AI Report

Wu Xinyun 1004536

Wang Han 1004520

Gao Xin 1004517

Xiong Maihe 1004537

## Task Description:

Our main task is to apply Named-entity recognition(NER) to a given sentence. NER is a subtask of information extraction which concerns the processing of human language texts by Natural Language Processing(NLP) methods. The aim of NER is to locate and classify named entities mentioned in various unstructured text into pre-defined categories such as names, organizations, quantities, monetary values, etc.

## Dataset Description:

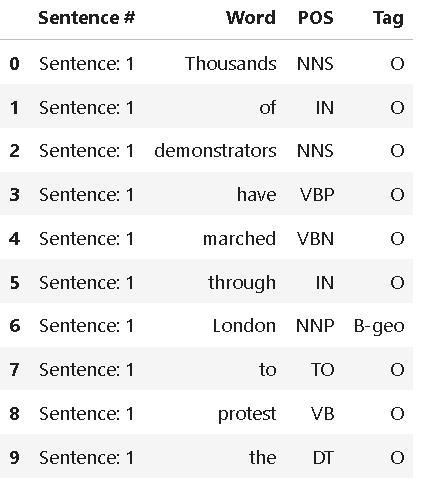
We chose the subset of data where we obtained our data from Kaggle(with the link <https://www.kaggle.com/datasets/abhinavwalia95/entity-annotated-corpus>).

Our entity tags are encoded using a BIO annotation scheme, where each entity label is prefixed with either B or I letters. B- denotes the beginning and I- inside of an entity. The prefixes are used to detect multiword entities, e.g. sentence: "World War II", tags:(B-eve, I-eve, I-eve). All other words, which don’t refer to entities of interest, are labelled with the O tag(Figure 1).



*Figure 1.* Some simple examples of tags and their meanings

Our data set has 47959 sentences and 35178 words with 17 tags('O', 'I-nat', 'I-eve', 'B-nat', 'I-art', 'B-gpe', 'I-org', 'I-gpe', 'B-per', 'I-per', 'B-eve', 'I-tim', 'B-geo', 'I-geo', 'B-org', 'B-art', 'B-tim'). The following graph is the first ten rows of our data (Figure 2).



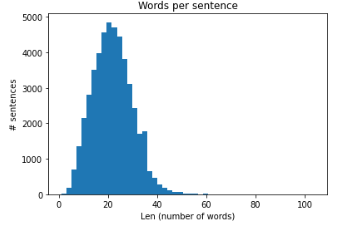
*Figure 2* The first 10 rows of our dataset

The POS means Part-of-speech tagging, which is the process of marking up a word in a text (corpus) based on its definition and its context(Table 1).

| Lexical Term | Tag | Example |
| --- | --- | --- |
| Noun | NN | Paris, France, Someone |
| Verb | VB | Work, train, learn |
| Determiner | DT | The, a |

*Table 1.* Grid displaying different types of lexical terms, and their tags

Besides, we also plotted the words per sentence in a histogram. (Figure 3)



*Fig 3* Words per sentence

## Preprocessing:

We used the “**word2idx**” dictionary to convert each word to a corresponding integer ID and the “**tag2idx**” to do the same for the labels. Representing words and tags as integers index can help us later in training deep learning models.

To feed the text into our Bi-LSTM-CRF model, all sentences should be the same length. We ensure this using the “**sequence.pad\_sequences()**” method and the “**MAX\_LEN”** variable. All texts longer than “MAX\_LEN” are truncated, and shorter texts are padded to get them to the same length. We also chose the post padding, which means we want to pad after each sequence. Then we did the train-test split with the “train\_test\_split” library while we set the test and validation split parameter to be both 0.2(80-20 ratio split). Finally, we stored our data into Dataloader with shuffle and appropriate batch size.

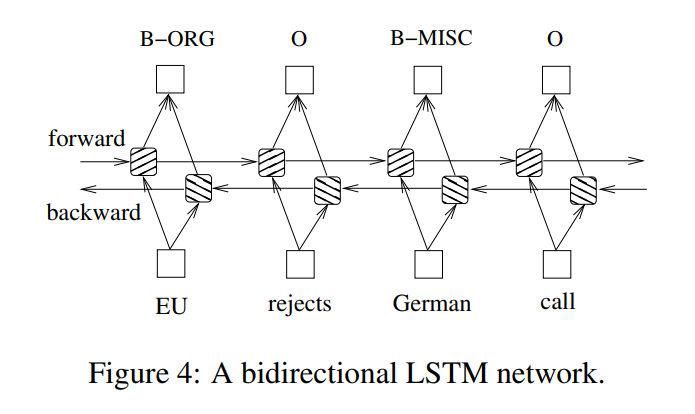
We first used the pre-trained BERT tokenizer for our BERT model with the method “BertTokenizer.from\_pretrainedd'bert-base-uncased', do\_lower\_case=True)”. We also used the “**sequence.pad\_sequences()**” method to transfer tokenized text and tags to id.

