50.039 Deep Learning Project Report

Wang Yanbao 1004865

Lin Yutian 1004881

1. Introduction

In the evolving landscape of healthcare, the digitization of medical records enhances patient care by enabling the use of technology to extract critical information. Named Entity Recognition (NER), a critical task within NLP, identifies and categorizes crucial medical terms from texts, such as diseases and medications, streamlining the data analysis process.

In this project, our objective was to leverage a Bidirectional Long Short-Term Memory (BiLSTM) model to efficiently and accurately identify medical entities from the unstructured text of clinical reports.

2. Data Preparation

2.1 Dataset

Our project utilizes the n2c2 NLP Research Data Sets, comprising unstructured clinical notes from the Research Patient Data Registry at Partners Healthcare. The dataset consists of 303 training samples and 202 testing samples. Each sample includes a pair of files: one containing the original medical report in text format and a corresponding annotated file that identifies specific medical entities, marking their type and position in the text.

The entities are categorized into nine types: [DRUG, STRENGTH, FORM, FREQUENCY, ROUTE, DOSAGE, REASON, ADE (Adverse Drug Event), DURATION]

2.2 Tagging System

To enhance the model's capacity to recognize entities more accurately, we have refined the tagging system used in the original dataset by extending the 9 labels.

a. We introduced the 'O' tag to label tokens that do not constitute a medical entity, thus providing a clear distinction for non-entity text.
b. We transformed the original tags into a format that distinguishes the beginning of an entity from its continuation. This is crucial for identifying multi-token entities such as "once a day" as a cohesive unit. Thus, we utilize the 'B-' prefix to denote the beginning of an entity, 'I-' for tokens inside an entity, and 'O' for non-entity tokens.

This results in a modified tag set that expands the original 9 types into 19 distinct tags: [B-DRUG, I-DRUG, B-STRENGTH, I-STRENGTH, B-FORM, I-FORM, B-FREQUENCY, I-FREQUENCY, B-ROUTE, I-ROUTE, B-DOSAGE, I-DOSAGE, B-REASON, I-REASON, B-ADE, I-ADE, B-DURATION, I-DURATION, O]

2.3 Data Processing Steps

2.3.1 Step 1: Tokenization

First of all, we utilized the tokenizer from the fine-tuned BERT model, "praneethvasarla/med-bert" to tokenize the clinical report texts. The rationale for selecting a tokenizer from a model fine-tuned on medical corpora include:

a. The fine-tuned model is expected to be more effective at processing medical terminology.
b. The tokens generated should align with those the embedding model was trained on.

c. The embedding model is expected to better handle out-of-vocabulary medical terms, as the tokenizer can break rare words down into known subword units.

2.3.2 Step 2: One-Hot Encoding of Labels

After tokenization, we employ one-hot encoding to transform categorical labels into a binary format. This encoding method represents each label as a binary vector with a size corresponding to the label set (19 in our case), where the position of the actual label is denoted by '1' and all other positions by '0's.

2.3.3 Step 3: Label Mapping

The next step involves mapping each token to one of the 19 labels. The output format is a structured JSON object that contains the tokens, their respective token IDs, and assigned labels.

2.3.4 Step 4: Input Truncation

Since the embedding model "praneethvasarla/med-bert" we used is limited to processing sequences of up to 512 tokens, we truncate longer inputs into multiple segments, each with a maximum length of 512 tokens.

2.3.5 Step 5: Padding

Inputs with fewer than 512 tokens are padded to reach the required length, maintaining a consistent input size for the model.

2.4 Dataset Splitting

The original dataset division was not optimal for our training and evaluation needs, particularly lacking a validation set. To address this, we restructured the dataset split into a 9:1:2 ratio for training, validation, and testing. This new split allows us to train our model on a sufficiently large dataset and preventing overfitting during training with a validation set.

2.5 Final Input to the Model

The final input to our model consists of tokenized clinical notes with mapped labels, truncated and padded to a fixed length of 512 tokens.

3. Model

3.1 Architecture

Our investigation cantered on two primary models: Long Short-Term Memory (LSTM) and Bidirectional Long Short-Term Memory (BiLSTM). Both models share a foundational structure that begins with BERT embeddings as input. Our objective was to compare these models under similar model size constraints to determine the most effective architecture for our NER tasks.

