BERT-Based Salary Prediction for Data Analyst Positions: An Application of DL on Sequences

Shaohong Chen
shaohong.chen@mail.utoronto.ca

You Peng
fred.peng@mail.utoronto.ca

Chengyuan Xue
kurt.xue@mail.utoronto.ca

Lingyun Jiang
johnnyl.jiang@mail.utoronto.ca

Abstract

We fine-tune a pre-trained BERT model and link it to a RNN trained on the data analyst jobs dataset. This BERT-Linked RNN takes in a sequence of text describing job requirements and other information of the job, and outputs a class of two values indicating the range of the salary. The model is further compared with classification tree as a baseline, and a RNN model which uses BERT features encoding. As a result, the BERT-Linked RNN performs the best with a 84% test accuracy.

1 Introduction

In recent years, due to the explosive growth of digital data, there has been an increasing demand for skilled professionals capable of analysing complex datasets. On the other hand, more people choose to become a data analyst in this era of Big Data. While the market is full of participants, a critical aspect of the data analyst role is the negotiation of compensation, which often depends on the specific skills, experience, and job requirements associated with the position. However, companies in different fields may have specific requirements for the data analysts they want, and different candidates for such jobs also have varying levels of proficiency. Such variations make it difficult to set up a universal salary standard. Traditionally, salary negotiation in the job market has been influenced by subjective factors, making it challenging for both job seekers and employers to accurately assess the value of a given role.

To address this challenge, we propose a machine learning based approach, which utilizes advanced natural language processing (NLP) techniques to automate the process of salary prediction for data analyst positions. Specifically, we use the pre-trained Bidirectional Encoder Representations from Transformers (BERT) model and fine-tune it on the dataset containing various data analyst jobs around the world. The model will learn the context in the data science fields, and predict the salary range of a job given a sequence of text describing its job requirements.

Our study aims to demonstrate the efficiency of fine-tuning the BERT model for multi-class classification tasks in the context of salary prediction for data analyst roles. By training the model on a comprehensive dataset of job descriptions and associated salary ranges, we seek to develop a robust predictive model capable of accurately estimating salaries based on the textual features of job postings.

Overall, our BERTRNN model outperforms the baseline decision tree and RNN models. And it reached a accuracy of 83.98% in test set. Candidates with different experience levels and recruiters for companies of all sizes can have an estimation of their own or their employees' salary based on the model's output. This helps to promote a fair and transparent salary negotiations in the job market.

2 Related Work

The dataset we are analyzing is tabular data, which consists of a sequence of categorical data (e.g., job descriptions, competitors, and etc.), and corresponding salary ranges as the only numerical data. The application of deep learning (DL), particularly Natural Language Processing (NLP), in interpreting and predicting outcomes from these categorical tabular data has gained significant achievements over recent years. We will briefly review the most widely-used approaches for tabular data modeling in this section.

2.1 Traditional Modeling Methods

Historically, the analysis of tabular data is mainly dependent on statistical methods and classical machine learning (ML) methods including logistic regression, and tree-based ensemble methods such as gradient-boosted decision trees (GBDT), which are typically the top-choice in various ML methods [1]. These methods have been favored for their interpretability, high performance, and ease of efficient training [2]. Although there is still no universally superior solution among GBDT and deep learning models, these tree-based models have limitations on feature extractions (especially with categorical data) and processing sequential data while Recurrent Neural Networks (RNNs) and transformers excel in handling sequential dependencies [1, 2].

2.2 Deep Learning Methods (NLP)

With the rise of deep learning, NLP techniques have been increasingly applied for tabular data modeling. A recent survey conducted by Fang et al (2024) explores a comprehensive application of Large Language Models (LLM) on tabular data, highlighting the different applications of LLMs from tabular data prediction, to data synthesis, and Question Answering (QA). It provides a taxonomy of metrics and techniques for LLMs' application, revealing how LLMs could be trained on tabular data and fine-tuned to improve the model performance [2].
The above survey also points out the benefits of transfer learning, using pre-trained model (with sufficient syntactic and semantic understanding of language by training on large corpus of data) to train on task-specific data to improve the model performance.

