Transformer Recommenders

Haoping Lin, Yang Yao

April 2023

Abstract

In this project, we implemented a recommender based on transformer to excavate latent sequential patterns beneath data. From introducing crucial mechanism used in transformer, to explaining training process, to evaluating models with example, we will discuss how BERT(Bidirectional Encoder Representations from Transformers) works as a recommender.

Introduction

Recommendation systems are becoming increasingly popular in today's digital age. These systems are designed to provide personalized recommendations to users based on their past behavior, preferences, and interests. They are widely used in various domains, such as social media, and entertainment, and e-commerce, to enhance user experience and improve engagement.

The origins of modern recommendation systems date back to the early 1990s when they were mainly applied experimentally to personal email and information filtering. One of the most famous and classical recommendation method should be the Matrix Factorization which implements the concept of collaborative filtering. In recent years, due to flourishing of AI and deep neural network, the emergence of recommendation systems applicable to different domains have been more fully developed.

Despite well performance from transitional recommendation system, we propose that recommenders with ability to detect and learn possible hidden sequential patterns would perform better and more precisely in some situations, especially for dataset that demonstrate sequence. Thus, we choose to implement BERT on our recommender, which take advantage of sequential learning ability from transformers. In this report, we will discuss the architecture of the model, the training process and strategy, validation and test results to gain a comprehensive understating on this model and evaluate its possibility.

Architecture

To better understanding the architecture of a BERT model, it is essential and useful to learn key concepts and mechanisms behind. These concepts are so important that it decides what and how transformer learn. We will start from the most essential mechanism in a transformer model called self attention(Figure 1).

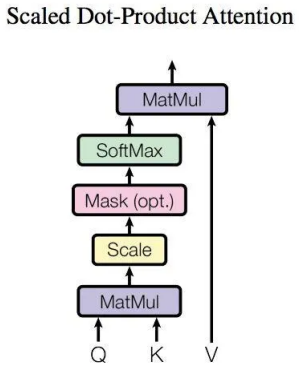

Figure 1: Self-Attention

The concept of self-attention is proposed by google scholars in paper Attention Is All You Need in 2017. This mechanism in deep learning models allows the model to focus on different parts of the input sequence and selectively weigh the importance of each part when generating an output. It is first applied in the case of natural language processing tasks, such as language translation or sentiment analysis. It can help the model to recognize which words in the input sequence are most important for understanding the meaning of the sentence or paragraph. The reason why this mechanism can also be applied to recommendation models is that item set could also be sorted by some sequence. For example, time sequence is a great feature be considered when dealing with historical behaviors. Each item can be think of as a word in sentences while historical behavior of a user could be think of as the sentence. The transformer will try to understand sequential relationship between items just like learning patterns of words in sentence.

Self-attention works by computing a set of attention weights for each position in the input sequence based on the relationships between all the other positions. These attention weights are used to compute a weighted sum of the input sequence, which represents the context vector for each position. Therefore, there are two layers of embedding applied to each item before input it to the model. These two layers are called value embedding and positional embedding. Value embedding is not difficult to understand as we map tokens Ids to vectors of fixed length. Then it comes to the positional embedding which represents the position of an item in a give sequence. Results of two layers will be added

together and used for computing of Query, Key and Value(K, Q, V).

K, Q, V is not something special but just matrix of linear transformation from embedding result of an item. Among these values, K and Q will be used to learn and compute importance of an item related to another. We call it a score. In general, higher score means higher importance on this word. Then the score will be scaled and multiplied back to V. So far, a basic idea of self-attention is implemented.

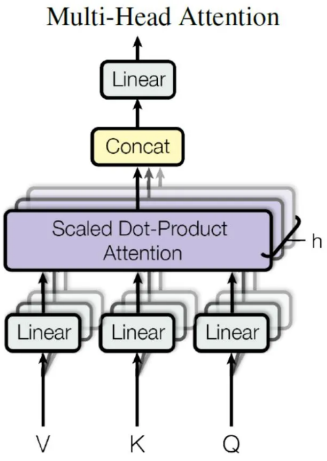

Figure 2: Multi-head Attention

Next, one-head attention and multi-head attention(Figure 2) will be introduced to understand how self-attention are contributing to the whole model. One-head attention is a simple form of attention, where one set of K, Q, V is used to compute the score of each item, based on their relevance to a given sequence vector. However, it can only represent one kind of relation between words, which may limit the model's ability to learn more patterns. Therefore, to improve its capture and learning ability, multi-head attention appears. Multi-head attention, on the other hand, is a more complex form of attention that allows for multiple sets of K, Q, V to be used simultaneously to compute multiple context vectors. These context vectors can then be concatenated and passed through a linear transformation to produce the final output. Based on different linear transformations, multi-head attention could learn different relations between words that its output contain more information. In our project, multi-head attention empowered our model with ability to recommend similar movies to users base on their viewing history without giving genre information to the model.

With necessary understanding of self-attention and multi-head, we can built a basic transformer model by adding dropout layer and norm layer. These two layers are used to avoid overfitting and make training faster. When constructing the whole model, outputs of a transformer block are imported to all transformer

