Analysis of BERT and LSTM Based Models on Multi-label Coding Questions Classification

Tianle Wang
Department of Computer Science
University of Toronto
tianle.wang@mail.utoronto.ca

Yifan Zhao
Department of Computer Science
University of Toronto
ethany.zhao@mail.utoronto.ca

Yuezhexuan Zhu
Department of Computer Science
University of Toronto
yuezhexuan.zhu@mail.utoronto.ca

Abstract

Inspired by the neural language models introduced in the lecture, we have decided to design a neural network that can predict the labels of algorithms and data structures required to solve a coding question. To accomplish this, we have chosen to utilize the problems from Leetcode. With relatively small data-set size, we use techniques including data augmentation and transfer learning to improve model performance. Our goal is to create a model that can predict the most likely algorithm for a given coding question based on its text description.

For this project, we have structured two models on the basis of BERT[1] and LSTM[2] along with specialized fine-tunings. Our analysis shows that BERT performs better than LSTM, and has potential for achieving higher accuracy.

All related documents and data can be found inside our github repository[1].

1 Introduction

All members of our team share a common interest in solving coding questions, which we perceive as a means of enhancing our logical reasoning and programming proficiency. We have observed that the identification of an appropriate algorithm constitutes the initial and crucial step towards devising a solution. Any biased identification can lead to a series of fruitless tries until the correct algorithm is identified.

To address this issue, we aim at designing a Neural Network that can effectively generate potential algorithms and data structure labels for a given coding question based on its textual description. Our proposed model would take a coding question description and predict data structure and algorithm labels that feature in real solutions.

After selection and feasibility analysis, we decided to build our model on the basis of BERT[1], a multi-purpose transformer architecture, and LSTM[2], a classic model for sequence classification.

2 Data Preprocessing

Instead of fetching data from Leetcode, we decide to use the processed dataset from gzipChrist[3] on Kaggle. The dataset comprises 1825 problem descriptions and their corresponding required

[1] Github Link here

algorithms or data structures. To facilitate analysis, we developed a Python script to vectorize the labels associated with each problem. Following processing, the dataset contains 1825 rows and 44 column, some of the processed data are shown in Table 1. The first column representing the problem description and the remaining columns indicating the presence or absence of the corresponding labels, denoted by binary values of 0 or 1. There are 43 labels in total.

Table 1: **Data After Processing**

Description	**Array**	**DP**	**String**	**Math**	$\cdots$	**Hash Table**	$\cdots$
Given an array of integers ...	1	0	0	0	$\cdots$	1	$\cdots$
You are given two non-empty ...	0	0	0	1	$\cdots$	0	$\cdots$
...	...	...	...	...	...	...	...

3 Model Architecture

3.1 BERT Pre-trained Model

3.1.1 General Design

On the basis of BERT, we selected the pre-trained CodeBERT[4] model and its corresponding tokenizer due to its relevance towards coding. Since there's a mismatch between BERT output and target labels counts, we added two extra fully connected layers to reduce output dimension to 43.

Algorithm 1 Forward Propagation Route

```
Input ← Translate(Input, en→ rand→ en)          ▷ Data Augmentation
Input, Mask ← tokenizer.encode(Input)           ▷ Input Encoding
Mid1 ← CodeBERT(Input, Mask)                    ▷ BERT Layer
Mid2 ← Linear(Mid1, 768 → 512)                  ▷ FC #1
Mid3 ← Linear(ReLU(Mid2), 512 → 43)             ▷ FC #2
Output ← Sigmoid(Mid3)                          ▷ Final Activation
```

Considering the fact that each description has more than one label, we first use the sigmoid function to regularization outputs and then apply binary cross-entropy loss as a loss function.

Through several rounds of tuning, we use 2×10^{-5} for the learning rate, 8 as the batch size for data loader, Adam for optimizer with weight decay of 0.01, and set the epoch to be 30.

Algorithm 2 Single Training Step

```
for data in Dataloader do:
    input, target ← data
    output ← model(input)
    loss ← BCELoss(output, target)              ▷ Binary Cross-Entropy Loss
    loss.backward()                             ▷ Pytorch Function
    Adam.step()                                 ▷ Optimizer Step
    Adam.zero_grad()
end for
```

3.1.2 Input Encoding

1825 inputs are less than the ideal amount, so we perform data augmentation on the dataset loader by translating loaded problem descriptions to a randomly selected language and then back to English before tokenizing the text, which is referred to in the algorithm box above.

As shown in histogram3, as most descriptions in range from 50 to 400 characters, we use 512 as the maximum length for input encoding.

3.1.3 Validation & Analysis

For calculating validation accuracy, we convert the predicted tensor to binary form using 0.5 as threshold and employed `sklearn.metrics.accuracy_score` to get the accuracy score, which basically evaluates the percentage of complete matches between prediction and target. Then, we

recorded the loss for both training and validation and accuracy for validation, with results shown in Figure 1.