2.3 Bidirectional Encoder Representations from Transformers (BERT)

BERT is such a pre-trained model, which is similar to other transformer encoders, except that it is bidirectional. The key innovations of BERT are the "masked LM" (MLM) in the pre-training step which randomly masks out tokens in a sequence, and the technique called Next Sentence Prediction (NSP), which enables the model to understand the sentence relationships [3]. These new techniques allow BERT to become a powerful pre-trained model in solving a large range of NLP tasks.

Overall, the incorporation of NLP into the analysis of tabular data represents a significant shift from traditional methods to advanced deep learning models. Our project builds upon these methods, applies them specifically to the challenge of salary prediction using the categorical data, and compare their performances.

3 Model Selection

We introduce various implementations of neural network models that leverage the BERT architecture and compare them with the baseline Gradient-Boosted Decision Trees (GBDT) model and traditional RNN model. Each model has a different architecture used to explore the capabilities of the bert model for salary prediction task. Due to the limited size of dataset, we choose to use fewer layers in the architecture.

3.1 Traditional RNN

We built the model with a GRU(num_layers = 2, Hidden_size = 512) based on embeddings generated from an initial embedding layer(embedding_size = 300) and a linear layer for salary output. This model can be considered as a baseline for those integrating pre-trained transformers. It offers insights

into the performance of simpler, more traditional RNN architectures in contrast to the more complex, transformer-based models.

3.2 BERT Fine-Tuning

This model focuses on the direct application of BERT for specific fine-tuning tasks by adding an additional linear layer. Without complex design, this model simplifies the architecture by connecting a single fully connected linear layer directly to the output of BERT. This model is particularly suited for tasks where the primary requirement is to harness the deep linguistic understanding embedded within BERT, with minimal architectural modifications, for efficient task-specific tuning.

3.3 BERT Feature Based RNN

We use BERT as word embedding layer and use the word embedding as input to a GRU(num_layers = 2, Hidden_size = 512) and a linear layer. The model is designed to extract and utilize a richer set of features from BERT by averaging over multiple hidden layers, the result is used as the word embedding of the token input. This configuration aims to capture a broader context by utilizing the nuanced feature set extracted from BERT's layers, enhancing the model's ability to interpret complex linguistic patterns.

3.4 BERT Linked RNN

This model perform an end-to-end connection between a BERT model and a GRU(num_layers = 2, Hidden_size = 512), and the salary output is given by the additional linear layer. In this model, we assume that the BERT model encapsulates all necessary information, which makes it potentially faster and less resource-intensive.

3.5 Gradient-Boosted Decision Trees (GBDT) - Baseline Model

In comparison with other deep learning models, we select GBDT - the tree-based model - as our baseline model. We simply use the built-in Gradient-Boosting Regressor of sklearn, and then transfer it using the built-in Multi-Output Regressor so that it accepts our salary prediction (which contains a lower bound and an upper bound) as target. Then we can fit our model with training data and predict the outcome.

4 Experiment Setup

4.1 Data Selection

In our experiment, we used a public dataset containing around 1000 pieces of detailed job descriptions and salary ranges for data science positions collected by Kaggle Users from scrapping job postings on glassdoor.com in 2021. This dataset stood out to us because it contains a string description of job postings for each entry of data, thus making it possible for us to perform natural language processing on the input tokens. After selection, we decided to use the following features provided in the original dataset for maximum information while minimizing less-related data: [*Job Title, Job Description, Location, Headquarter Location, Company Name, Founded Year, Company Size, Type of Ownership, Industry, Sector of Position, Degree Requirement, Skill Requirements*] in our input. Our task will be predicting 2 values, the upper and lower bounds of the salary for the job.

4.2 Preprocess

The raw dataset we obtained was cleaned for exploratory data analysis only, so we will further preprocess the data for our model training pipeline. The job details include not only the string provided by employers but also the skill requirements, and company metadata, as separate categorical columns in the data sheet. Since we are using sequential deep learning inputs for all of our models, we will concatenate all these related information into one string, to the amount of useful tokens captured by our models in training.

