RecoBERT: Recommendation with Bidirectional Encoder Representations from Transformer

Candidate:

Kar Hoe TAN (Andy)

(kar.tan.21@ucl.ac.uk)

UCL Supervisor:

Prof. Brooks PAIGE

(b.paige@ucl.ac.uk)

Huawei Supervisor:

Dr. Xianghang LIU

(xianghang.liu@huawei.com)

Dr. Alejo LOPEZ

(alejo.lopez.avila@huawei.com)

This report is submitted as part requirement

for the MSc Data Science & Machine Learning

at

University College London

It is substantially the result of my own work except where explicitly indicated in the text. The report will be distributed to the internal and external examiners, but thereafter may not be copied or distributed except with permission from the author

September 12, 2022

Abstract

Learning user's preferences from their historical behaviours is crucial in recommender systems. There has been a large body of research on sequential neural network that encode user's historical behaviours from left to right into universal representation for making recommendations. However, modeling user behaviour sequences from left to right is impractical as user's preferences may not strictly follow an ordered sequence. To address this limitation, we proposed a bidirectional self-attention model called RecoBERT to model user behaviour sequences. RecoBERT is a smaller version of a BERT model adapted to the recommender system domain. To learn useful features from user's historical behaviours, we adopt two objectives to sequential recommendation, classifying next sequence and predicting randomly masked items in user sequences. We pre-train RecoBERT until convergence and discuss how the hyperparameters and parameters affect the performance on the two pre-training tasks. In this way, we could extract useful features by reusing weights from the pre-trained RecoBERT for downstream tasks in fine-tuning. We conduct extensive experiments to show the effectiveness of transfer learning with pre-trained RecoBERT in two downstream tasks, predicting multiple user profile information. In fine-tuning downstream tasks, we show that freezing the pre-trained RecoBERT model weights and fine-tune with a few additional output layers achieves better performance relative to fine-tuning the entire architecture, and non transfer learning MLP model.

For accompanying code, see

https://github.com/AndyTKH/DSML_Project_Reco.git

Acknowledgements

I would like to thank my industrial supervisors Dr. Xianghang Liu, and Dr. Alejo Lopez for their useful comments, remarks and engagement during this project. Furthermore, I would like to express my gratitude to my family, who supported me for the postgraduate study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter we discuss the challenges and prospects of recommender systems, and focus on how self-supervised learning (SSL) can solve these challenges, which motivated transformer research in this project. Then, we summarize all contributions we made to sequential recommendation, follow by outlining the structure of the thesis.

1.1 Project Motivation

In recent years, recommender systems have been widely deployed in many online platforms, e.g., *Netflix* (online movies & TV shows), *Youtube* (online videos) , *Amazon* (e-commerce), as a tool that provides suggestion of items to users for a pleasant user experience whilst driving company incremental revenue. Hence, modeling user's preferences from their historical behaviours is crucial for recommender systems. Unlike image annotation, recommender systems is faced with data sparsity issue due to limited and unlabelled users historical clicks/records. To overcome this, self-supervised learning (SSL) emerges as a popular technique to extract useful information and transfer knowledge from unlabelled data.

There have been various self-supervised Recurrent Neural Network (RNN) for sequential recommendation [2, 4, 13] . The RNN model encodes user's historical behaviour sequences from left to right and make recommendation based on the encoder representations. RNN is a left to right unidirectional model and we argue that

modeling user's historical behaviours from left to right is impractical in real-world application. This is because user's behaviour/preferences do not follow any strict order. For example, a user's sequential clicks on comedy, follow by a science fiction movie. The order of the sequence does not hold important information, but the context of the sequence does hold important information about the user. To address the unidirectional limitation, we seek to use bidirectional model that encodes representations from Transformer. Inspired by the success of BERT [1] in language context understanding. We propose a sequential recommendation model called **RecoBERT**, which employs multi-layer bidirectional encoder, pre-trained on Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) tasks, whose objective is to reconstruct the original user's historical clicks behaviour from the augmented data, as illustrated in Figure 1.1 below. To our best knowledge, we only found one paper (BERT4Rec)[11] that uses BERT model for sequential recommendation. The Bert4Rec model randomly mask the input sequences and then predicts the masked items based on its surrounding context. Author of Bert4Rec only conducts pre-training experiments on BERT, and so we have no idea the fine-tuning performance on downstream tasks when utilizing transfer learning from a pre-trained BERT model. Hence, to demonstrate how BERT can be pre-trained for sequential recommendation and its performance on downstream recommendation tasks, we introduced **RecoBERT** model.

Conventional BERT [1] pre-trains the final transformer encoder layer by considering the first special [CLS] token output from each sequences for Next Sentence classification task, as shown in 1.2. The major problem is that this first [CLS] token output usually does not represent a good summary of the semantic content of the input, resulting in poor performance for its Next Sentence classification task. To tackle this problem, our RecoBERT model considers averaging all tokens for each sequence in Next Sentence classification task, as illustrated in 1.3. To predict for the Next Sentence classification, we split each user's historical behaviour sequence into two sequences, sequence A and sequence B, and then predict whether sequence A

consecutive sequence is sequence B. In addition to training for Next Sentence classification task, RecoBERT also trains on Masked language Modelling task. Specifically, we randomly mask a portion of the input sequence, and then predict for the masked items. To demonstrate how well RecoBERT has learned a universal user representations that could be used for different downstream task. We conduct fine-tuning experiments to predict two user profiles, gender and the age group. Our experiments show that freezing the pre-trained RecoBERT model weights and fine-tune with a few additional output layers achieves better performance relative to fine-tuning the entire architecture, and MLP model.

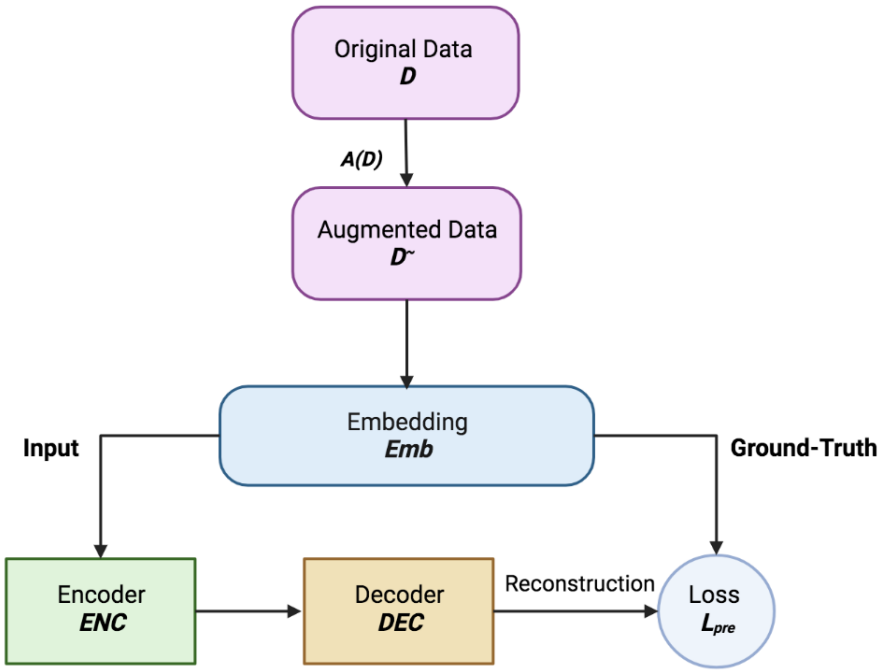

FIGURE (1.1) Masked Language Models (MLM) Transformer

1.2 Our Contribution

The contribution of this project is as follows:

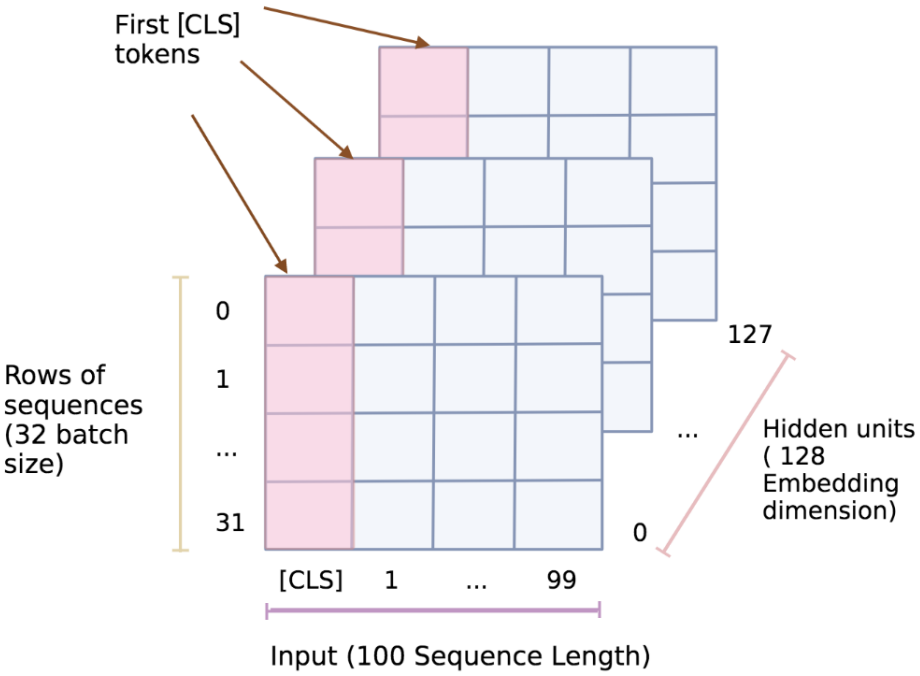

FIGURE (1.2) First CLS tokens output

- We create and pre-train a bidirectional multi-head attention encoder RecoBERT to model user behaviour sequences. We are the first to pre-train BERT-like model with two objectives, next sentence prediction, and masked language modeling prediction, to learn useful features from user's historical behaviours.
- We proposed two ways to pre-train for the next sentence classification, namely first [CLS] and averaging tokens methods.
- We discuss how the hyperparameters and parameters affect the performance on the pre-training tasks, and report the optimal hyperparameter settings for best pre-training results.
- We proposed two ways of fine-tuning our pre-trained RecoBERT model for downstream tasks, namely Freeze + fine-tune, and fine tune entire architecture. Then, we compare the gender, and age groups classification results for all fine-tuning models, and the baseline MLP model.