3.1.1 LSTM

The architecture for the LSTM model is as follows:

a. Embedding Layer: Due to our limited computational resources, we decided to use a pretrained embedding model "praneethvasarla/med-bert". It is a fine-tuned BERT model on medical corpus. It is chosen due to two primary considerations:
   - Domain-Specific Understanding: We hypothesized that an embedding model familiar with medical terminology would significantly enhance our model's ability to understand and classify entities unique to the medical field. And "praneethvasarla/med-bert" has been fine-tuned on medical texts, enabling it to capture and represent the specific terminology and semantic relationship present in medical data.
   - Strength of BERT as a Base Model: BERT's architecture, which employs bidirectional transformers, allows it to contextualize words in a manner unmatched by unidirectional models. We are confident in BERT as a foundation for our embedding layer due to its proven effectiveness across a wide range of natural language processing tasks.

b. LSTM Layer: A single-layer LSTM with a hidden size of 256. This layer processes the sequence of embeddings output by the "praneethvasarla/med-bert" model, capturing the temporal dynamics of the sequence.

c. Output Layer: A linear layer that maps the LSTM outputs to 19 different classes. A dropout of 0.5 is applied before this layer to improve generalization.

3.1.2 BiLSTM

The BiLSTM model shares the same initial embedding structure as the LSTM model, with the primary difference being in the LSTM layer:

a. Embedding Layer: Utilizes a "praneethvasarla/med-bert" to transform input tokens into embeddings.

b. BiLSTM Layer: Uses a bidirectional LSTM with a hidden size of 128 for each direction, effectively capturing information from both the past and future context within the sequence. The total output for each time step is 256 dimensions, concatenating the outputs from both directions.

c. Output Layer: Similar to the LSTM model, it maps the concatenated BiLSTM outputs to 19 classes. The same dropout rate of 0.5 is used to prevent overfitting.

3.1.3 LSTM vs BiLSTM: Capturing More Features or Exploiting Bidirectional Context?

We intended to make a comparison between a 256-dimensional LSTM and a 128-dimensional BiLSTM, specifically designed around our limited computational resources. This choice allowed us to keep both models comparable in size, enabling a fair and focused analysis on their performance under similar constraints.

On one hand, a larger hidden dimension in the LSTM model enables it to capture a wider range of input features, aiding in understanding complex medical texts. On the other hand, BiLSTM's bidirectional capability allows it to gather contextual information from both past and future states within a sequence. Therefore, the comparison is rooted in a critical question: Is it more beneficial to capture a wider range of features with a larger LSTM, or to understand context more deeply with a bidirectional BiLSTM?

In the experiment, we trained both of the two models at a learning rate of 0.001 for 20 epochs. These results suggest that despite the LSTM's capacity to capture a wider array of features, the BiLSTM's ability to process information in both directions provides a critical edge in performance within the confines of the same computational budget. Thus for our following exploration, we will focus on the BiLSTM model.

	Accuracy	Recall Score	Precision Score	F1 Score
LSTM (hidden_dim=256)	0.8895	0.7553	0.8853	0.8031
BiLSTM (hidden_dim=128)	0.9054	0.7717	0.8916	0.8162

Table 1 - Comparative Performance Metrics: The table displays the results achieved by an LSTM with a hidden dimension of 256 and a BiLSTM with a hidden dimension of 128.

4. Training Procedure

4.1 Adaptive Class-Weighted CrossEntropy Loss

We employed the CrossEntropy loss function, a common choice for classification problems. However, given the inherent class imbalance in NER datasets, a significant portion of tokens are labelled as 'O' (Outside) indicating non-entity words. We recognized a potential challenge. Specifically, there was a risk that the model might learn to favour the 'O' label disproportionately, as predicting the majority class could superficially minimize the loss, despite poor identification of actual named entities.