In our pre-processing step, we concatenated the features above into a large string, kept only alphabetic characters, and used space to replace all the other separation characters to separate words, to perform tokenization with pre-trained BERT Tokenizer. We will use this processed array of tokens as the input for our models. Then we combine the values in columns 'Lower Salary' and 'Upper Salary' into a tuple (lower, upper) for each data entry, as our training targets.

4.3 Training and Evaluation

Performance Metrics: Since our task is designed to be a range prediction with upper and lower bound predictions, we used Mean Square Error(MSE) instead of traditional classifications as the loss function over the two predicted logits. In our experiment, we carefully selected Intersection over Union as our criteria for accuracy. Although it may create a slight discrepancy between loss and accuracy due to different calculation metrics (MSE calculates value differences while IoU checks range overlapping), the range overlap will be more helpful in determining whether we are giving an accurate prediction. Also, a smaller IoU almost implies that at least one of our predictions is largely different from the target, thus making the accuracy and loss function closely tied, even if they don't regress in the same trend.

Model Pipeline: We split the dataset into 80% for training, 10% validation and 10% test set. In each epoch, we reshuffle the training set, then the training loop iterates through the entire training dataset in batches with batch size 10 using the Adam optimizer to apply momentum in our back-propagation, to achieve faster convergence. By the end of each epoch, we use the validation set to calculate a validation loss, and we keep monitoring the loss to apply early stopping in case the model overfits the training data, in which case the validation loss will stop decreasing for a consecutive number of epochs. We set up a patience parameter to control how many epochs we monitor to decide whether we early stop the training with a default value of 10. After the training, we evaluate the performance of our model by calculating the loss and accuracy on the test dataset, and we will use these measurements as the main dependent variable in the performance comparison between models. For our models involving pre-trained BERT, we froze the parameters of BERT to 1) ensure we don't lose already trained features, and 2) save computational resources. With the BERT base model frozen, we will be fine-tuning the BERT classification model by training our customized final layers on top of BERT base.
For our baseline model, we combine the validation and test sets into one test set, fit the model using the training set, and predict the range of salary using the large test set. Then we follow the same accuracy function to get the final test accuracy.

User Interface: We also implemented a user interface where users can load a trained model and input a job description text via a text file, to predict a salary range for the position. A detailed usage description can be found in the project GitHub page.

5 Results and Evaluation

Best Performance of Salary Range Prediction Over Models					
Model Name	Decision Tree (GBDT)	Traditional RNN	BERT Linear Fine-Tuning	BERT Feature Based RNN	BERT Linked RNN
Loss	/	819.98	1333.40	1767.19	141.55
Accuracy	69.81%	61.75%	40.88%	37.53%	83.98%

5.1 Decision Tree (GDBT)

HyperParameters: number of estimators=150, learning rate=0.3, max depth=3

In this baseline model, we simply follow the above pipeline and fine-tune the hyperparameters of GDBT to achieve a better performance. The performance is measured using the same accuracy function as the other models (which computes the average Intersection of Union between prediction salary range and the target range). By increasing/decreasing one of the parameter and fixing the other two, we fine-tune the model by training the model several times and choose the setting with the highest average test accuracy.

For example, we fix the number of estimators=150 and max depth=3, and increase the learning rate from 0.2 to 0.3. Running 3 times give the test accuracy of 0.6946, 0.7135, 0.6904, whose average=0.6995 is greater than that of learning rate=0.2 (accuracy=0.6345). Doing again for 0.3 to 0.4, the average accuracy decreases and thus we know that our model performs well at learning rate=0.3.

5.2 Traditional RNN

HyperParameters: batch size=10, learning rate=5e-3, patience=5, max epochs=50

In this model, we use GRU architecture as our RNN layer. The tokenized data are mapped to embedding space, fed into GRU layer, and then the output are derived through a fully connected linear layer. From the plot, we could see that the training loss drops quickly at the early epoch and gradually converges to a small number close to 0, while the validation loss keeps decreasing at a smaller rate and converges to a certain value around 750. The validation accuracy of our model starts at a relatively high value ($\approx 42\%$) and keeps increasing to about 70% with some tiny fluctuations. Based on the trends of each curve and the quantitative data, we could infer that our model is performing well because both the loss curves are decreasing and converging to a small value, while the accuracy curve keeps increasing and converges to a relative high value. Since we set the patience=5, we actively apply early-stop to our model so that we could ensure the model doesn't overfit. By training the model several times, and getting similar results with high validation accuracy (varies from 65% to 70% under different hyperParameters settings), we could say that our model is performing well on this salary prediction task.