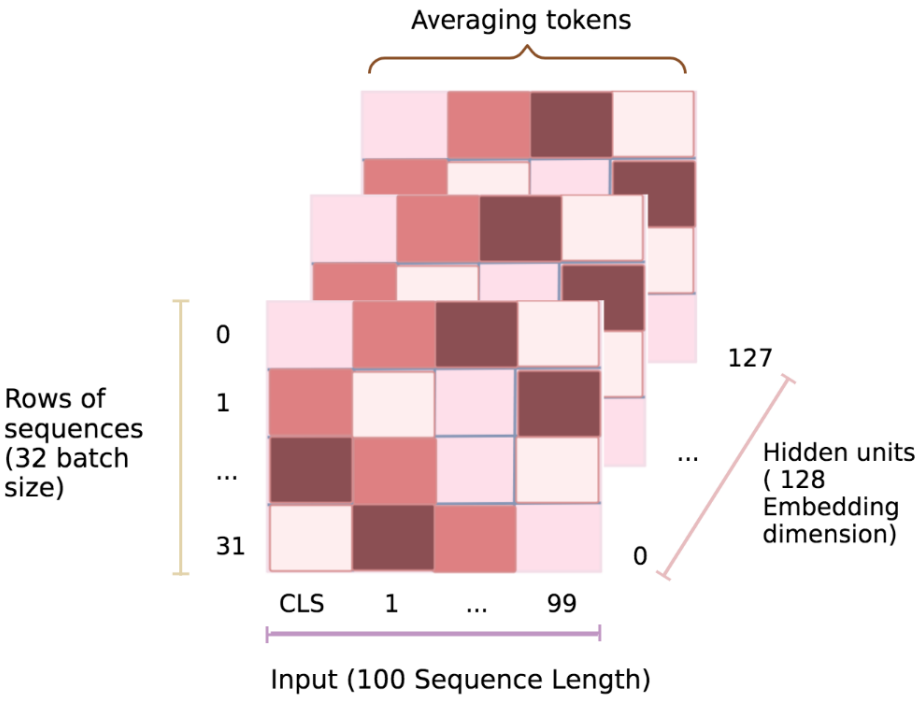

FIGURE (1.3) Average tokens output

1.3 Structure of the Thesis

In Chapter 2, we present an overview of the transformer model. Firstly, we describe the preprocessing procedures for our dataset, leading to three types of embeddings. Then, we discuss how we create positive and negative samples from these embeddings. Furthermore, we discuss the overall architecture of the masked language models, namely RecoBERT model which includes the pre-training tasks, Next Sentence classification and Masked Language modeling tasks. Finally, we discuss the performance metrics used throughout the experiments. In Chapter 3, we discuss and analyze the raw dataset used in the experiments. Chapter 4 covers both pre-training and fine-tuning experimental results, we analyze the results and advise on further improvements. Lastly, Chapter 5 summarizes all results and discuss the future work.

Chapter 2

Background

In this chapter, we cover an overview of the transformer model. The chapter is divided into Preprocessing, Embeddings, Positive and Negative Sampling, Transformers, Masked Language Models, and the performance metrics. As an overview, We will first discuss the preprocessing where we tokenize the input, then we will create multiple embeddings that act like a lookup table for the model. Furthermore, we discuss how we create positive and negative samples which are needed for Next Sentence classification task. After that, we will discuss the key architecture of transformer, in particular the RecoBERT model. Then, we further discuss the two pre-training tasks, Next Sentence Classification and Masked Language Modeling tasks. Finally, we will discuss the performance metrics used to evaluate models in our experiments.

2.1 Preprocessing

In preprocessing, our goal is to tokenize our input data, and turn our data into tokens/numbers so that our neural network is able to process and train with those data. In most cases, we will need to clean the data by making the English sentences into lower case, and create a set of vocabulary for all unique words. However, our raw dataset is already in the form of integer numbers, so we only need to create a set of vocabulary for all unique numbers. Then, we need to create special tokens

for RecoBERT model. There are four special tokens that we used in RecoBERT, and each of them has a different purpose:

Token	***Purpose***	***Token integer***
[CLS]	First classification token in a sequence	max(vocab_dict) + 1
[SEP]	Separate token to separate a sequence	max(vocab_dict) + 2
[Mask]	Replace original token with a masked token	max(vocab_dict) + 3
[PAD]	Extend the sequence to a fixed length	0

TABLE (2.1) Special Tokens

From the table 2.1 above, we add one to the maximum value from our set of vocabulary dictionary to represent the value for [CLS] token. Similarly, we add two and add three to the maximum value to represent value for [SEP] and [MASK] token respectively. For [PAD] token, We assign zero to it.

2.2 Embeddings

There are three types of Embeddings to create for training the RecoBERT: Token Embeddings, Segment Embeddings, and Position Embeddings. In Token Embeddings, we add those special tokens to the input tokens sequence. For instance, if the input tokens sequence is [2, 4, 6, 8, 10], and we decide to mask input 4 from the sequence, then the function should reconstruct the sequence into [11, 2, 13, 6, 12, 8, 10, 12]. Note that 11, 13, and 12 are [CLS], [MASK] and [SEP] token respectively. We always insert a [CLS] token to the beginning of each input tokens sequence and a [SEP] token to the end of the input sequence. The second [SEP] token will be placed in between the input sequence for positive and negative sampling purposes. We will discuss positive and negative sampling in later section.

For Segment Embeddings, we define zeros to all tokens in a sequence, and ones to all tokens in another sequence. For instance, the function should convert the input sequence [11, 2, 13, 6, 12, 8, 10, 12] into [0, 0, 0, 0, 0, 1, 1, 1], where 11, 13, and 12 are [CLS], [MASK] and [SEP] token respectively. Note that the Segment changes from zeros to ones after the first [SEP] token in an input sequence.

For Position Embeddings, we create index positions for all tokens in an input sequence. For instance, the function should convert the input sequence [11, 2, 13, 6, 12, 8, 10, 12] into [0, 1, 2, 3, 4, 5, 6, 7].

We randomly mask 20% of the input sequence, then add padding to the end of the sequence to make sure all input sequences are of the same length. For instance, if we set the sequence length to ten, given an input sequence [11, 2, 13, 6, 12, 8, 10, 12] , we will pad the sequence by adding two additional zeros at the end of the sequence: [11, 2, 13, 6, 12, 8, 10, 12, 0, 0].

Once we have created the three embeddings above, by summing up all the embeddings and normalizing them, we can then specify a desired embedding dimensions to form the embedding layers, as illustrated in Firgure 2.1. RecoBERT takes the embeddings as input to its encoder, and train the parameters of the embedding layers so that the embedding layers could cluster similar context tokens, and retain relationship between tokens. Depending on the complexity of your dataset, we typically set the embedding dimensions in a range from 50 to 500.

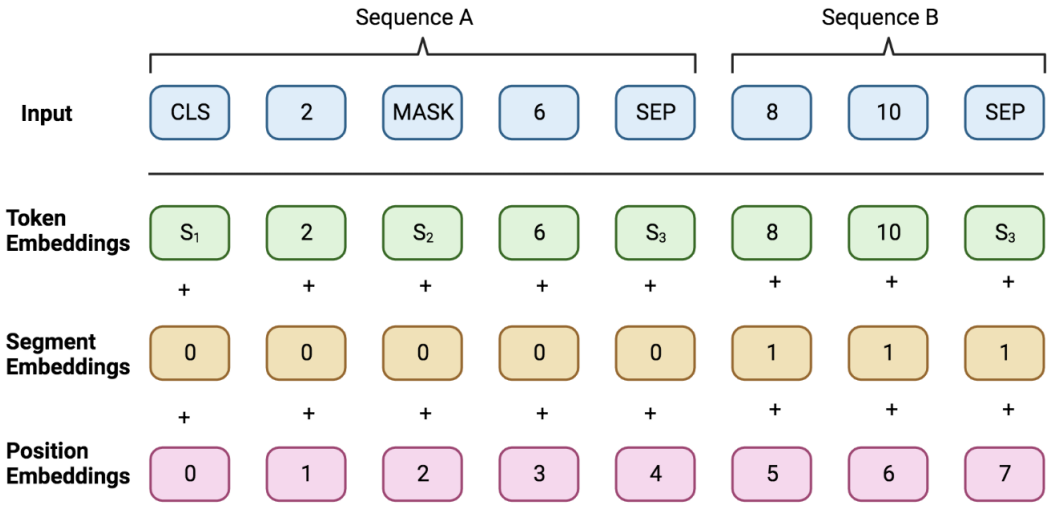

FIGURE (2.1) Embeddings

2.3 Transformer

Transformer architecture [12] was first released by Google in 2017. This architecture takes into account long-term dependencies through attention mechanism which solves the vanishing gradient problem of the Recurrent Neural Network [5] when having long sequences input. BERT which stands for "Bidirectional Encoder Representation with Transformers" is our main focus in this thesis, and BERT architecture is made up of an encoder and does not explicitly use a decoder such that the encoder finds representations and patterns from the input and attention mask. RecoBERT is a smaller version of BERT and the encoder architecture is shown in 2.2, in which the main components are the Multi-Head Attention and Position-Wise Feed-forward network.

2.3.1 Muti-Head Attention

Multi-Head Attention allows the model to learn different semantic meanings of attention. To create self-attention for the input sequence, We introduce Query (Q), Key (K), and Value (V) weight matrices into the Scaled Dot-Product Attention, as illustrated in Figure 2.3. Equation 2.1 shows the weight matrices of Query (Q), key (K), and Value (V).

$$\begin{aligned} Q &= R^{\text{batch size} \times \text{num heads} \times \text{seq length}_q \times d_k} \\ K &= R^{\text{batch size} \times \text{num heads} \times \text{seq length}_k \times d_k} \\ V &= R^{\text{batch size} \times \text{num heads} \times \text{seq length}_k \times d_v} \end{aligned} \tag{2.1}$$

where batch size represents number of rows of input sequence in batch, num heads is the number of heads in Multi-Head Attention, both seq length_q and seq length_k are the input sequence length, while d_k and d_v are the dimension of K and V respectively.

We use Scaled Dot-Product in Equation 2.2 to calculate the self-attention, first we compute the dot product of the query (Q) with the key (K), then the output is scaled by $\frac{1}{\sqrt{d_k}}$. We then apply a Softmax function to the output to give a score between 0 and 1, and finally a dot product of the output with value (V) to give us the Scaled Dot-Product output, as shown in 2.3. To calculate the Multi-Head Attention, We pass the Scaled Dot-Product output to the "Add & Normalization" layer. The architecture is as shown in Figure 2.2.

$$y = \left[softmax\left(\frac{Q \bullet V}{\sqrt{d_k}}\right) \right] \bullet V \tag{2.2}$$

2.3.2 Position-wise Feed-Forward

Position-wise Feed-Forward layer takes in the Normalized output from Multi-Head Attention and pass it to a fully-connected layer which is then activated by a Gelu (Gaussian Error Linear Unit) function, and finally pass the output to a fully connected layer as shown in Equation 2.3.

$$\begin{aligned} Gelu &= \frac{x}{2}\left(1 + erf\left(\frac{x}{\sqrt{2}}\right)\right) \\ y &= Fc\Big(Gelu\big(Fc(x)\big)\Big) \end{aligned} \tag{2.3}$$

where Fc represents the fully connected layer, erf is an error function in $Gelu$ activation function, and finally y is the encoded output and attention mask. This concludes the encoder architecture.