To counteract this, we adjusted our loss function to reduce the contribution of the 'O' labels to the overall loss. The modified loss function, which we can call Adaptive Class-Weighted CrossEntropy Loss, is then:

$$\text{AdaptativeClassWeightedLoss}(y, p) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} w_c \, y_{i,c} \log\left(p_{i,c}\right)$$

We introduced a weight factor w_c for each class c. In our case, for the 'O' class, this weight is set to 0.4 (to scale down its contribution), and for all other classes, this weight can be set to 1 (or another value if further balancing is needed). This adjustment aims to mitigate the imbalance's impact by decreasing the relative penalty for misclassifying entity tokens compared to non-entity tokens, thereby encouraging the model to focus more on the accurate identification of named entities.

4.2 Hyperparameter Search (Grid Search – Learning Rate + Hidden Dimension)

4.2.1 Learning Rate Range Test

In our study to determine the optimal learning rate for our BiLSTM model, we employed the learning rate range test. This empirical method evaluates how the loss of the model changes as the learning rate is varied. By plotting the loss against the learning rate on a logarithmic scale, we observed that the loss decreases as the learning rate increases from 10^-7 and reaches its nadir at around 10^-3. After this point, the loss begins to rise gradually, escalating sharply as the learning rate approaches 1.0.

The inflection point of the curve provides critical insight into selecting a learning rate that is aggressive enough to facilitate rapid learning, yet conservative enough to avoid the instability associated with overly large learning rate steps. Considering the plot's indication that loss remains stable and low within the range between 0.001 and 0.01, we conclude that learning rates between 0.001 and 0.01 are acceptable for our model's training.

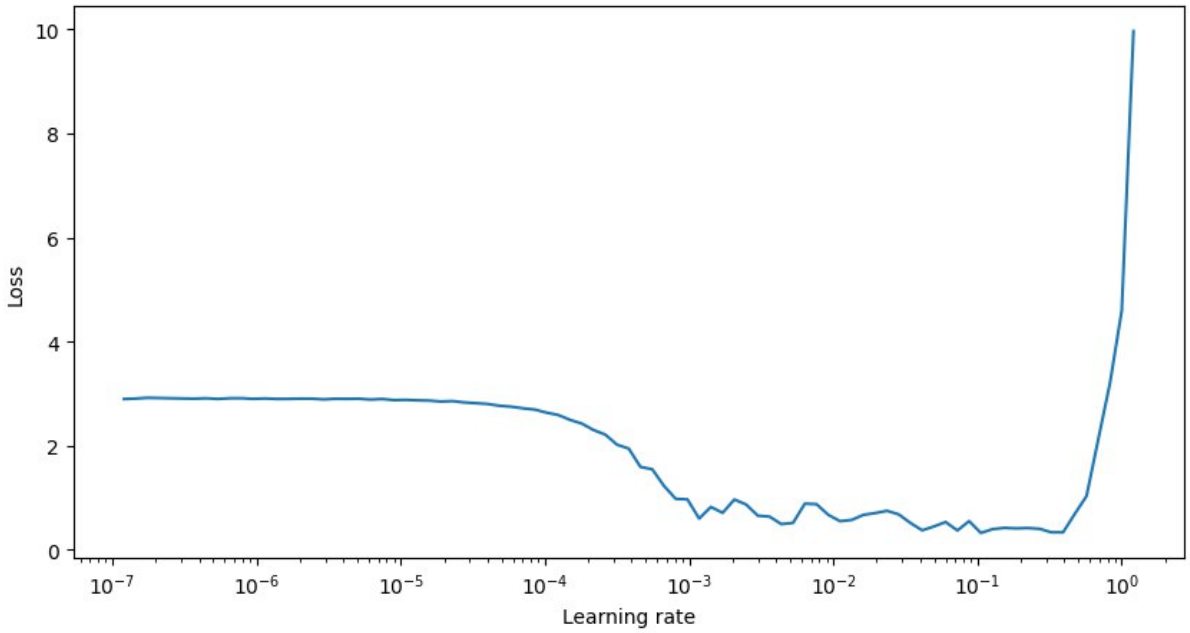

Figure 1 - Optimization of Learning Rate: The graph illustrates the loss response as the learning rate increases.

4.2.2 Grid Search

To optimize our BiLSTM model's performance, we employed a grid search approach to systematically explore the effects of varying hidden dimension sizes and learning rates. We evaluated combinations of three hidden layer dimensions (32, 64, 128) and three learning rates (0.001, 0.005, 0.01), training the model for 20 epochs for each configuration. After training, we assessed the models' performance by measuring accuracy, recall, precision, and F1 score on an independent testing dataset, ensuring that our findings reflected the models' ability to generalize to new data.