5.3 BERT with Linear Final Layer

HyperParameters: batch size=10, learning rate=1e-4/5e-4, patience=5, max epochs=50

In this model, we are Fine-Tuning BERT by adding a fully connected layer directly after BERT. With the default hyperparameters, the model did not yield optimal results. The loss dropped and accuracy increased quickly for the validation dataset in the first 20 epochs during our training, with the loss dropping from over 10000 to 1500 and accuracy increasing to around 37%. However, the convergence becomes much slower after the initial epochs. We also had to use $patience = 20$ for this model, because the validation loss for this model has a large variation due to the small learning rate. For the later 400 epochs, the model only converged from around 30% to around 44%. After training for all 500 epochs, the test accuracy is only 34.52%, with a test loss of 1436.77, suggesting our model may have potential overfitting as well. Interestingly, the validation loss is still showing a decreasing trend with a small slope. However, we decided not to keep training the model for more epochs since the overall accuracy did not increase with the decrease in validation loss.

Later we experimented to increase $learningRate = 5e^{-4}$ to see if we could avoid the variation in validation loss. The model early stopped at epoch 196, with a test accuracy of 40.88%. This result proves the model with $learningRate = 1e^{-4}$ is overfitting, but it still did not outperform our baseline model. The validation loss and accuracy plots during training for both learning rates are included in the appendix.

5.4 BERT Feature Based RNN

HyperParameters: batch size=10, learning rate=0.001, patience=10, max epochs=200

This model uses BERT to provide feature embedding, and the extracted features are fed into an additional RNN to be further trained. The loss curve for the training and validation set all decreased smoothly and became flattened after 20 epochs, this shows that the model was not over-fitting. To make sure the model doesn't stuck in a local optimum, we increased the learning rate from 1×10^{-4}, 0.001, 0.005, till 0.01. We also adjusted patience from 5 to 10 to allow enough training iterations for the model to walk out of the local minimum. However, all learning rates eventually reached the same level of accuracy around 40% and loss around 1500 for the validation set, then the model was forced to early stop. This suggests that the model might just performed poorly due to a small training size. In addition to that, the BERT model's parameters are not fully

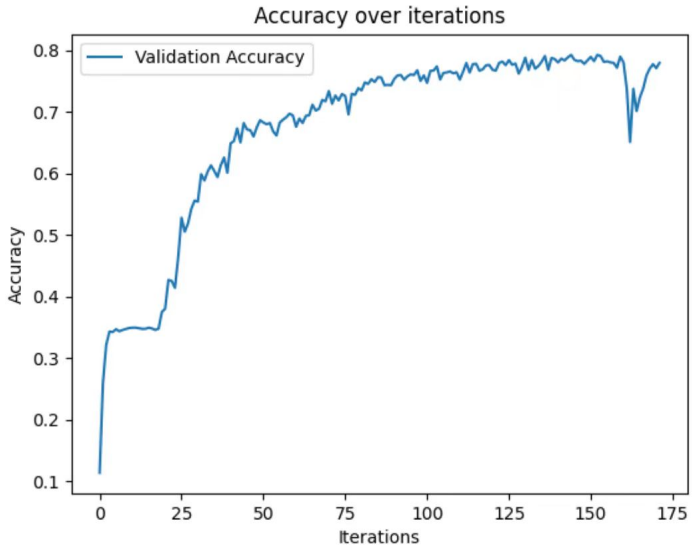


Fig. 1. Validation Accuracy of the best performing BertRNN model

fine-tuned since some of the layers are frozen, so the embedding learned by BERT may not suit well for our task. The loss curve and accuracy curve are provided in the appendices.