In comparison to the random selection of one label from a pool of 43, which yields a probability of $\frac{1}{43} = 0.023$ for choosing the correct label, our model demonstrates a significant enhancement, achieving a peak validation accuracy of 0.24 at epoch 14.

In addition to evaluating complete matching, we proceeded to compute the label-wise matching accuracy between predictions and targets. Our analysis revealed a **0.9632** accuracy on the reserved test set, with an average discrepancy of 1.582 labels overall. This discrepancy may arise from predicted labels that do not exist or from the omission of actual labels. In conclusion, the BERT model's performance is satisfactory, and we will proceed to examine LSTM models for further comparison.

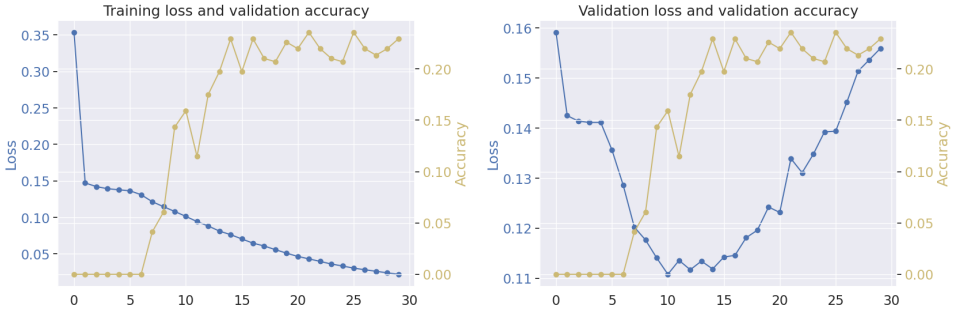

Figure 1: BERT train loss V.S. Epoch, validation loss V.S. Epoch and Accuracy V.S Epoch

3.2 LSTM Model

3.2.1 General Design

In order to facilitate model training, it is essential to transform the input word sentences into vector representations. The tool employed for this purpose is the `keras.preprocessing.text.Tokenizer`. Within the context of an LSTM[2] model, input sequences necessitate a fixed length to ensure efficient processing. Consequently, the maximum length of tokenized inputs is chosen as the target length, and the remaining inputs are padded accordingly.

The proposed model will incorporate an embedding layer for the input and utilize an LSTM layer for processing the output. To achieve regularization of the outputs, a sigmoid function will be applied. Furthermore, the binary cross-entropy loss function will be employed to quantify the discrepancy between the predicted and actual outputs.

Through several rounds of tuning, we use 2×10^{-5} for the learning rate, 128 units for LSTM layer, maximum description length as the padding length, 32 for batch size, Adam for the optimizer with weight decay of 0.01, and set the epoch to be 60.

Algorithm 3 Forward Propagation Route

```
Input ← Translate(Input, en→ rand→ en)        ▷ Data Augmentation
Input ← tokenizer.encode(Input)               ▷ Input Encoding
Mid1 ← LSTM(input, unit=128)                  ▷ LSTM Layer
Mid2 ← Linear(Mid1, 128 → 43)                 ▷ Dense Layer
Output ← Sigmoid(Mid2)                        ▷ Final Activation

Model Fitting by Tensorflow:
mode.fit(X, y, epochs=60, batch_size=32, validation_split=0.2)
```

3.2.2 Training & Validation

For the purpose of comparison, we use the same schema as the BERT model, results are shown in Figure 2. Recall the raw accuracy of arbitrarily picking one correct label with probability $\frac{1}{43} = 0.023$, the LSTM model shown best validation accuracy of our model reaches 0.138 at epoch 57 also a noticeable improvement. Same as the BERT model, we calculated label-wise matching accuracy

between prediction and target, where we get 0.9470 on the reserved test set - an average of 2.279 for different labels overall, which can be either from prediction labels that don't exist or missing actual labels.

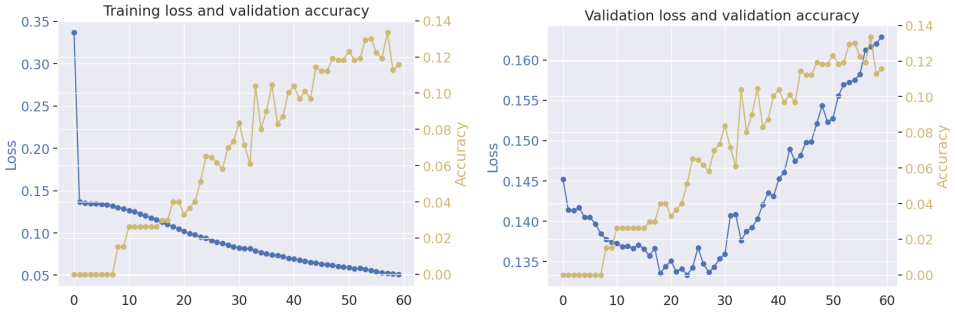

Figure 2: LSTM train loss V.S. Epoch, validation loss V.S. Epoch and Accuracy V.S Epoch