Learning Rate	Hidden Dimension	Accuracy	Recall Score	Precision Score	F1 Score
0.01	32	0.8435	0.6697	0.7954	0.7144
0.01	64	0.8793	0.7275	0.8799	0.7746
0.01	128	0.883	0.7419	0.8830	0.7901
0.005	32	0.8663	0.6920	0.7995	0.7345
0.005	64	0.8899	0.7474	0.8763	0.7952
0.005	128	0.8947	0.7635	0.8980	0.8141
0.001	32	0.8724	0.6977	0.8909	0.7367
0.001	64	0.8937	0.7543	0.8900	0.8020
0.001	128	0.9054	0.7717	0.8916	0.8162

Table 2 Performance Metrics for Hyperparameter Tuning: This table displays the accuracy, recall, precision, and F1 scores achieved by training the BiLSTM model with varying hidden layer dimensions and learning rates over 20 epochs.

The results from the grid search indicate that as the hidden dimension size increases, there is a general trend of improved performance across all metrics. Specifically, the largest hidden dimension of 128 consistently outperforms the smaller sizes at each learning rate level. Moreover, the lowest learning rate of 0.001, when paired with a hidden dimension of 128, achieves the highest scores in all four metrics: an accuracy of 0.9054, a recall of 0.7717, a precision of 0.8916, and an F1 score of 0.8162.

Based on the evidence gathered from our grid search, we conclude that a hidden dimension size of 128 paired with a learning rate of 0.001 strikes an optimal balance for our model. Therefore, we have chosen this configuration to proceed with further development and training of our BiLSTM model.

4.3 Optimizer

We have chosen the Adam optimization algorithm with a learning rate of 0.001 to refine our weights during training.

4.4 Early Stopping Based on F1 Score

Initially, we employed an early stopping mechanism that monitored the validation loss. Specifically, we stopped training if there was no improvement for 5 consecutive epochs. However, upon careful observation, we noticed that the validation loss continued to decrease while the F1 score on the validation set began to decline.

Given the class imbalance in our NER task, the majority of tokens are typically not part of named entities ('O' labels), and a model could achieve a low loss by simply predicting the majority class. This would result in high accuracy due to the class imbalance but poor precision and recall for the minority classes, which are often the most important for the task. Therefore, to more accurately identify and classify named entities, we shifted our early stopping criterion to focus on the F1 score. Now, our training process is designed to stop if there is no increase in the F1 score on the validation set over 5 iterations. This change prioritizes the balance between precision and recall, ensuring that our model remains effective at identifying the relatively rarer named entities

4.5 Model Checkpointing

Our training framework is designed with a checkpointing mechanism to capture the state of our model at the end of each epoch. These checkpoints, saved in '.pth' files under the “checkpoint” directory, include all model parameters and optimizer states. This enables us to track progress, perform evaluations at various training stages, and resume training if necessary without loss of information.

5. Result

5.1 Comparative Analysis of Model Convergence

In assessing the performance of our model, we carefully monitored the loss, accuracy, and F1 score throughout the training process. The decision to stop training was guided by the analysis of both the loss and F1 score over the training epochs.

The loss curves from our training and validation sets showed early rapid decreases and then stabilized. And the parallel progression of the training and validation loss curves suggests that our model was trying to learn generalizable patterns rather than overfitting.

Meanwhile, the F1 score presents a similar pattern of rapid improvement that stabilizes as training progresses. Its stabilization in later epochs indicates that both precision and recall have reached satisfactory levels.