Meanwhile, we also adopt the 8:2 train-test split ratio and stored data in Dataloader to maintain consistency with the Bi-LSTM-CRF model, which boosts further comparisons and discussion.

## Description of Models and Loss

In this project, we used two models to classify named entities in text into pre-defined categories.

Firstly, we implemented a similar bidirectional LSTM model based on Zhiheng.H’s sequence tagging paper (Figure 4).



We first have an embedding layer with dropouts to get the word embeddings to put in the network. Name entities task requires accessing past and future input features. With a bidirectional LSTM network, we can efficiently use past features (via forwarding states) and future features (via backward states). To classify the output from Bi-LSTM, we add a fully connected layer after it.

We use cross-entropy loss as it works as a multi-label classification problem.

Secondly, we implemented a pre-trained BERT model from Hugging Face for the text name entities classification task. BERT is an acronym for Bidirectional Encoder Representations from Transformers. It is pre-trained on unlabeled data extracted from BooksCorpus and utilizes the bidirectional nature of the encoder stacks.

In our task, we choose the BERT base model, which is a BERT model consisting of 12 layers of Transformer encoder, 12 attention heads, 768 hidden sizes, and 110M parameters in total.

Same as Bi-LSTM, we also use cross-entropy loss for the BERT model.

## Description of Hyper-Parameter Settings

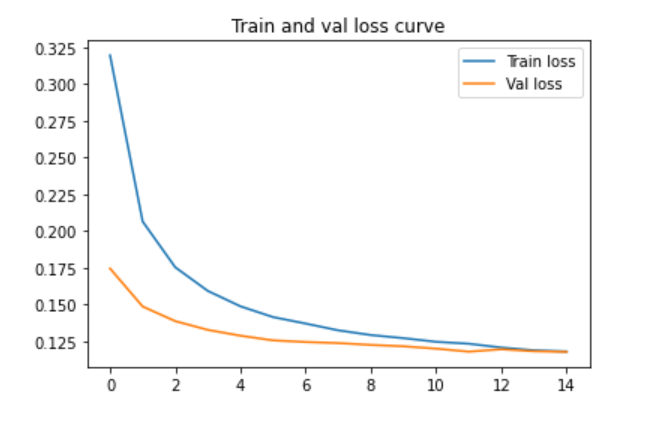
We set up a hyper-parameter fine-tuning for the masked language model BERT. We use Adam optimizer, which allows us to apply different hyperparameters for specific parameter groups. For this BERT, we apply weight decay to all parameters other than bias, gamma, and beta terms. Moreover, during training BERT, we implement gradient clipping with maximum gradient norm 1 to prevent exploding gradients in the deep networks

For the Bi-LSTM model, to prevent overfitting, we implement a dropout layer with a dropout rate of 0.5.

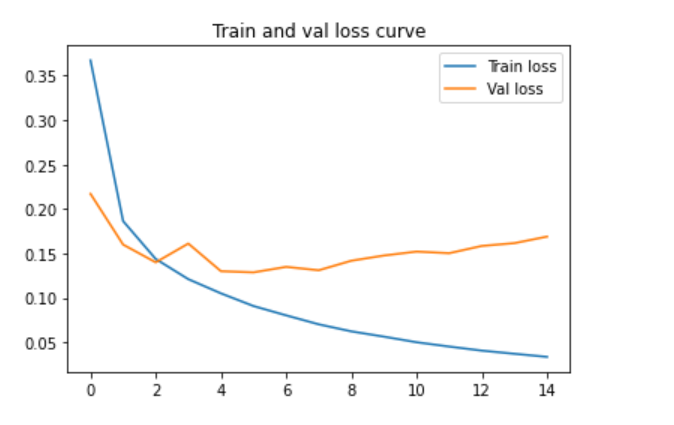
Both models are trained for 20 epochs in batch size 64. During training, both models traced and saved the best models according to the minimum validation loss.

## Train and Test Performance(Figure 5, Figure 6)

Below are two graphs that visualize the training and validation loss for both models. The analysis of both loss curves is in the next session.