4 Comparison of Models

In the realm of multi-label text classification, our experiment demonstrated superior performance by BERT, and in case of having sufficient computational resources, BERT proves to be the prior choice. Specifically, BERT has advantage over three aspects:

First, BERT model achieved a 0.24 validation accuracy rate, surpassing the LSTM model's 0.138. Furthermore, both BERT and LSTM exhibited comparable label-wise accuracy rates of 0.9632 and 0.9470, respectively. This outcome may be attributed to BERT's ability to effectively capture contextual information through its attention mechanism, while LSTMs potentially struggle with extended text sequences.

Second, BERT's validation accuracy converges around epoch 15, while LSTM maintains an increasing trend till epoch 55. So, despite BERT having more per-epoch training costs, it converges faster.

Third, BERT outperforms LSTMs in handling out-of-vocabulary words due to its utilization of subword tokenization for addressing rare and unknown terms.

5 Limitations

In examining our dataset, several limitations must be acknowledged to ensure a comprehensive understanding of its potential impact on the analysis.

Firstly, the dataset is relatively small. In particular, a small dataset may not provide enough information to enable models to learn the key relationships between words or phrases in the text to identify patterns effectively and generalize effectively to new data.

Secondly, our model is limited to handling input from a text-based dataset (i.e. removing images from description), rendering it incapable of addressing problems that necessitate image understanding.

Thirdly, the labeling of LeetCode problems is inherently biased on associated with widely-used data structures and algorithms which indicates that the model's performance on non-Leetcode-like question-generation input and questions with uncommon algorithms may be significantly compromised.

Lastly, pre-trained tokenizers may separate formulas and characters from their associated words, making it difficult to learn relevant information from such text on certain topics. This can pose a challenge for our models that rely on tokenization to understand and process text data.

6 Future Expectations

With multiple online judging platforms, we may continue on tuning our model for a generalized coding question labeling network not limited to only Leetcode questions. Also, with BERT's strong competence, it's promising to further employ newly emerged models, including GPT and T5 to see if these models can reach human-level recognition. And, on the basis of using a pre-written tokenizer, a self-defined tokenizer and encoder may have the potential to further enhance model performance.

References

[1] Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. ArXiv. /abs/1810.04805.

[2] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," in Neural Computation, vol. 9, no. 8, pp. 1735-1780, 15 Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[3] gzipChrist. (2023, February). Leetcode Problem Dataset, Version 8.82. Retrieved April 10, 2023 from https://www.kaggle.com/datasets/gzipchrist/leetcode-problem-dataset.

[4] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., & Zhou, M. (2020). CodeBERT: A Pre-Trained Model for Programming and Natural Languages. ArXiv. /abs/2002.08155

A Appendix

Leetcode question description length histogram:

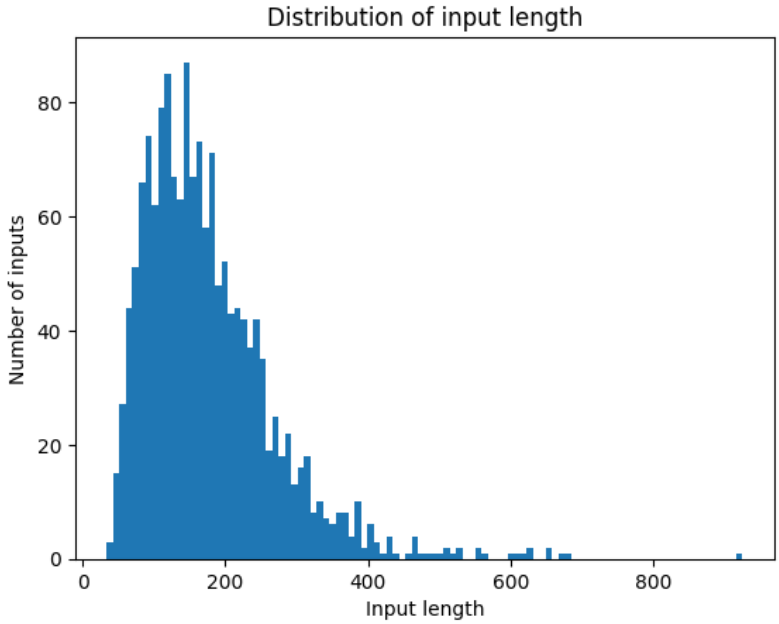

Figure 3: Input Length Distribution Histogram