Since BERT main focus is building a contextual understanding from the encoder's output and attention mask and does not explicitly use a decoder. Hence, we use a shallow network as decoder in RecoBERT to pre-train for Next Sentence classification and masked Language Modeling tasks.

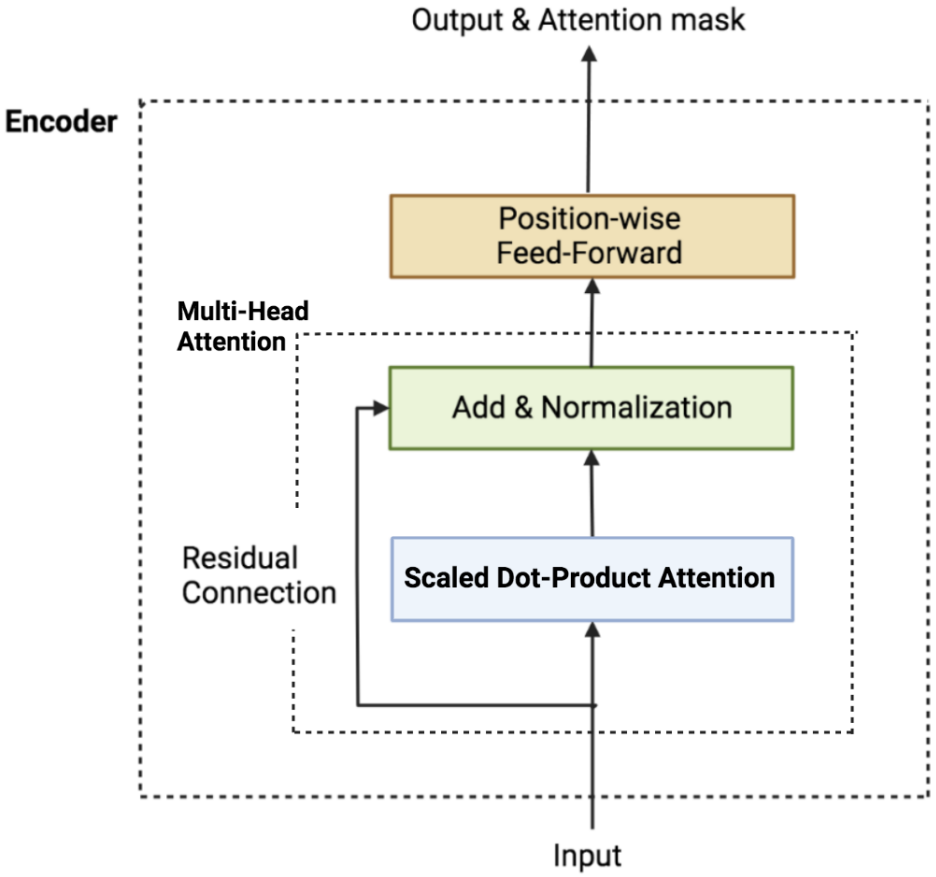

FIGURE (2.2) RecoBERT Encoder

2.4 Masked Language Models

Masked Language Models (MLM) predict the 'masked' tokens that are randomly chosen from the input sequence. This 'masking' approach is bidirectional, which is the key to RecoBERT model. MLM is known to learn context extremely well from its input in comparison to other transformer models. This is because the model attempts to predict a 'masked' item from a sequence while having access to its past tokens and future tokens. Hence, MLM is bidirectional and is forced to learn the context of the sequence to predict the 'masked' token. This contextual advantage of MLM is well suited in predicting user's clicking behaviour for recommendation tasks.

To implement masking in BERT, typically 20% of the input tokens are chosen randomly from the sequence. Then, from these chosen tokens, a probability of 80%

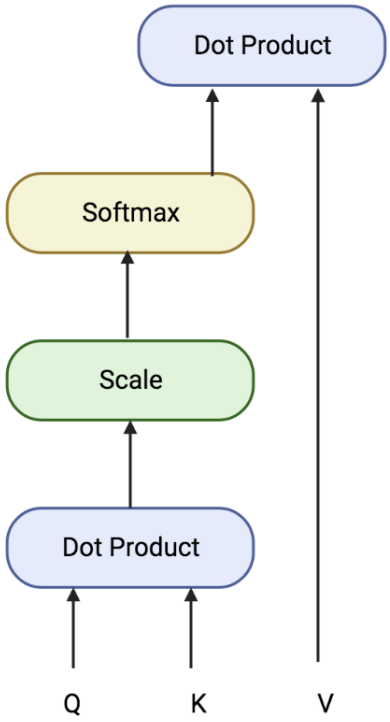

FIGURE (2.3) Scaled Dot-Product

of the tokens are replaced with special [MASK] tokens, 10% of the tokens are replaced with some incorrect random tokens, and another 10% of the tokens are left unchanged with the correct tokens. The choice of the split (80%/10% and 10%) is ideal, which is carefully chosen and tested by the BERT author [1]. BERT paper states that if 100% of the tokens are masked, then model will learn very little about its context because it will only focus on the target words in its learning objective. Instead, if the split is 80% masked tokens and 20% correct tokens, then the model will only learn that all tokens are correct in the case of no masked tokens, hence no context learning. With the same logic, if the split is 80% masked tokens and 20% incorrect tokens, then the model will learn that all all tokens are incorrect in the case of no masked tokens. Therefore, In essence, the ideal split (80%/10% and 10%) will force BERT model to learn the contextual representation of every input token. After masking, the input data is now known as augmented data which we then pass it to the embedding process, as shown as Figure 1.1.

2.4.1 Positive and Negative Samples

To pre-train RecoBERT for Next Sentence classification task. We have to create both positive and negative samples from the input sequences, as well as the labels, as shown in Figure 2.4. To do this, we separate each input sequence into two, sequence A and sequence B, and we also record the index position for both sequence A and B. Next, we randomly shuffle all sequences. During sampling, If sequence A index position equals to sequence B index position, then this sequence A and B belong to the positive samples, otherwise they belong to negative samples. For positive samples, we set the label as "True", while for negative samples, the label will be "False".

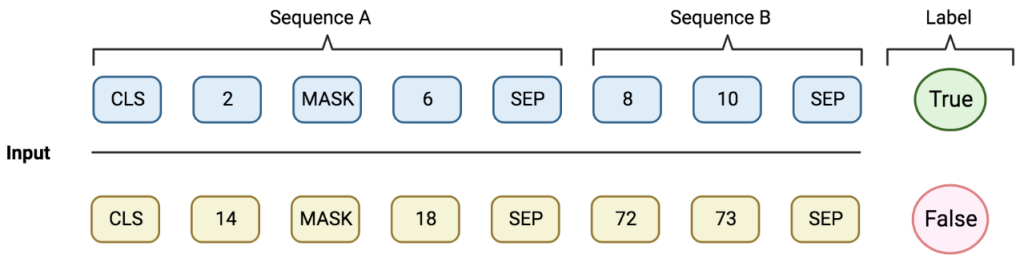

FIGURE (2.4) Positive and Negative Samples

2.4.2 RecoBERT Next Sentence classification task

RecoBERT Next Sentence classification task predicts whether sequence A consecutive sequence is sequence B or not. We introduce two classification methods: First [CLS] token method, and the averaging tokens method.

The first [CLS] token method employs only the first [CLS] tokens from each sequences, as illustrated in Figure 1.2, and then pass it to the feedforward neural network of fully connected layer with Tanh activation function, then through a linear classifier, as shown in Equation 2.4.

$$
\begin{aligned}
Tanh(x) &= \frac{\exp^{x} - \exp^{-x}}{\exp^{x} + \exp^{-x}} \\
y &= linear\_clsf\Big(Tanh\Big(Fc(input)\Big)\Big)
\end{aligned}
\tag{2.4}
$$

where $Tanh$ is the Tanh activation function that gives an output in the range from -1 to +1, $input$ represents the first [CLS] tokens from the final encoder layer, Fc a fully connected layer with size (embedding dimension x embedding dimension), whereas $linear\_clsf$ is a linear classifier with size (embedding dimension x 2), and finally we pass the output y to the softmax function to give a value between 0 and 1 for the two Next Sentence classes (isNext and NotNext). To give an example, if the model predicts 0.7 to isNext and 0.3 to NotNext, this means that high chance the sequence A consecutive sequence is sequence B. This is a well known classification method by the BERT model and the architecture is as shown in Figure 2.5.

The averaging tokens method employs the average of the tokens from each sequences, as illustrated in Figure 1.3. This means that instead of using just the first [CLS] tokens, we now use all tokens in the sequence but averaging them. We then pass the average tokens tensors as input into the same architecture in Equation 2.4 to predict a value between 0 and 1 for the two classes (isNext and NotNext). The architecture of averaging tokens method is shown in Figure 2.6.

2.4.3 RecoBERT Masked Language Modeling task

RecoBERT Masked Language Modeling (MLM) task predicts the masked items, given an augmented masked sequence. To obtain MLM predictions, we pass the encoder output to a feedforward neural network, a fully connected layer with Gelu activation function, then through a normalization layer, and finally to a linear decoder in Equation 2.5.