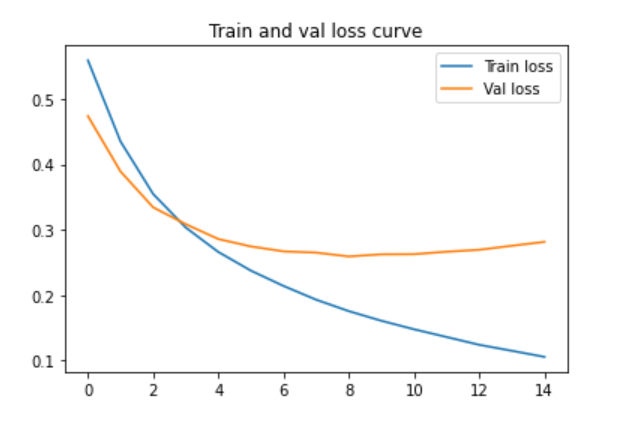
*Figure 5.* Train and validation loss curve for Bi-LSTM



*Figure 6.* Train and validation loss curve for Pre-trained BERT

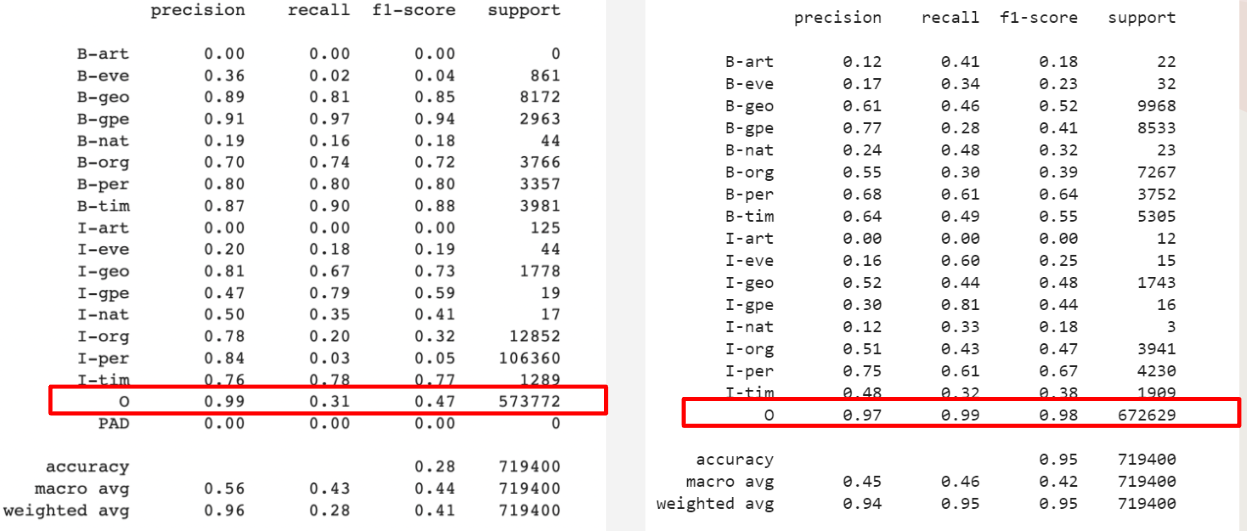
## Results Discussion and Improvements

The loss curve for Bi-LSTM shows quite a good result, with both loss curve drop and reaching very low loss values. Pre-trained BERT, however, produces poor results. The training loss drops, while the validation loss fluctuates after a few epochs and even increases at the end. Perhaps overfitting is to blame. Hence, we trained another BERT model which has only 3 hidden layers and without pre-trained. The loss curve below is a bit better but the validation still has a growing trend to the end of training. (Figure 7)



*Figure 7.* Train and validation loss curve for 3-hidden-layer BERT

Our models are also compared based on their respective label-specific results along with their test accuracy scores. The left result is from the Bi-LSTM model, we found the model has a tendency to predict the more frequent label, for example, the O label. In this way, even though it has high precision, it only gives a low recall score, which means the model tends to predict words with O labels despite their actual tags. Hence, we decide to use the F1 score instead accuracy as our model selection criteria. The right result is from the non-pre-trained 3-layers BERT model which gives higher F1 scores. (Figure 8)