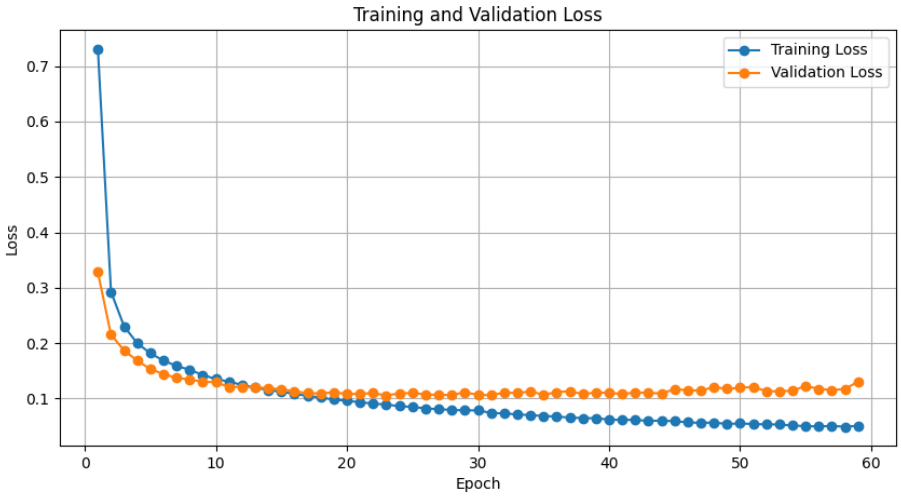

Figure 2: Training and Validation Loss Trends

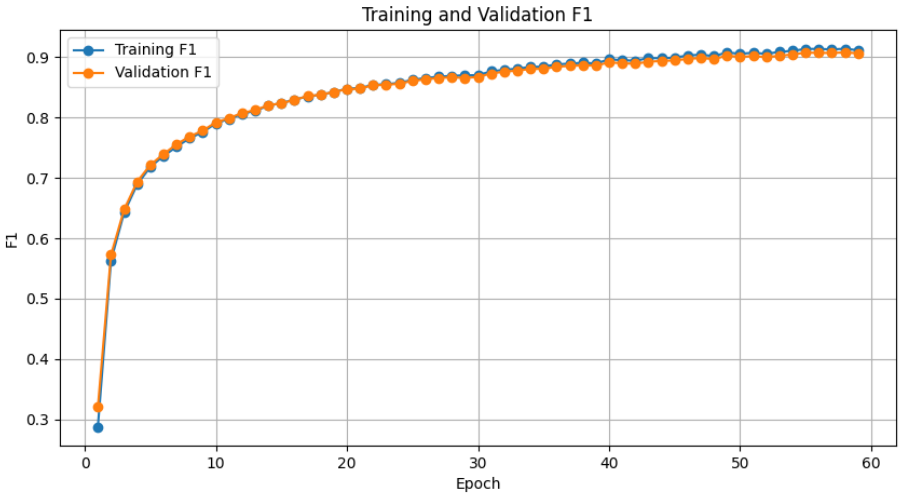

Figure 3: Training and Validation F1 Scores

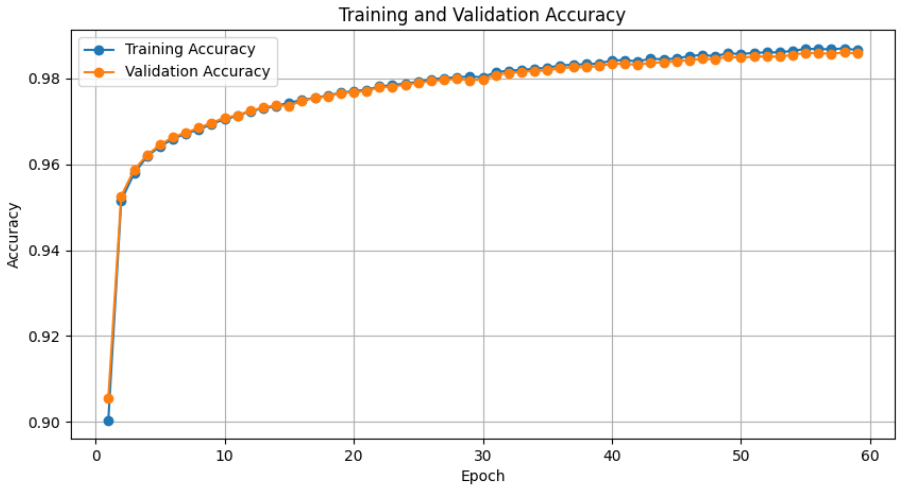

Figure 4: Training and Validation Accuracy Scores

5.2 Evaluation Matrices

Our BiLSTM model, with a hidden dimension of 128 and a learning rate of 0.001, has shown excellent performance across all datasets. Trained with 60 epochs, it achieved a low average loss and high accuracy, precision, and recall scores on the training dataset, which were closely mirrored on the validation set. Notably, on the testing dataset, the model maintained strong accuracy (90.09%) and an F1 score of 0.8206, demonstrating its effective generalization to unseen data.