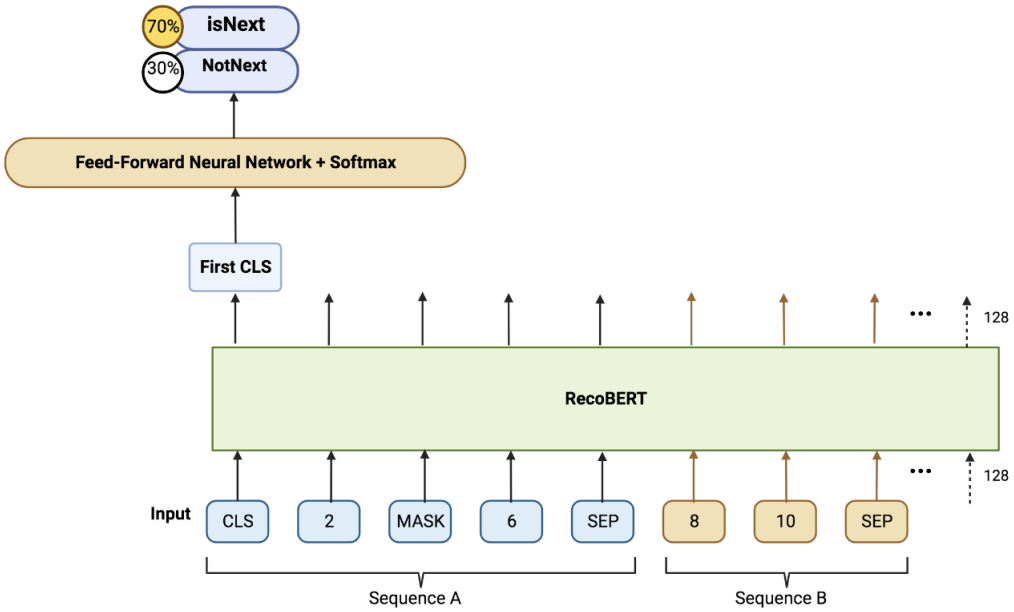

FIGURE (2.5) First CLS Next Sentence classification architecture

$$
\begin{aligned}
Gelu &= \frac{x}{2}\left(1 + erf\left(\frac{x}{\sqrt{2}}\right)\right) \\
y &= linear\_dec\left(layer\_norm\left(Gelu\left(Fc(input)\right)\right)\right)
\end{aligned}
\tag{2.5}
$$

where $Gelu$ represents the Gelu activation function, while erf is an error function, $input$ is the masked token positions from the final encoder, Fc represents a fully connected layer with size (embedding dimension x embedding dimension), $layer\_norm$ is a normalization layer, and finally $linear\_dec$ is the linear decoder with size (embedding dimension x vocab classes). Vocab classes is the unique vocabulary dictionary we computed for our dataset. The architecture is as shown in Figure 2.7. Observe from Figure 2.7, the vocab classes are ordered numbers range from 1 to 80, but in our real experiment the total of vocab classes is way larger than 80.

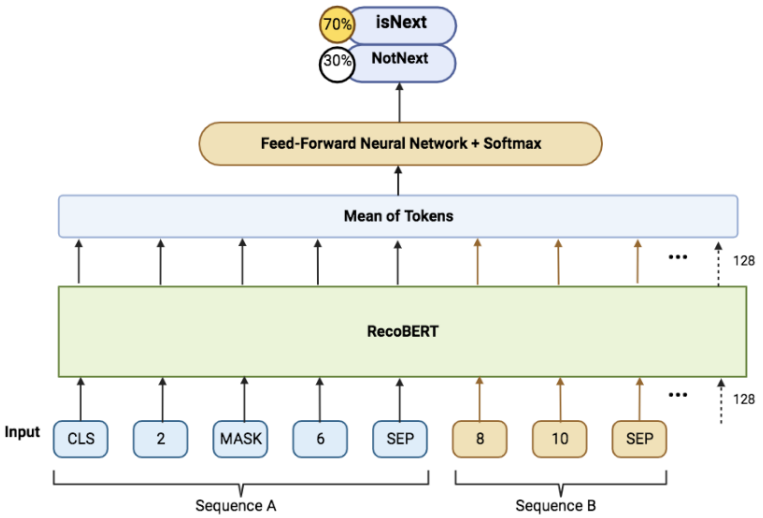

FIGURE (2.6) Avg tokens Next Sentence classification architecture

2.4.4 Loss and Optimization

Loss functions are used to optimize the model during training, and the goal is minimizing the loss function to get a better model. We use cross-entropy loss function and Adam optimizer throughout the experiments to train our model. Cross-entropy loss [3] is a combination of both softmax and negative log-likelihood:

$$
\begin{aligned}
p_i &= \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} \\
L_{CE} &= -\sum_{i=1}^{n} t_i \, log(p_i), \text{ for } n \text{ classes}
\end{aligned}
\tag{2.6}
$$

where p_i is the Softmax probability for the $i^{th} class$, while t_i is the truth label. The log in Equation 2.6 is calculated to the base 2. Softmax converts our logits into probabilities, such that each logit is converted to a probability between 0 and 1, and eventually all output probabilities should sum to 1. The cross-entropy then takes the probabilities to measure the distance from the truth labels. To give an example, if we have a binary classification task with the aim to predict "True" or "False", then the truth labels t_i are 0 and 1 to represent "True" and "False" respectively. If the predicted

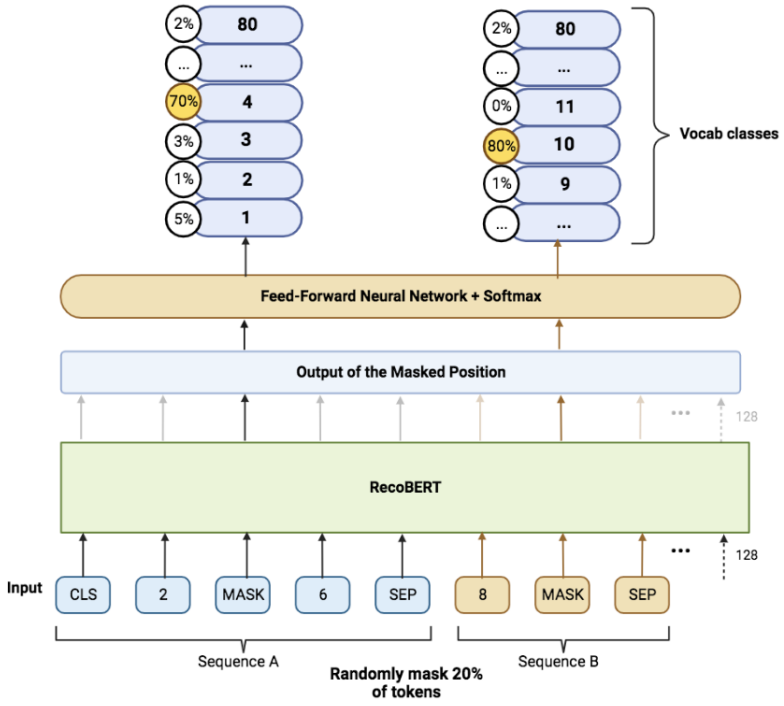

FIGURE (2.7) Masked Language Modeling prediction architecture

logit p_0 is close to 0, while the truth label t_0 is 0, then the distance between the predicted logit and the truth label is small, hence the cross-entropy loss calculated will be low. In contrast, If the predicted logit p_1 is close to 0, while the truth label t_0 is 1, then the cross-entropy loss will be high due to the long distance between the predicted logit and the truth label.

Adam[6] which stands for Adaptive Moment Estimation is an optimization technique for gradient descent. The optimizer combines Momentum algorithm [8] with Root Mean Square Propagation algorithm [7] to give an optimized gradient descent that outperforms most other optimizer in terms of training costs and performance, according to "An overview of gradient descent optimization algorithms" article [10]. Hence, we make Adam optimizer as the default method for updating weights iteratively. The model will then adjust to the weights accordingly to minimize the cross-entropy loss.

2.5 Performance Metrics

There are 2 performance metrics used throughout the experiments, which are Accuracy and F1 Score. Accuracy evaluates only the True Positives and True Negatives as shown in Equation 2.8. For clarification on True/False Positives and Negatives. We take the gender "Male" or "Female" classification as an example:

True Positives: User is a "Male" and model classified it as "Male".

True Negatives: User is a "Female" and model classified it as "Female".

False Positives: User is a "Male" and model classified it as "Female".

False Negatives: User is a "Female" and model classified it as "Male".

$$\text{Accuracy} = \frac{\text{True Positives + True Negatives}}{\text{All Samples}} \tag{2.7}$$

F1 Score takes into account False Positives and False Negatives that penalize wrongly classified labels. Hence, this metric is useful to evaluate imbalance classes by penalizing all wrongly classified labels. F1 Score Equation is as shown below:

$$\text{F1 Score} = \frac{\text{2 x True Positives}}{\text{(2 x True Positives)} + \text{(False Positives + False Negatives)}} \tag{2.8}$$

Chapter 3

Datasets

Our recommender system experiments seek to learn user's historical click items behaviour sequences. Hence, we will use Tencent[1] dataset, that can be downloaded from this link. The raw dataset format is of the following: [User ID,, QQ browser[2] clicking items sequence,, Kandian[3] clicking items sequence,, Gender, Age, Life status, Education status, Profession]. We could easily split each section by the double commas (,,) and concatenate each section together into four different lists. The lists are: User ID list, QQ sequences list, Kandian sequences list, and profiles list. The full dataset has over a million of users, it would be computationally expensive and time consuming to pre-train with the full dataset, hence we decided to pick a total of two thousand users for our experiments. Below is a breakdown for all four lists:

3.1 User ID list

The list of User ID is made up of non-repeating positive integer numbers. For instance, the list [560, 235, 101, 56, 78] represents five unique user id, and each user id corresponds to a new user.

[1]https://www.tencent.com
[2]https://browser.qq.com
[3]https://kandian.qq.com

3.2 QQ browser clicking items sequences list

QQ browser clicking items are made up of a variety of online videos, news, and short articles. For instance a randomly picked user may have a QQ clicking items sequence of [45, 87, 66], where the item number could represent a click to a short business article or an online sports video, etc. Note that there are three numbers in this QQ clicking items sequence, so we can say the QQ sequence length for this user is three. We plot a histogram to get the statistics of the QQ sequence length for all two thousand users:

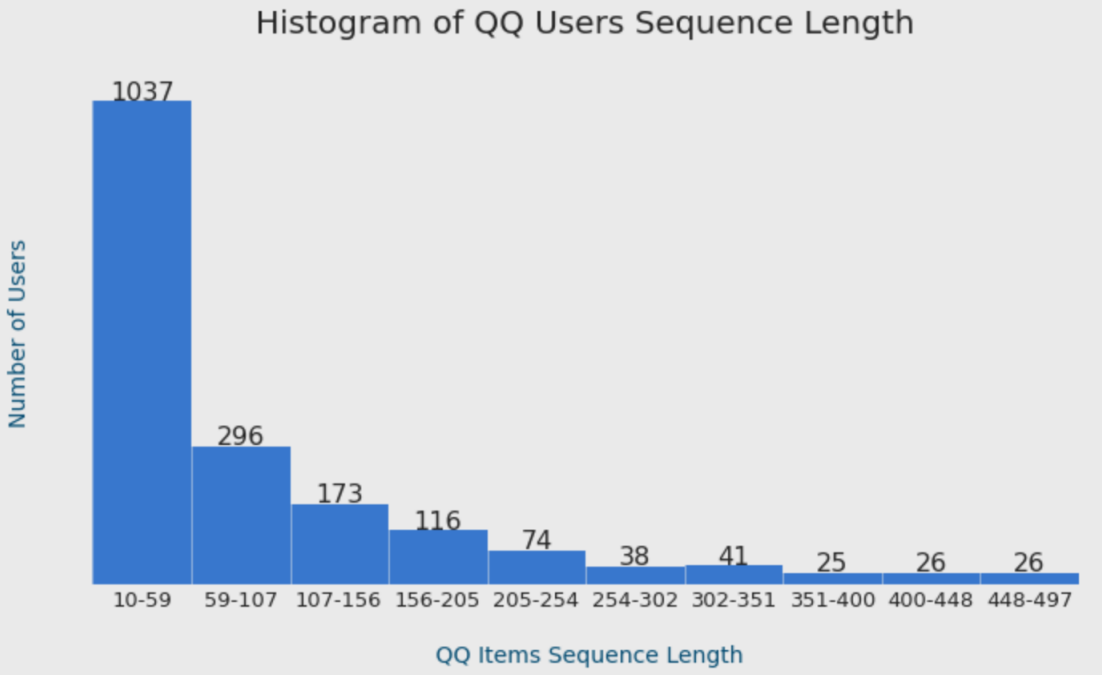

FIGURE (3.1) QQ Users Sequence Length Histogram

From Figure 3.1 above, over 66% (1037+296) of the users have QQ items sequence length between 10 and 107, and the minimum and maximum of sequence length are 10 and 3573 respectively. Note that the histogram is truncated on the right hand side because the number of users decreases exponentially as sequence length increases to the right. For instance, there is only one user with a sequence length of 3573. In our experiments, we fix the sequence length to 100, as a representation of QQ users clicking items sequence for each user. If a user QQ items sequence length is less than

100, then we add [PAD] tokens to the end of the sequence. We will utilize the QQ items sequences in the pre-training experiments.