5.5 BERT Linked RNN

HyperParameters: batch size=10, learning rate=0.001, patience=20, max epochs=500

Although the pattern was not as obvious for other models, we found the sample order we had in the data loader may have a large influence on the quality of our trained model. Since we had shuffle enabled when sampling data into batches, we had several different models trained. While most of the models we trained with this set of hyperparameters had both validation and test accuracy around 60% - 66%, one of the models reached a test accuracy of 83.9%.

In the experiment that returned the best-performing model, we had an early stop in epoch 171, yielding an outperforming result in both the validation and test dataset, in comparison with the other models. By the end of the training iterations, the validation accuracy reached 77.9%, with a validation loss of 257. The model managed to reduce training loss to just 6 from the initial > 10000 and was not overfitting, making the test dataset reach 83.9% in its accuracy. The accuracy trend for this experiment is given in Figure 1. We can see the validation accuracy kept increasing with fluctuation after an initial bottleneck between epochs 5 to 20. It increased to almost 80% after the 125-th epoch and then triggered early stopping.

With the graph, we can see that although the overall accuracy of this model increases on the validation set, the value of accuracy may have a large variance even though the trend shows an overall increment. As a result, we increased the patience to 30 to experiment with the performance. We then witnessed the increment in accuracy kept increasing while the number of iterations increased. Unfortunately, due to the large variance, early stopping was triggered by the end of epoch 143. The test accuracy of this model was 75%, which is significantly higher than most of the models we had with default hyperparameters, and the loss was 388.60. The detailed plot of loss and accuracy increasing with epochs for this experiment is included in the appendix. With this experiment, we are confident that we may be able to achieve a more stable result with some more experiments with different values for patience, that ensure performance while avoiding the model overfitting training data.

6 Conclusion

In the salary range predicting task we are discussing in this report, we initially set the Decision Tree and traditional RNN model as our baselines. During our experiments, we found that the model using BERT encoded features as RNN inputs, and the model applying the common fine-tuning method of BERT by adding another fully connected layer directly both did not yield desirable results. These may caused by the limited fine-tuning that we have done since we had a very limited amount of data(less than 1000 pieces), and we have frozen the parameters of the BERT base model due to computational resource and time constraints. Also, our data may not be fully aligned with the features learned by pre-trained BERT, so we may not be able to utilize BERT-encoded features in the BERT Feature RNN model.

In comparison, although we had similar constraints, the traditional RNN and Bert Direct Link RNN both showed convincing accuracy after a certain amount of iterations. Traditional RNN overfitted very quickly that early stopping was triggered in just over 20 epochs, while the BERT Direct Link RNN was able to reach a higher overall accuracy without overfitting for almost 200 epochs.

Overall, the BERT Direct Link RNN model has given us the best accuracy and loss over epochs on both the validation dataset and the test dataset, showing it could be the most desirable model that we would like to use in our salary prediction task. The follow-up experiments with this model also tell us the possible improvements in BERT Direct Link RNN, if we can carefully select an optimal patience value. We were not able to experiment on the choice but the experiments have shown us a way to continue our exploration.

Meanwhile, there are some ethical considerations when using the model to predict salary. Since similar jobs in different countries can have different salary expectations, and some countries are the minority in the training data. It's important to regularly evaluate the model for fairness across different demographic groups to prevent discriminatory impact, otherwise salaries for jobs in low GDP countries may be overvalued by the model. On the other hand, the model is not transparent enough to show which job requirement is influencing the salary the most. Lastly, the disparity between the model's confidence inside and outside the training set can have a negative impact on the privacy of the training samples since the model doesn't satisfy the differential privacy. We can replace with a differentially-private SGD to mitigate this issue in the future.

7 Description of Individual Contribution

You Peng: Coded accuracy computation and curve plots. Fixed bugs occasionally. Ran Bert Feature Based RNN and tuned hyper-parameters based on the performance. Wrote Abstract and Introduction section. Wrote Bert Feature Based RNN performance in the Result section. Wrote ethical considerations in the Conclusion section.