	Accuracy	F1 Score	Recall Score	Precision Score
Training	0.9866	0.9116	0.9283	0.8960
Validation	0.9858	0.9066	0.9197	0.8940
Testing	0.9010	0.8206	0.7741	0.8963

Table 3: Performance Metrics of BiLSTM Model on Training, Validation and Testing Datasets

6. Limitations

Despite the promising results achieved by our model, we acknowledge several limitations that, if addressed, could potentially enhance its performance.

6.1 Embedding Model Adaptation

Our current system employs the "bert-med" embedding model, a domain-specific adaptation of BERT, to encode input tokens. While this model has been fine-tuned for medical terminology, it has shown limitations in encoding the specialized vocabulary present in our dataset. Based on our observations, it tends to split most of complex medical terms in our dataset into subwords. It indicates a mismatch between the vocabulary of the embedding model and the corpus in our dataset. We believed a further fine-tuned embedding model, specifically tailored to our dataset, could offer more cohesive and accurate representations of terms, leading to improved model performance.

6.2 BiLSTM Hidden Dimension Size

The architecture of our model incorporates a BiLSTM layer with a hidden dimension size of 128 for each direction. Comparative analysis with a unidirectional LSTM with a hidden dimension of 256 yielded comparable results, with the BiLSTM achieving slightly better performance. This suggests that increasing the hidden dimension size of the BiLSTM to 256 for each direction could allow the model to capture a richer set of features, thus improving its ability to model complex dependencies in the data. However, due to resource constraints, this hypothesis remains untested and represents a potential avenue for future work.

6.3 Conditional Random Field (CRF) Potential

we believe that incorporating a Conditional Random Field (CRF) layer could further enhance the model's capacity for sequence tagging. A CRF layer can explicitly model the dependencies between labels in a sequence, which is especially beneficial in NER tasks where the context and label sequence patterns play a crucial role. Although we did not experiment with a CRF layer in the current iteration of our model, we believe that its inclusion could yield performance gains.

7. Conclusion

In conclusion, our BiLSTM model with a hidden dimension of 128 and a learning rate of 0.001 has effectively performed Named Entity Recognition in clinical text, balancing precision and recall with high accuracy across training, validation, and testing datasets.

8. contributions

Lin Yutian: Model, Training and Report.

Wang Yanbao: Data Processing, Training and Performance Visualization

Appendix:

Setup Guide:

Our models are available in our Github repository:
https://github.com/wangyanbao666/Medical-NER.git

Ensure your directory structure is organized as follows before running the notebook:

project/

├── input/ # training and testing data

├── models/ # the final model(s)

├── checkpoints/ # storing checkpoints

└── NER.ipynb # the main Jupyter notebook

Here's a step-by-step guide to help you get started with our **NER.ipynb** notebook, which includes code to load data, build, train, and evaluate our models.

Step 1: Install Required Libraries

First, ensure that you have all the required libraries installed. You can do this by running the following command in your terminal:

`pip install -r requirements.txt`

Step 2: Configure PyTorch

Depending on your CUDA version, you may need to install a specific version of PyTorch. Please refer to the PyTorch previous versions page to find and install a compatible version for your setup.

Step 3: Download Checkpoints and Datasets

Due to the large size of the model checkpoints, we have provided only the final checkpoints for each model on Google Drive. Use the following links to access and download them:

- Model Checkpoints: Google Drive - Checkpoints
- Training and Testing Dataset: Google Drive - Datasets

Step 4: Setup Model and Checkpoints

If you intend to train the model yourself and use your own checkpoints, please rename your best checkpoint file to best_model.pth and place it in the **models/** directory, structured according to the model type and hyperparameters used.

After completing the above steps, you are ready to open **NER.ipynb** and run the scripts.