3.3 Kandian clicking items sequences list

Kandian is a different platform from QQ with the same group of users, and the clicking items are also made up of online videos, news, and short articles. We plot a histogram in Figure 3.2 to get the statistics of the Kandian sequence length for all two thousand users.

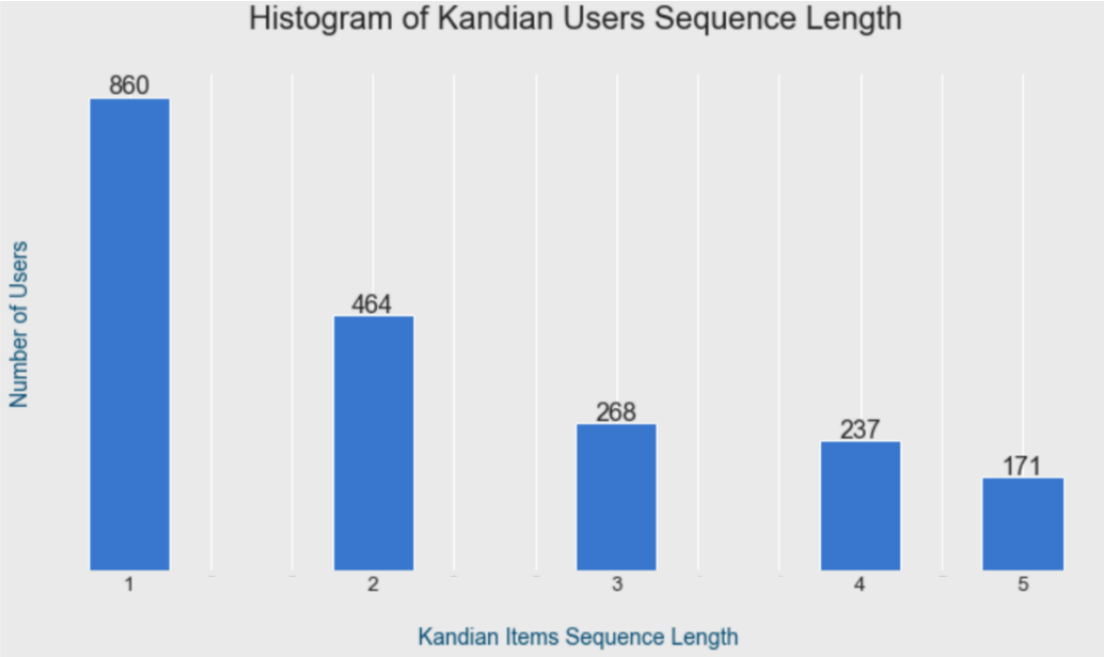

FIGURE (3.2) Kandian Users Sequence Length Histogram

From Figure 3.2 above, almost 80% (860+464+268) of Kandian sequence length is less than or equal to 3. Also, we find that the sequence length is so much shorter than the QQ sequence length with the minimum and maximum of Kandian sequence length being 1 and 5 respectively.

3.4 Profiles list

Each user profile contains five profile information: gender, age, life status, education status, and profession. Each profile information has different label as shown in Table 3.1. For example, gender will have 2 labels, 1 or 2 to represent "male" or "female". Then, for each user we batch together the profile information. For instance, a randomly picked user has a profile of [2, 4, 2, 3, 5], this corresponds to [female, 31-40, married, high school education, doctor].

Profile	1	2	3	4	5	6
Gender	Male	Female				
Age	≤ 10	11-20	21-30	31-40	41-50	51-60
Life Status	Single	Married	Pregnant	Child-rearing	Divorced	Widowed
Edu status	Primary	Middle	High	College	Undergrad	Postgrad
Profession	Investment Advisor	Tester	Copywriter	Accountant	Doctor	Chef

TABLE (3.1) Profile table

We utilize gender profile as the target dataset in our fine-tuning experiments, but we will only consider gender and age profile information in the experiments because there are significant missing data for Life status, Education status, and Profession labels. We plot histogram for both gender and age groups profile in Figure 3.3 and 3.4 respectively.

In Figure 3.4, we notice the age group (11-20 years), (21-30 years), and (31-40 years) dominate the age profile dataset. Furthermore, in Figure 3.3 there are more than 72%(1454) users are male, and only 28%(542) users are female. The imbalance classes could pose a problem in fine-tuning experiments, as the model will be biased towards the majority class. To overcome this issue, we use sampling technique to balance up the minority class. First, we find all indices in QQ items sequences that belong to female labels, then we pick randomly from these sequences and resample the QQ items sequences by shuffling items in each sequence. This way, We can add more female QQ items sequences and female labels into the dataset until both labelled classes are equal in number (1454 male QQ items sequences and 1454 female QQ items sequences).

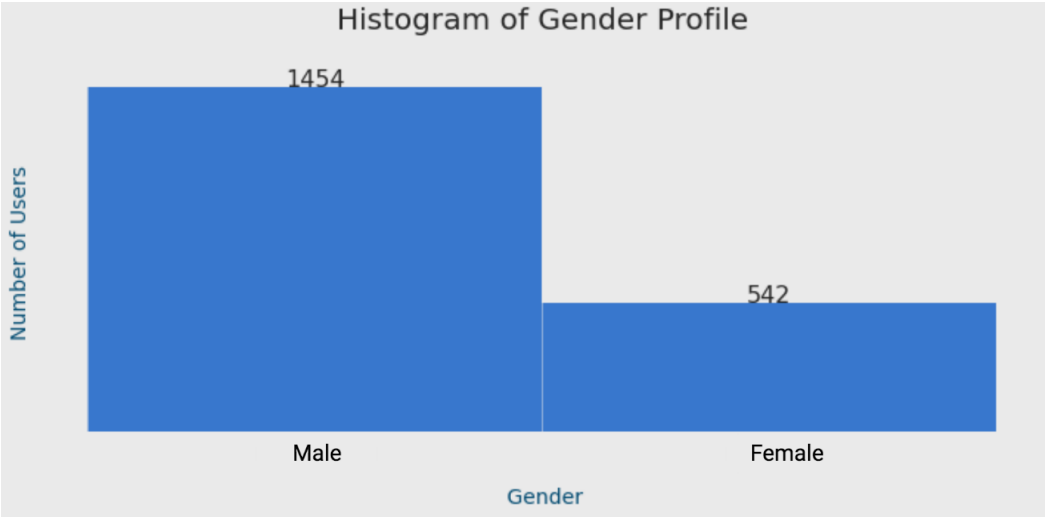

FIGURE (3.3) Gender Profile Histogram

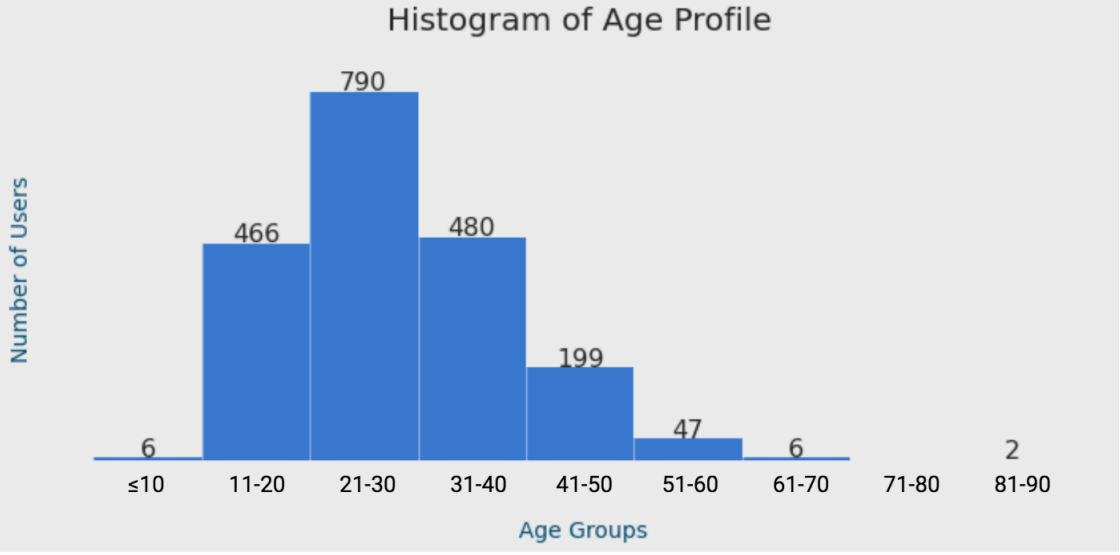

FIGURE (3.4) Age Groups Profile Histogram

Chapter 4

Experiments

In our experiments, we attempt to answer the following research questions: **Research Question 1** – How does the RecoBERT model perform when we pre–train it for next sentence classification and masked language modeling tasks using the sequential recommendation datasets?

Research Question 2–Does transfer learning via pre-trained RecoBERT model improve downstream tasks performance?

Research Question 3 – Are there any interesting insights we can draw from the RecoBERT experimental results, and how can we further improve the model?

