dictionary is needed.

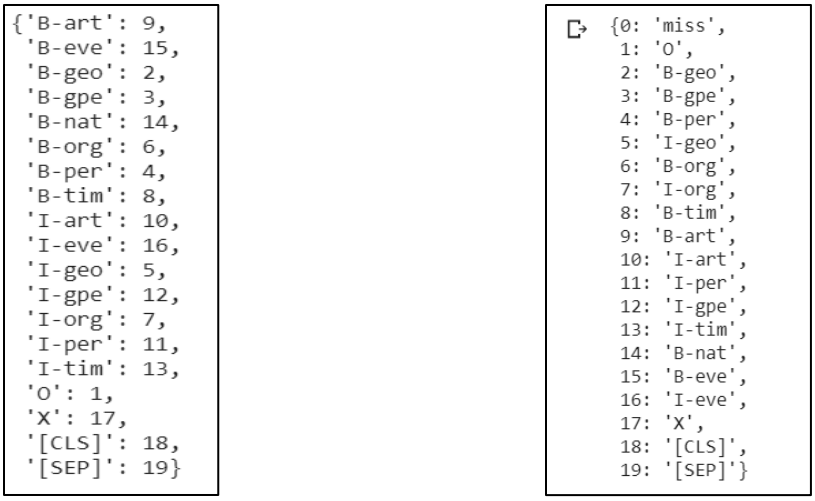


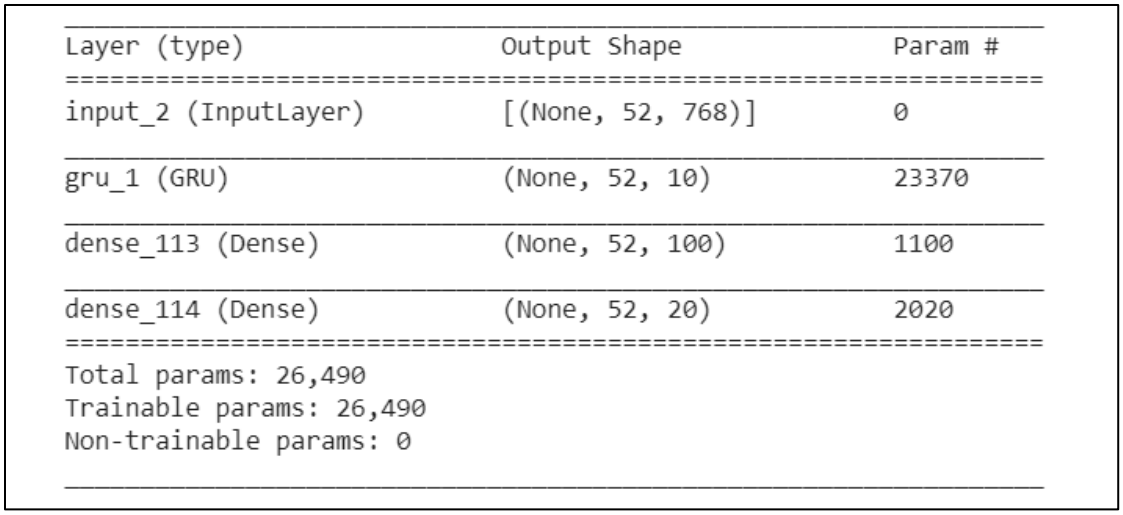
Figure9 (left) Tag to Number; (right) Number to Tag

2.5 Create Model

We first choose BERT model. We use cased large model. Cased means the model can distinguish the upper case and lower case of characters. The model has 12-layer, 768-hidden, 12-heads, 110M parameters. You can arbitrary choose to fine-tune parameters in which layer's. We set all parameters of all layers trainable. Input the sequence processed above and get the sequential output. Then output is the word embedding.

Then we feed the sequential output into the Keras layers to classify the tag of words.