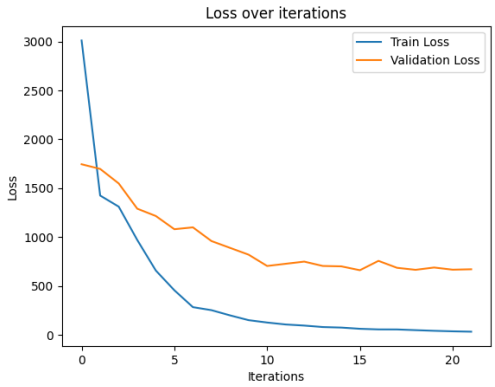
Lingyun Jiang: Implemented RNN and GDBT models, and tokenization function. Fixed bugs occasionally. Ran and fine-tuned these two models based on their performance. Wrote Related-Work and corresponding parts of the "Results and Evaluation" sections.

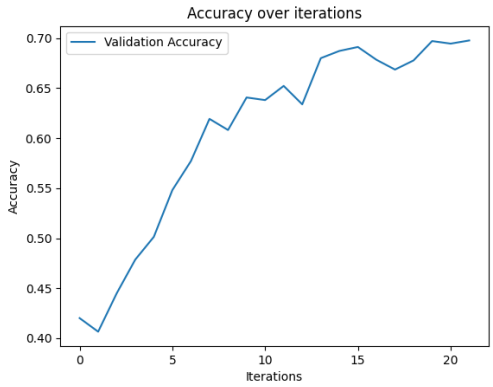
Chengyuan Xue: Implemented Bert Feature Based RNN, BERT Fine-Tuning and BERT Linked RNN. Coded the training and validation methods and basic pipeline of the model training. Ran Bert Linked RNN and tuned hyper-parameters based on the performance, collected related results. Wrote Model description section.

Shaohong Chen: Implemented Preproecessing function and user interface for prediction, debugged cuda-related functionality and improved code readability. Ran the experiments for salaryBERT model class which is the Fine-tuned BERT model with one additional linear layer. Wrote the performance analysis for the model above and the Bert Direct Link RNN model. Also wrote the Experiment Setup and Conclusion parts of the report.

Appendices

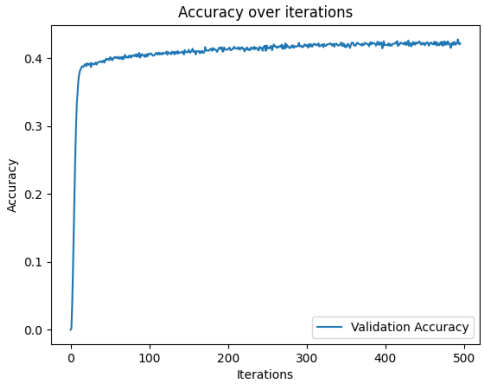
Loss and Accuracy plot for Model of Traditional RNN

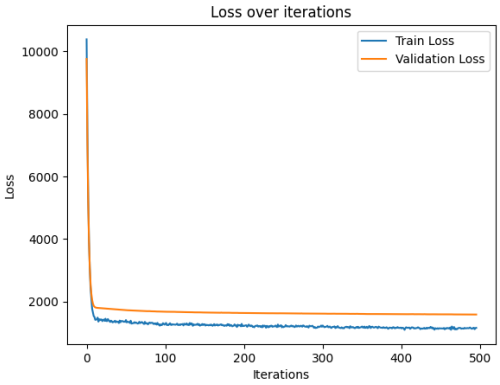




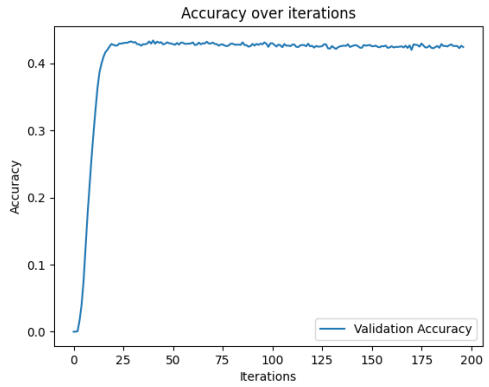
Loss and Accuracy plot for Model of BERT with Linear Layer