4.1 Pre-training

4.1.1 Pre-processing

We conduct pre-training experiments using two thousand users QQ browser clicking items sequences that was collected from Tencent. We split the two thousand sequences into training (80%) and validation (20%). Each sequence varies in sequence length from a minimum of 10 to a maximum of 3000. According to QQ users sequence length histogram in Figure 3.1, most users have clicking items sequence length within 100, so we decided to set a sequence length of 100 for all our experiments. Each sequence is split into two, [tokens_A and tokens_B], tokens_B has the most recent 5 items from the sequence, while tokens_A consists of the most recent

92 items from the sequence excluding tokens from tokens_B. Recall if both tokens_A and tokens_B have the same index number, then they belong to one QQ items sequence, hence this is considered a positive sample. We can create negative samples by randomly shuffling tokens_A and tokens_B, resulting in different index number between tokens_A and tokens_B. In our pre-training experiments, We generate 5000 positive samples with "True" labels, and another 5000 negative samples with "False" labels. For each sample, we insert a special [CLS] token at the beginning of the sequence, and a special [SEP] token is placed between tokens_A and tokens_B, as well as at the end of each sequence. If the sequence length of the sequence is less than 100, we will pad the sequence with [PAD] tokens till we reach a sequence length of 100. See a simple example below:

Original Sequence: [4, 5, 4, 7, 9, 3, 10, 50, 60, 70, 12, 80, 100]

Tokenized Sequence: [[CLS], 4, 5, 4, 7, 9, 3, 10, 50, [SEP], 60, 70, 12, 80, 100, [SEP],[PAD], ... , [PAD]]

Kindly refer to Table 2.1 for the special token representations. We then randomly pick some tokens from the tokenized sequence by setting a condition: $Minimum(30, 20\%$ **of tokens**. This condition ensures that we pick a minimum of 20% tokens and a maximum of 30 tokens for later masking. For each chosen token, 80% probability the token is replaced with a special [MASK] token, 10% of the time the token is replaced with a randomly picked incorrect token, and another 10% of the time the token is left unchanged. We record the replaced token and its index position in Token Embeddings, and Position Embeddings respectively. Furthermore, We create Segment Embeddings from our tokenized sequence by assigning [0] from the beginning of the sequence, which starts from a [CLS] token all the way to the first [SEP] token, then assign [1] to the rest of the tokens until the second [SEP] token in the sequence. We call a masked tokenized sequence as augmented sequence.

At the end of pre-processing, We will batch the data for each QQ items sequence into five sections, [A]– augmented sequence, [B] – segment ids, [C] – masked tokens, [D] – masked token positions, [E] – "True or "False" label as shown in batch dummy

example below:

Batch dummy example: [[A], [B], [C], [D], [E]]

Positive sequence sample: [4, 5, 4, 7, 9, 3, 10, 50, 60, 70, 12, 80, 100]

Pre-processed batch:

[[[CLS], 4, 5, [MASK], 7, 9, 3, 10, 50, [SEP], 60, 70, 12, [Mask], 100, [SEP], [PAD], ..., [PAD]],[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, ..., 0], [4, 80], [3, 13], [True]]

4.1.2 Embeddings Architecture

We have token embeddings (103866 x 128), segment embeddings (2 x 128), and position embeddings (100 x 128), which corresponds to (vocab size x embedding dimension), (number of segment x embedding dimension), and (sequence length x embedding dimension) respectively. We sum up all three embeddings and then normalizing it to obtain the final embedding layers, as shown in Figure 2.1. Note that we set 128 as our default embedding dimension. We find that high embedding dimension can be computational expensive and does not contribute to better performance by the model, so we consider 128 as an optimal embedding dimension.

4.1.3 RecoBERT Architecture and Results

RecoBERT is essentially the encoder part of the transformer, which is made up of Multi-head Attention. To compute the Multi-head Attention, we set the Query, Key and Value with size (32 x 4 x 100 x 64), which corresponds to (batch size x number of heads x sequence length x dimension of K/V) as shown in Equation 2.1, and pass it to the Scaled Dot-Product function 2.2 to get the context output (32 x 100 x 256) which corresponds to (batch size x sequence length x number of heads*dimension of V), we then pass the output to a linear model (256 x 128), where 128 is the embedding dimension, then we add this output to the Query, and normalizing it to give a Multi-Head Attention output (32 x 100 x 128) which corresponds to (batch size x sequence length x embedding dimension). We find that the optimal number of heads is 4,

which is a low number and this makes perfect sense, since we are dealing with QQ clicking items sequences with short sequence length ($\leqslant$100) so the model does not need a high number of heads or a multiple "representation subspaces" to understand the sequences.

The output from Multi-Head Attention goes into a Position-Wise Feed Forward network, by first going through a linear network (128 x 128*4), where 128 and 128*4 are the embedding dimension and feed forward dimension respectively, note that the feed forward dimension is 4 times as large as the embedding dimension on default. Then the output is activated by a *Gelu* function, follow by another linear network (128*4 x 128) to give a Position-wise Feed Forward output (32 x 100 x 128) which represents (batch size x sequence length x embedding dimension). This concludes the encoder part, as illustrated in Figure 2.2.

The number of encoder layers or number of transformer blocks is another important parameter to be determined. The Huggingface BERT [1] model uses 12 encoder layers, however we only use 1 layer in our RecoBERT model, which is best for our recommendation tasks. This is reasonable because BERT requires high number of encoder layer to train on a huge amount of corpus from the Toronto book and Wikipedia[1], which can be fine-tuned for a wide range of tasks. On the other hand, our RecoBERT is trained on QQ users clicking items corpus, and can only be fine-tuned for users clicking behaviours and users profile prediction in the recommender system domain. Therefore RecoBERT performs best with 1 encoder layer, and increasing the encoder layer not only would lengthen the training time, but performance degrade.

Dataloader passes a batch size of 32 samples into the model, and the encoder output is used to calculate the total cross-entropy loss which is a summation cross entropy losses from both next Sentence classification task and the Masked Language Modeling task:

[1]https://www.wikipedia.org

$$Total\ Loss_{CE} = Next\ Loss_{CE} + Mask\ Loss_{CE} \tag{4.1}$$

We use hyperparameter settings as shown in Table 4.1, and pre-train the RecoBERT model for 1000 epochs, using Adam optimizer with a 0.00001 learn rate. The total training time is 11 hours on a Tesla V100 GPU. Note that RecoBERT is comparatively smaller than the real BERT model in terms of parameters, and training time.

We record the Training and Validation Loss for each epoch until convergence. From Figure 4.1, Validation loss is at minimum around 0.411 in the 100^{th} epoch, then the validation loss increases steadily. The training loss also exhibits a dramatic decrease in loss from 1^{st} epoch till the 100^{th} epoch, thereafter, the training loss decreases steadily as epoch increases. This shows that after the 100^{th} epoch, the model begins to overfit the dataset, resulting in higher validation loss and hence worse prediction. To analyze the performance for both next sentence prediction and masked language modeling tasks, we decided to run the pre-training independently for each task.

Model	**Encoder Layers**	**Embedding Dimension**	**Multi Heads**	**Parameters**	**Processing**	**Training Time**
BERT	12	768	12	110M	4 TPUs	4 days
RecoBERT	1	128	4	21M	Tesla V100	11 hrs

TABLE (4.1) RecoBERT and BERT parameters

4.1.4 RecoBERT Next Sentence classification task results

RecoBERT predicts "True" or "False" whether sequence A consecutive sequence is sequence B. A simple classification method or baseline method is to make use of the first [CLS] tokens output from the batch sequences of last encoder layer, as shown in Figure 1.2.

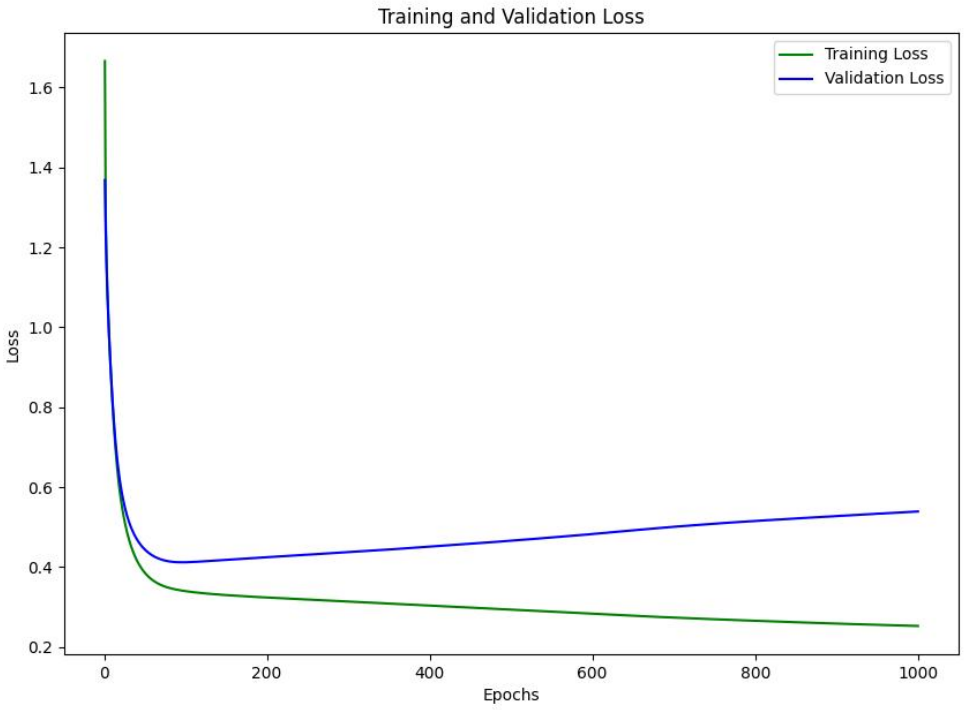

FIGURE (4.1) RecoBERT Training and Validation Loss

We take the first [CLS] tokens output (32 x 128) from the RecoBERT encoder output (32 x 100 x 128) which corresponds to (batch size x sequence length x embedding dimension), and then pass it to a fully connected layer with size (128 x 128), which is then activated with a Tanh function. Finally, we pass it to a linear classifier with size (128 x 2) and through softmax function to predict for the 2 classes, "True" or "False". To improve the classification accuracy, instead of using the first [CLS] tokens output for prediction, we take the average of tokens from the batch sequences for prediction, as shown in Figure 1.3. We pass the averaging tokens output (32 x 128) to the same architecture as discussed above to predict for the 2 classes.

we run training on both methods for 20 epochs, using Adam optimizer with a 0.00001 learn rate. Note that in this test we only consider the cross-entropy loss from the Next sentence classification task. The validation loss after convergence and the test accuracy are shown in Table 4.2. The average tokens method performs

slightly better than the first CLS method, with 52% test accuracy and 50% test accuracy respectively. If we look at the validation loss plot in Figure 4.2, we can see that although in general the average tokens method has a lower validation loss with a mean of 0.0688 from 1^{st} epoch to 15^{th} epoch, but as the epoch increases further, the validation loss can be unstable. This contrasts with the first CLS method in which validation loss decreases slowly and converges to 0.0693. Overall, the test accuracy is low and this is probably due to the simple architecture of RecoBERT, while the classification task on QQ users sequences may be too challenging for the model since a lot of vocabs in the sequences are non-repeating vocabs.