*Figure 8.* Comparisons of accuracy metrics between Bi-LSTM and 3-layers BERT

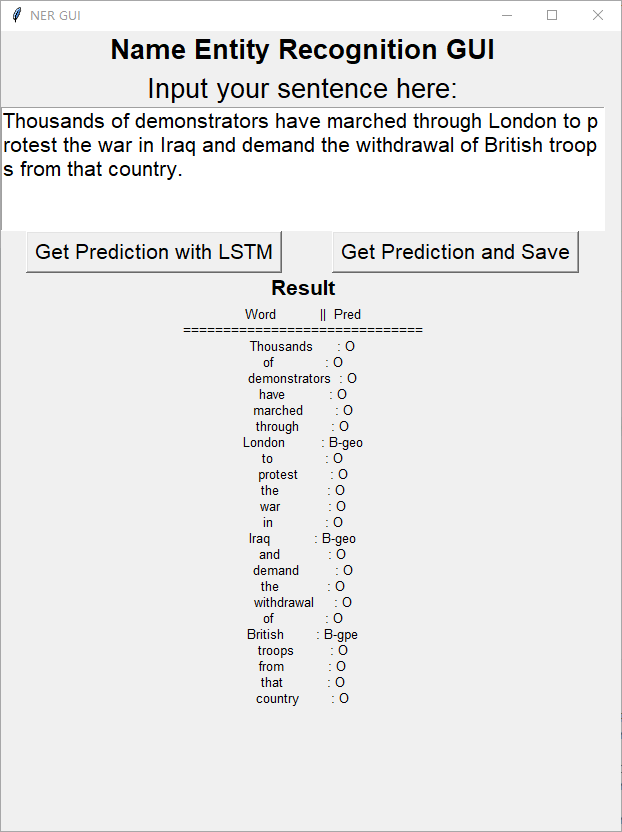
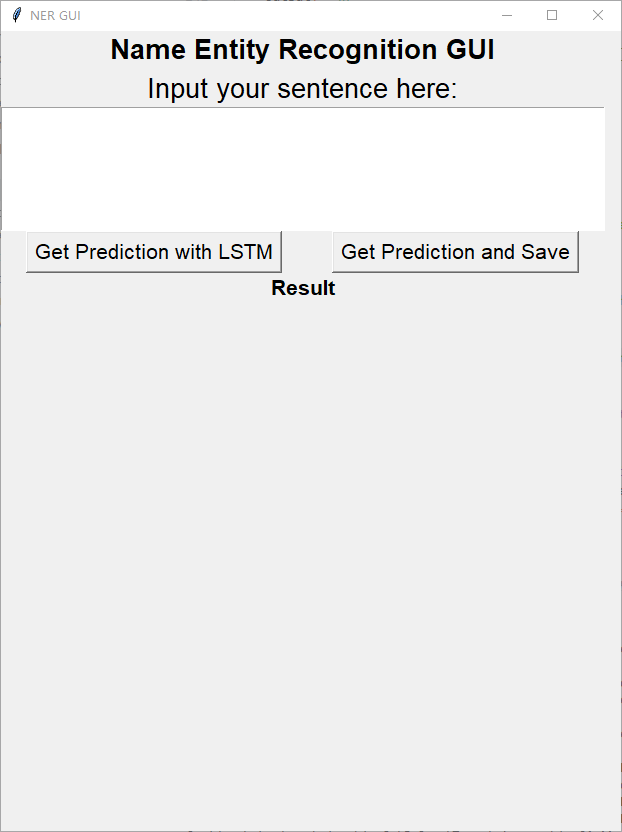
## GUI

**Design Choice**

Before the formal design of the GUI, the functions of the GUI were rigorously demonstrated within the group. We think that people who know how to get raw data and use our model for training will not stick to the GUI itself. Due to the format, distribution, etc. of the original data, it is most convenient to use the code directly. Therefore, we think that people who use GUI should only want to use the model we trained, in order to get the tag predicted by the model. Therefore, for GUI, its function is defined as inputting a sentence, and outputting the corresponding tag according to the input sentence

**Features of the GUI**

The GUI contains 3 parts. It has a text input box for entering the sentence that needs to be predicted. After the sentence input is completed, click the button component on the left to get the result of the model prediction. Or click the button on the right to get the prediction and save the result to a file, which has the name “output.txt”. The corresponding example is shown in the following figure(Figure 9).



*Figure 9.* GUI showcase

**Usage of the GUI**

Pre requirement:

numpy

Torch1.12.0+cuda113 (The version of current colab. It is the version we trained the model with.)

Open the GUI:

1.Switch to the folder contains GUI.py

2.Run the following command:

python GUI.py

Reference:

1. Huang, Z., Xu, W. & Yu, K. (2015). Bidirectional LSTM-CRF Models for Sequence Tagging (cite arxiv:1508.01991)