We created three Keras layers shown as below. The input layer is the word embedding getting from the BERT model, and the size of word vector is 768. The second layer is the a GRU layer. The third layer is a full concatenate dense layer with 100 hidden nodes, using Relu activation function. The last layer dense layer is the output layer and will output 20 categories.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 52, 768)]         0
_________________________________________________________________
gru_1 (GRU)                  (None, 52, 10)            23370
_________________________________________________________________
dense_113 (Dense)            (None, 52, 100)           1100
_________________________________________________________________
dense_114 (Dense)            (None, 52, 20)            2020
=================================================================
Total params: 26,490
Trainable params: 26,490
Non-trainable params: 0
_________________________________________________________________
```

Figure10 Architecture of Our Model

The target output is the 20 tag numbers. So, we use Sparse Categorical Cross Entropy and Softmax activation to solve the classification problem.

2.6 training

We use 50000 words (1250 sentences) for training, 1000 words for validation. We run 20 epochs. It takes about 6 min for each epoch.

2.7 Another model and Training procedure

As described above, in the project, we first use the BERT model and tokenization function which uses Tensorflow 1.2. But some functions in tf1.x are deprecated and are incompatible with the current Tensorflow data loader and Keras layers. When we use this model, we can't handle the out of memory problem. The BERT model automatically feed all sequence into the memory instead of taking a batch size of data. So, we can't train all of our data. To solve this problem, we use another version of BERT model provided by Pytorch and transformers. The training data process and target tags process are all the same as using the previous google research model.

The only difference is we use BERT classification model provided by transformers instead of Keras layers. We write the customized training loop and evaluation loop. Unlike the history attribute of Keras model, we only calculate tags that have meanings. This means to omit tags like 'O', 'X', '[CLS]', '[SEP]', only consider the correct or

wrong about the meaningful tags like ‘per’, ‘org’, etc. And we also combine the ‘per-B’, ‘per-I’ tags together when calculate the model performance.

We used totally 1048575 words and 47959 sentences, 80% for training and 20% for validation. We run 5 epochs with batch size equal to 10. The BERT classification model uses sparse categorical cross entropy loss function.

3. Results and Analysis

3.1 BERT model & Keras layer

The training accuracy is 87%. The validation accuracy is 50%.

We think there are two main reasons why accuracy is low. The first one is because the memory problem, we didn’t train enough data. Secondly, the loss the calculated by Keras layer includes the padding value. The padding value is used to make sentences have the same length in order to feed into BERT model. Include the padding value into loss can have bad influence to learn the tags.

3.2 Pytorch & BERT classification model from transformers

We got training loss of 0.02. Validation accuracy of 97.03%, F1 score is 85.04%. We also calculate the confusion matrix showing below.

	precision	recall	f1-score	support
per	0.85	0.83	0.84	1863
org	0.75	0.74	0.75	2169
geo	0.84	0.91	0.87	3154
tim	0.89	0.87	0.88	1634
eve	0.31	0.35	0.33	31
gpe	0.96	0.93	0.95	1844
art	0.16	0.36	0.22	28
nat	0.50	0.31	0.38	13
micro avg	0.84	0.86	0.85	10736
macro avg	0.85	0.86	0.85	10736

We can see data of art and event tags are significantly less than other data. And the F1 score of art and event are the lowest.

3.2 Difficulties Encountered

Our first challenge is understanding the whole process of BERT model and BER task. We are very new to NLP. Concepts like tokenization, word vector and how to convert word tags into a sequence of tags that corresponds the sentences take us long time to understand.

Secondly, the tokenization is difficult. The function provided by BERT research group is not very clear. There are many different versions and hard to know which one should we use.

We also met difficulty when convert tag to numerical label and pad label sequence. The dataset provides one tag for one word. However, a word could be converted to several tokens, so, the additional target tags should also be added. Don't add additional tags will cause the word vectors after embedding don't match the target labels.

We explore a little bit about the loss function. Because this is a typical classification problem, we have totally 20 tags (20 categories) using one integer to represent the tag, it is normal consider about the Sparse Categorical Cross entropy and Softmax activation. But using it will cause nan loss value, probably because the too many categories. Then we tried use MSE regression and Relu activation, minimizing the gap between prediction and the target label. However, MSE regression is suitable to continue targets. Use it to discrete-target classification problem loss the meaning of classification, for example the probability of choosing each category not sums to one. Also the prediction of regression loss can produce prediction number more than 20 (more than index in number to id dictionary). So, it causes additional difficulty changing prediction back to tag. Finally, we solve it by dividing the input word embedding by 255. We found after dividing, use Sparse Categorical Cross Entropy and Softmax activation won't cause nan value.

The out of memory problem is also tough. We first use tokenization and pretrained BERT model which only support Tensorflow 1.2. We get the word embedding successfully. But later we need to run Keras layer for classifying tags. Tensorflow 1.2 works bad with Keras layer and causes memory problem. We didn't find elegant solution solve this problem. We tried to write customized training loop, but it didn't work. So we tried another version of tokenization function and BERT classification model provided by Pytorch and transformers.

4. Conclusion

In this project, we have made a Literature Review on NLP Techniques and apply BERT model on Annotated Corpus for Named Entity Recognition Dataset. Literature Review on

NLP Techniques includes introduction of Word2vec, GloVe, RNN, Attention mechanism, Cove, ELMo, and BERT. Especially in section 1.7, we introduce BERT is Bidirectional Encoder Representations in details from Transformers in 2 parts: Attention Mechanism part and Transformers part. By using Attention mechanism and Transformers, BERT solves drawbacks in traditional RNN. The At the end of Literature Review on NLP Techniques, we did a summary on NLP techniques. In implementation part, firstly we choose an Uncased BERT model to get embedding of input sequences. The model has 12-layer, 768-hidden, 12-heads, 110M parameters. Then we add GRU and dense layer form Keras and train the model in Google Colab and Amazon Web Service. Result… to be added.

5. References

[1] J. Devlin, M. Chang, K. Lee and K. Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, arXiv:1810.04805 [cs.CL], May 2019, Available: https://arxiv.org/abs/1810.04805

[2] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean, Distributed Representations of Words and Phrases and their Compositionality, October 2013, Available: http://search.proquest.com/docview/2085905727/

[3] M. Peters, M. Neumann, M. Iyyer, et al, Deep contextualized word representations, March 2018, Available: https://arxiv.org/abs/1802.05365

[4] Kyunghyun C., Bart M., Caglar G., Dzmitry B., et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, arXiv:1406.1078 [cs.CL]

[5] Taylor, W. L. (1953). Cloze procedure: A new tool for measuring readability. Journalism Quarterly, 30, 415-433.

[6] Ashish V., Noam S., Niki P., Jakob U., Llion J., et al. 2017. Attention is all you need. In Advances in Neural Information Processing Systems, pages 6000–6010.