	Validation Loss	Accuracy
First CLS method	0.0693	50%
Average method	0.0688	52%

TABLE (4.2) Next Sentence classification results

4.1.5 RecoBERT Masked Language Modeling task results

RecoBERT MLM task predicts the masked items given the masked sequence. We gather RecoBERT output values that correspond to the masked token positions from the augmented sequence to produce a masked output of size (32 x mask_pred x 128). Note that mask_pred is the number of masked tokens we want to predict. Then, we pass it to a fully connected linear layer (128 x 128) , which is then activated by Gelu function. Finally, we pass it to final linear decoder layer (128 x 103862) for masked token prediction, 103862 represents the vocab classes we tend to predict for the masked tokens.

Similar to Next Sentence task, we run training for 20 epochs using Adam optimizer with a 0.00001 learn rate. In this test, we only consider the cross-entropy loss from the MLM task. Our experimental results show that the MLM task performance

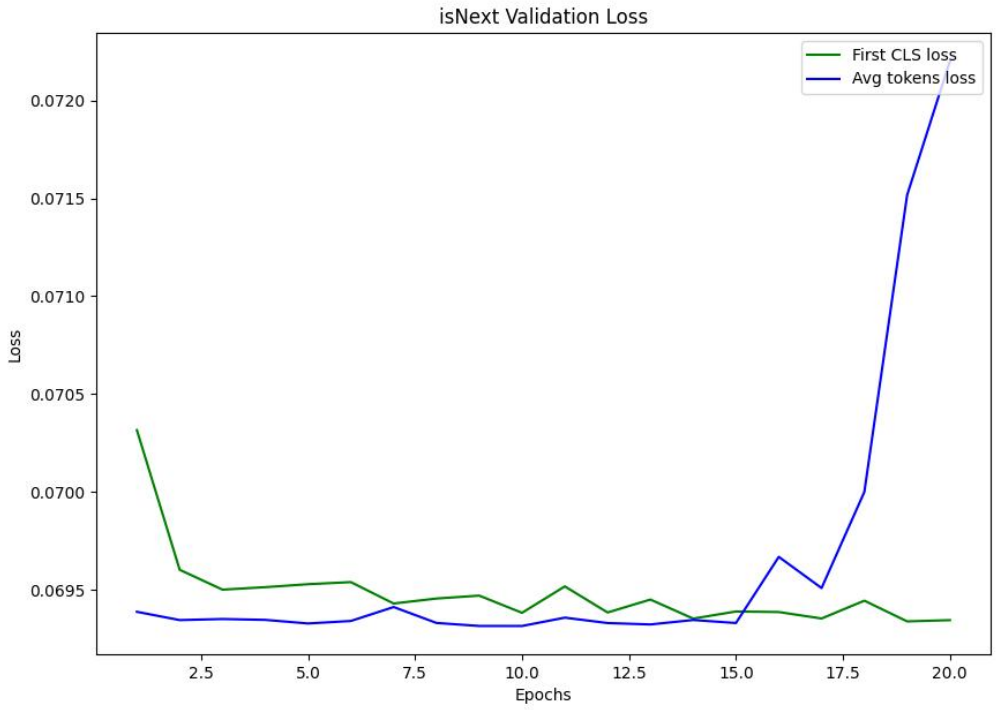

FIGURE (4.2) Next Sentence Validation Loss

is very sensitive to the maximum masked tokens hyperparameter. Recall from preprocessing section, we pick the minimum of 30 tokens or 20% tokens from the sequence for masking: $Minimum(30, 20\% \text{ of tokens})$. Varying the condition above by increasing the masked tokens can lead to an improvement of masked tokens prediction. Table 4.3 shows that as we increase the minimum of masked tokens from 10 to 40, the validation loss after convergence decreases from 0.871 to 0.337 as shown in Figure 4.3, and the test accuracy increases from 26% to 71%. It is also obvious that masking more tokens means the training time will be longer too. The training time increases from 5.5 minutes to 13.11 minutes for 40 masked tokens.

The superior results by the $Minimum(40, 20\% \text{ of tokens})$ masking setting is not surprising because most of our sequences have a sequence length $\leqslant$100. Hence, if we set $Minimum(10, 20\% \text{ of tokens})$ masking condition for an augmented sequence of length 100, we would mask only 10% (10) tokens from the sequence, the model will find it quite difficult to learn the context from only 10% portion of tokens being

masked. Hence, it makes sense to increase the minimum tokens to 40 by setting $Minimum(40, 20\% \text{ of tokens})$ masking condition. This ensures that no matter we have a short input sequence ($\leqslant$20) or a long input sequence ($\leqslant$100), we are almost surely to mask 20% of tokens by the (80%/10% and 10%) masking rule as discussed in Chapter 2: Masked Language Models section. Overall, a careful choice of masked tokens can have a great impact in the performance of RecoBERT masked language modeling task.

RecoBERT	Training Loss	Validation Loss	Accuracy	Training Time
10 tokens	0.652	0.871	26%	5.50 mins
20 tokens	0.532	0.675	43%	8.44 mins
30 tokens	0.350	0.437	62%	13.22 mins
40 tokens	0.264	0.337	71%	13.11 mins

TABLE (4.3) Masked prediction results

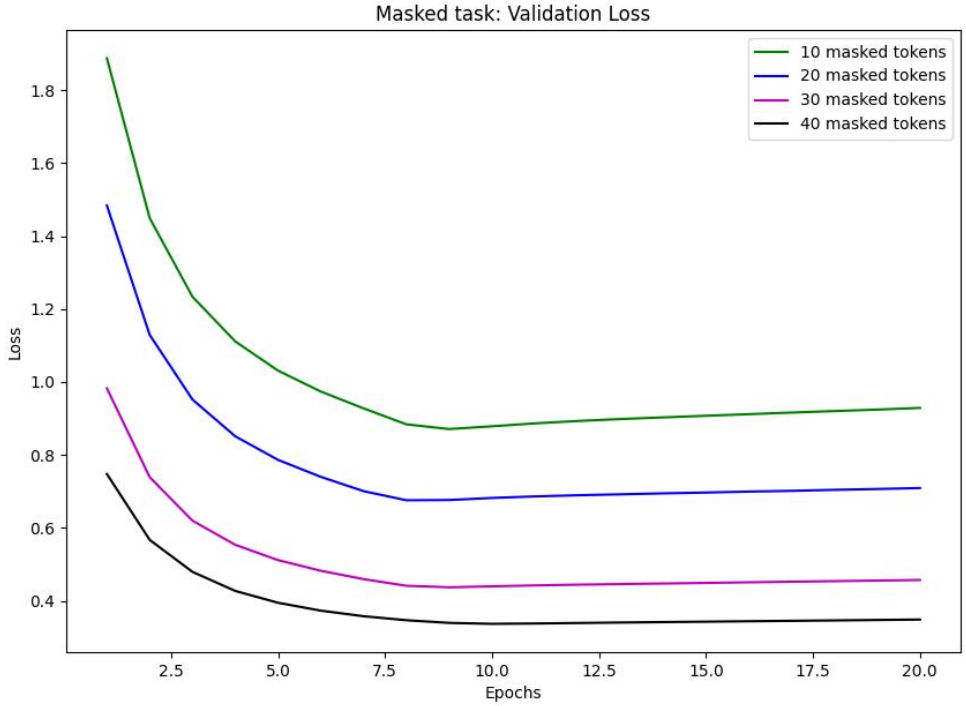

FIGURE (4.3) Masked Validation Loss

4.2 Fine-Tuning

After we have pre-trained our RecoBERT model, next step is fine-tuning the pre-trained model by adding a few neural network layers and freezing the actual layers of RecoBERT architecture. Our fine-tuning tasks involve predicting the user's gender and age, given the QQ user clicking items sequence. Hence, in this experiments, we will utilize the QQ users clicking items sequences and the target profiles as our fine-tuning training dataset.

There are 2 fine-tuning approaches in our experiments, first approach is to train the entire architecture including the pre-trained RecoBERT model. This means that error is back-propagated through the entire network and the pre-trained weights are also updated on our fine-tuning dataset. Another approach is freezing the entire pre-trained RecoBERT architecture, and attach a few neural network layers for training, in this case, only the weights of the newly attached neural network layers are updated.

4.2.1 Pre-processing

We conduct fine-tuning experiments using the same two thousand QQ user clicking item sequences and gender & age profiles as the labelled dataset, We split the two thousand dataset into training (70%) Validation(15%) and test (15%) sets. To achieve a sequence length of 100, We consider the most recent 98 QQ items from each sequence, and assign a special [CLS] token at the beginning of the sequence, and a [SEP] token at the end of the sequence. If the tokenized sequence is less than a sequence length of 100, we will assign [PAD] tokens at the end of the sequence till we reach sequence length of 100. Note that since we do not need positive/negative samples for finetuning, each sequence is not split into tokens_A and tokens_B :

Original Sequence: [12, 26, 88, 2, 25, 8, 44, 36] Tokenized Sequence: [[CLS], 12, 26, 88, 2, 25, 8, 44, 36, [SEP],[PAD],..., [PAD]]

To create segment embeddings for the sequence, we assign [1] to all numbered tokens and [0] to all [PAD] tokens. There is no masking required in this experiments, so no [Mask] tokens attached to our batch. At the end of pre-processing, We will batch the data for each QQ items sequence into three sections, [A]– tokenized sequence, [B] – segment ids, and [C] – Gender/Age labels:

Batch dummy example: [[A], [B], [C]]

QQ items sequence: [12, 26, 88, 2, 25, 8, 44, 36]

Pre-processed batch:
[[[CLS], 12, 26, 88, 2, 25, 8, 44, 36, [SEP], [PAD], ..., [PAD]], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, ...0], [Male]]

4.2.2 RecoBERT Transfer Learning Architecture

In transfer learning, We import pre-trained RecoBERT model using the same hyper-parameter settings (Embedding dimensions=128, number of Heads= 4, number of Encoder layer= 1, K/V dimensions= 64) from pre-training.

We introduce two methods of transfer learning: Freeze + Fine-tune method, and Fine-tune All method. In Freeze + Fine-tune method, We freeze the RecoBERT parameter updates by setting "False" to param.requires_grad on PyTorch. We then pass the tokenized input sequence and segment embeddings into the pre-trained RecoBERT that gives an output (128 x 103866), where 103866 is the vocabs size. Then we pass the output to a new linear network (103866 x 512) with Relu activation function, follow by a 10% probability dropout layer to avoid overfitting. Finally, we pass the output to the last linear classifier (512 x 2), and then a Softmax function to give us the "Male" or "Female" classification results. Above process can easily be applied to Age classification task, by changing the last linear classifier (512 x 2) into a linear classifier of size (512 x 8) because there are 8 age groups to classify. In Fine-tune All method, we unfreeze RecoBERT parameter updates by setting "True" to param.requires_grad, and follow the same architecture to train our model.