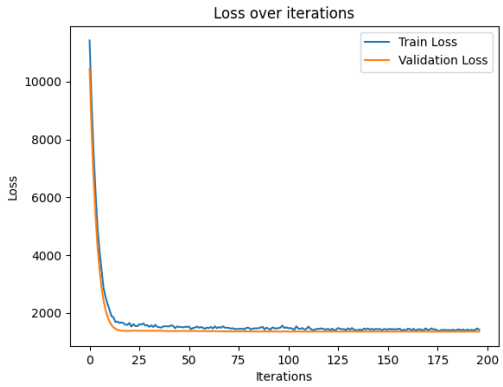
Learning Rate $1e^{-4}$



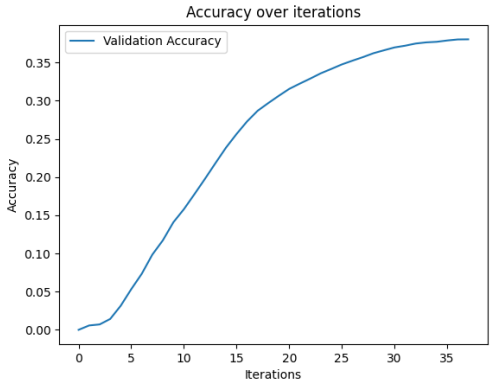


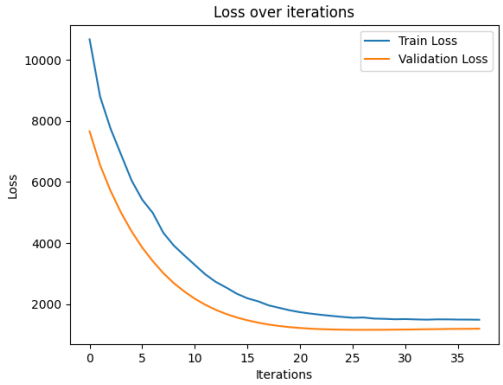
Learning Rate $5e^{-4}$





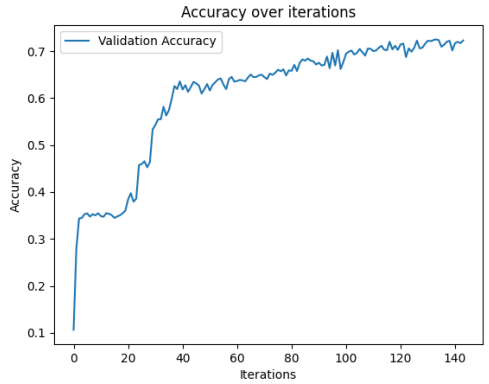
Loss and Accuracy plot for Model of Bert Feature Based RNN

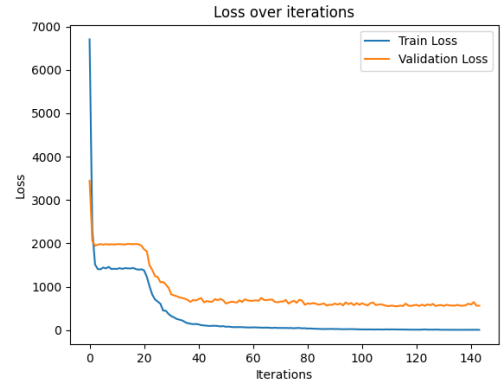




Loss and Accuracy plot for Model of Bert Direct Link RNN

Patience 30





References

[1] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko.Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34:18932–18943, 2021.

[2] Xi Fang and Weijie Xu and Fiona Anting Tan and Jiani Zhang and Ziqing Hu and Yanjun Qi and Scott Nickleach and Diego Socolinsky and Srinivasan Sengamedu and Christos Faloutsos. *Large Language Models(LLMs) on Tabular Data: Prediction, Generation, and Understanding - A Survey*, abs/2402.17944, 2024.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT:Pre-training of Deep Bidirectional Transformers for Language Understanding.* In Pro-ceedings of the 2019 Conference of the North American Chapter of the Associationfor Computational Linguistics: Human Language Technologies, Volume 1 (Long andShort Papers), pages 4171–4186, Minneapolis, Minnesota. Association for Computa-tional Linguistics. 2019.