4.2.3 Multi-Layer Perceptron Baseline

We train a Multi-Layer Perceptron network as our baseline to compare the performance against the RecoBERT Transfer learning network. In MLP, we use a very shallow network by passing the tokenized input sequence into a linear layer (100 x 50), and then feed the output to Relu activation function and a 10% dropout layer, and finally to a linear classifier (50 x 2)/(50 x 8) with a Softmax function to give the final classification results.

4.2.4 Gender Classification Results

We run fine-tuning training for 300 epochs using Adam optimizer with a 0.001 learn rate. The gender prediction results are shown in Table 4.4. To evaluate the performance of our model, We use the accuracy and macro average F1 score metrics. Macro average f1 score is suitable for equal labelled classes as it is calculated using the mean of all the per-class F1 scores.

Gender	Macro Avg F1 Score	Accuracy	Training Time
MLP	0.37	51%	2.01 mins
Freeze + Fine-Tune	0.55	54%	9.88 mins
Fine-Tune All	0.49	49%	14.23 mins

TABLE (4.4) Gender prediction results

In gender classification task, MLP prediction tends to be biased to "Female" Class even though we have sorted the imbalance classes issue in our pre-processing. MLP predicted only 9 males, but 342 females in a testing dataset that contains 172 truth males and 179 truth females. Hence, its macro average F1 score is the lowest at 0.37 due to bad prediction or low F1 score for male class. Our baseline accuracy is 50% since we have an equal amount of Male and Female dataset.

The Freeze + Fine-Tune model performs the best, with highest average F1 score (0.55), and highest accuracy (54%). This proves that transfer learning via a pre-trained RecoBERT model is in fact helpful for the downstream task in recommender

system, condition upon good pre-training performance of RecoBERT model. However, the training time 9.88 minutes is almost 5 times longer than the baseline MLP model. To further improve the performance of the downstream task, instead of a shallow network attached to the last layer of our pre-trained model, we would need a more complex fine-tuning architecture, for example dilated convolutional layers [14] to predict for the downstream task.

Fine-Tune All model updates weights for the entire architecture, including the pre-trained model. We see that both its average F1 score (0.49) and accuracy (49%) perform worse than the results of Freeze + Fine-tune model. Hence, we argue that updating weights for the pre-trained model may not be necessary and doing so may overwrite useful features from the pre-trained model, resulting in worse performance due to the needs to relearn the features. The training time (14.23mins) is the longest since we need to update weights for 21M parameters in the pre-trained model.

4.2.5 Age Groups Classification Results

For Age groups classification task, We run training for 300 epochs using Adam optimizer with a 0.001 learn rate. The Age groups prediction results are shown in Table 4.5. Age groups prediction is a more challenging task than previous gender prediction because now the model has to classify for 8 age groups. The imbalance classes issue is severe in this task as shown in Figure 3.4, so we are unable to resample the minority classes using the same technique used for gender task. To evaluate the performance of our model, We now use the weighted average F1 score because this metric takes into account the contribution of each class as weighted by the number of examples of that given class.

MLP prediction tends to be biased to age group 3 (21-30 years), which is a majority class with 39% proportion of the dataset, this results in high accuracy score (36%

Age Groups	Weighted Avg F1 Score	Accuracy	Training Time
MLP	0.21	36%	4.12 mins
Freeze + Fine-Tune	0.29	29%	9.90 mins
Fine-Tune All	0.26	27%	9.76 mins

TABLE (4.5) Age Groups prediction results

) for MLP. Accuracy does not reflect the true performance of a model here, therefore we should consider the weighted average F1 score metric. MLP has the lowest weighted average score at 0.21, as its F1 score weight is dominated by the biased class.

The Freeze + Fine-Tune model performs the best, with the highest weighted average score at 0.29, with 9.90 minutes training time. Fine-Tune All model performs slightly worse than the Freeze + Fine-Tune model as the weights update on the entire architecture may not be useful at all. Both transfer learning models training time are within 10 minutes.

Overall, from the results above we prove that transfer learning via a pre-trained RecoBERT model can be helpful for downstream tasks, as both transfer learning models in general perform better than the MLP baseline model. Another interesting insight from the experiments is that transfer learning models tend to have much lower validation loss during the training, while MLP started off with high validation loss, as observed in Figure 4.4. This makes sense because RecoBERT is pre-trained with sequential recommendation dataset hence it produces useful features for the fine-tuning recommendation tasks. As a result, validation loss is lower.

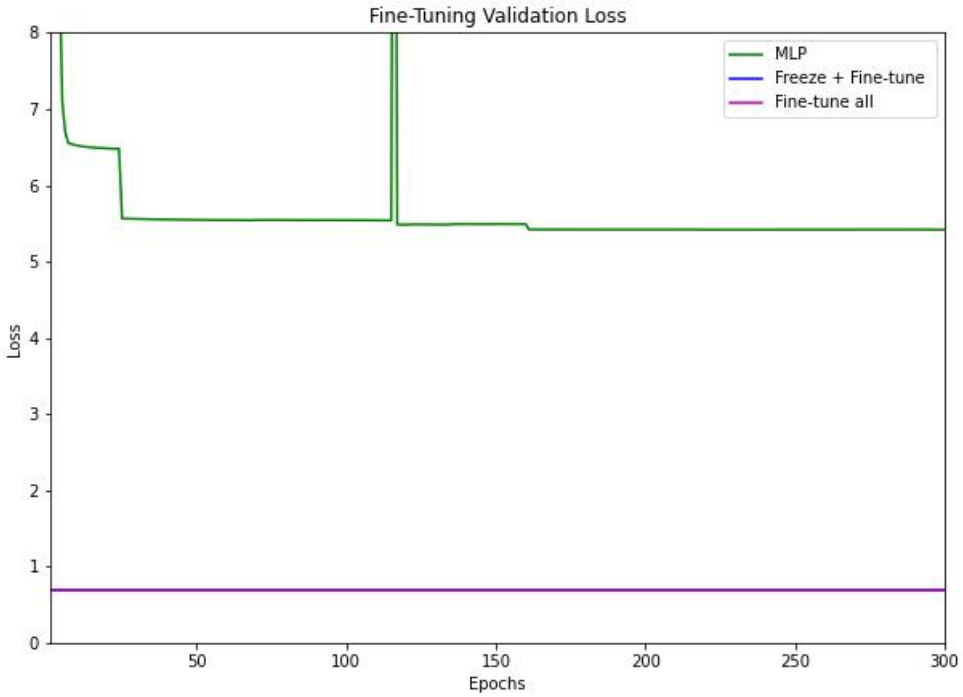

FIGURE (4.4) Age Groups Validation Loss

Chapter 5

Conclusion and Future Work

In this thesis we introduced RecoBERT model which is pre-trained for both Next Sentence classification and Masked Language Modeling tasks using sequential recommendation datasets, so that the model could learn useful features from user's historical behaviours. The pre-trained RecoBERT model is then fine-tuned for downstream tasks, predicting user profile information.

We analyzed recommendation datasets collected from Tencent, and pre-processed users clicking items sequences into positive and negative samples for classification task. In addition, we found imbalance classes in gender and age groups profile information that were used for fine-tuning. Hence, we used resampling technique to overcome this issue.

In pre-training, We have shown that averaging all tokens from encoder output outperforms the first CLS tokens method in Next Sentence classification task. We also discussed how changes of hyperparameters could impact the performance of the model in terms of the losses, accuracy and training time. Interestingly, increasing the minimum masked tokens would lead to better performance in Masked Language Modeling task as RecoBERT learns useful context of datasets through masked tokens.

In fine-tuning downstream tasks, we have proved that freezing the pre-trained RecoBERT model weights, and use the features produced by RecoBERT as inputs to train with a few additional linear layers achieves better results than fine-tuning the entire architecture, and MLP model.

As the extension of present work, we might want to extend pre-training datasets by incorporating user item features (article word counts, movie director's name, news published time and day) into RecoBERT instead of just modeling the user item. Finally, it would be interesting to modify RecoBERT into a twin BERT network with the exact same network weights, we can split each user's historical behaviour sequence into two sequences, sequence A and sequence B. Then, we feed sequence A into siamese [9] BERT A and sequence B into siamese BERT B to create user behaviour sequence embeddings for Next Sentence classification task. We hope RecoBERT would inspire new research work to meet challenges in recommender system.

Bibliography

[1] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[2] Tim Donkers, Benedikt Loepp, and Jürgen Ziegler. "Sequential User-based Recurrent Neural Network Recommendations". en. In: (Aug. 2017), pp. 152–160. DOI: 10.1145/3109859.3109877. URL: https://dl.acm.org/doi/10.1145/3109859.3109877.

[3] Robert M. Gray. *Entropy and Information Theory*. Berlin, Heidelberg: Springer-Verlag, 1990. ISBN: 0387973710.

[4] Balázs Hidasi and Alexandros Karatzoglou. "Recurrent Neural Networks with Top-k Gains for Session-based Recommendations". en. In: (Oct. 2018), pp. 843–852. DOI: 10.1145/3269206.3271761. URL: https://dl.acm.org/doi/10.1145/3269206.3271761.

[5] J. J. Hopfield. "Neurons with a graded response have collective computational properties like those of two–state neurons". In: *Proceedings of the National Academy of Science USA* 81 (May 1984), pp. 3088–3092.

[6] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: (2014). URL: http://arxiv.org/abs/1412.6980.

[7] Thomas Kurbiel and Shahrzad Khaleghian. "Training of Deep Neural Networks based on Distance Measures using RMSProp". In: *CoRR* abs/1708.01911 (2017). arXiv: 1708.01911. URL: http://arxiv.org/abs/1708.01911.

[8] Jerry Ma and Denis Yarats. "Quasi-hyperbolic momentum and Adam for deep learning". In: *CoRR* abs/1810.06801 (2018). arXiv: 1810.06801. URL: http://arxiv.org/abs/1810.06801.

[9] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". en. In: (2019), pp. 3980–3990. DOI: 10.18653/v1/D19-1410. URL: https://www.aclweb.org/anthology/D19-1410 (visited on 09/11/2022).

[10] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747.

[11] Fei Sun et al. "BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer". In: (2019). Ed. by Wenwu Zhu et al., pp. 1441–1450. DOI: 10.1145/3357384.3357895.

[12] Ashish Vaswani et al. "Attention is all you need". In: (2017), pp. 5998–6008.

[13] Chao-Yuan Wu et al. "Recurrent Recommender Networks". en. In: (Feb. 2017), pp. 495–503. DOI: 10.1145/3018661.3018689. URL: https://dl.acm.org/doi/10.1145/3018661.3018689.

[14] Fajie Yuan et al. "Parameter-Efficient Transfer from Sequential Behaviors for User Modeling and Recommendation". In: *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval* (2020).