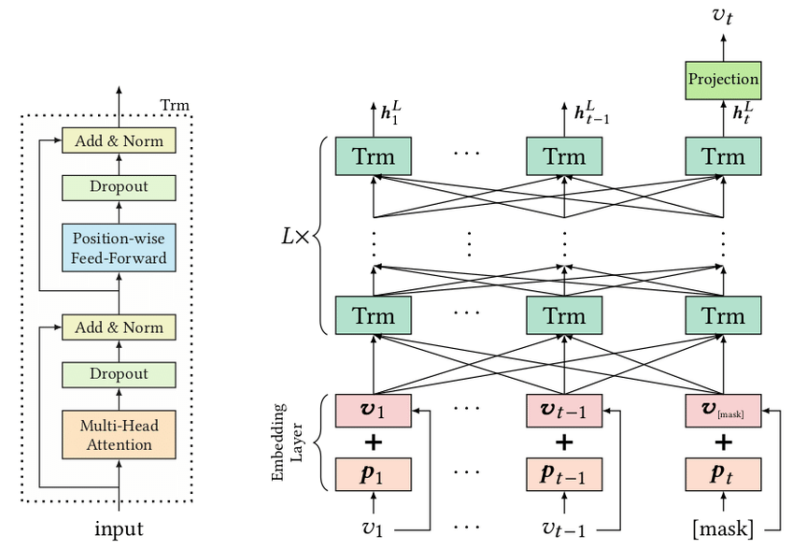

Figure 3: BERT Architecture

blocks in the next layer as the model tries to learn the context of an item. Therefore, it requires result not only from items before, but also from after. This is where bidirectional concept comes from.

Dataset and Preprocessing

To gain better training result, we choose the dataset called MoiveLens 20M(ML-20M), which is a collection of rating information with user and movies. This dataset contains about 20 million rating records with 465,000 tag applications applied to 27,000 movies by 138,000 users ranging from year 1995 to 2015. Among the whole dataset, we choose to use Ratings.csv(Figure 4) which contains columns of userId, movieId, ratings and timestamp, since its data fits the input of our model best.

For preprocessing of the data(Figure 5), we only choose data where rating more than 4. We want to collect positive behaviors so that rating less than 4 are seen as negative behaviors and will be filtered out. After that, reamaining date are grouped by userId to get a list of behaviors made up of movieIds for each user. Each list of movieIds is then sorted by timestamp to obey the time sequence. Simage/process.png we transformed the raw data to a time sequential dataset grouped by users.

ratings

userId	movieId	rating	timestamp
1	2	3.5	1112486027
1	29	3.5	1112484676
1	32	3.5	1112484819
1	47	3.5	1112484727
1	50	3.5	1112484580
1	112	3.5	1094785740
1	151	4.0	1094785734
1	223	4.0	1112485573
1	253	4.0	1112484940

Figure 4: Example of Dataset

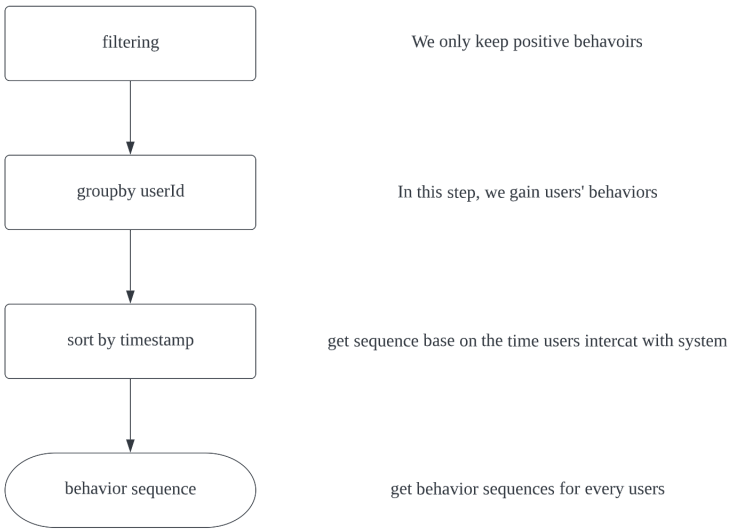

Figure 5: Data Preprocessing

Training

To train the BERT4Rec model, we start by several steps of data processing to prepare proper format of input data. As stated in the architecture section, each input is a sequence, or we call it a batch in training process of length 128. To ensure all input sequence reach the fixed length, we use zero as padding token to fill empty space in some sequence. This guarantees a consistency of matrix dimension during training. Besides, we performer masked operation with specified probability to improve model's performance and generalization. During this process, 85% of tokens remain the same, while 15% of tokens are somehow masked. Among masked tokens, 80% tokens are masked with specified mask token one, while 10% are replaced with some random token value and 10% are not changed. Masked prediction allows the model to focus only on the masked tokens, rather than processing the entire input sequence. At the same time, masked prediction handles some situations with noisy data and encourage the model to learn more general representations from dataset.

With formatted data, we are ready to train! Some important training parameters are given and explained below. We trained a total of 100 epochs with a start learning rate of 0.001. For each 25 epochs, the learning rate will be reduced by 10 times, that is 10% of previous learning rate. The reason behind is for faster convergence at the beginning while not overfitting after stages. In each epoch, we iterate batches over formatted data using dataloader in Pytorch. Again, each batch is a sequence of 128 tokens with padding and mask. While training, we use TensorBoard and tqdm to monitor progress and parameters of model. For loss function, we choose cross entropy and the loss(Figure 6) starts from 8 and dropped beneath 6 in several epochs of training. With smaller learning rate later, the loss value gradually turns to 5 and does not make much progress.

According our model structures and training strategy, we conclude two main reasons for the loss. First, learning rate becomes smaller as training goes on, which means model tends to stop learning from dataset after some time. Second, our BERT model only contains two layers of transformers, which limits the learning ability of the model. We believe that a BERT model with more layers of transformers would strengthen its learning ability and thus reach a lower loss and higher performance.

In general, input data are fetched and processed into a batch of 128 items with necessary padding and mask. By controlling mask rate and changing learning rate, the model is not overfitted during training. We followed its loss and validation score epoch by epoch and store model parameters locally. After 100 epochs about 14 million of data, we finally reach a loss rate of 5, a NDGC@20 of 0.7864, and a recall@20 of 0.9802. With this trained model, we are ready to move to next session to test our final results.

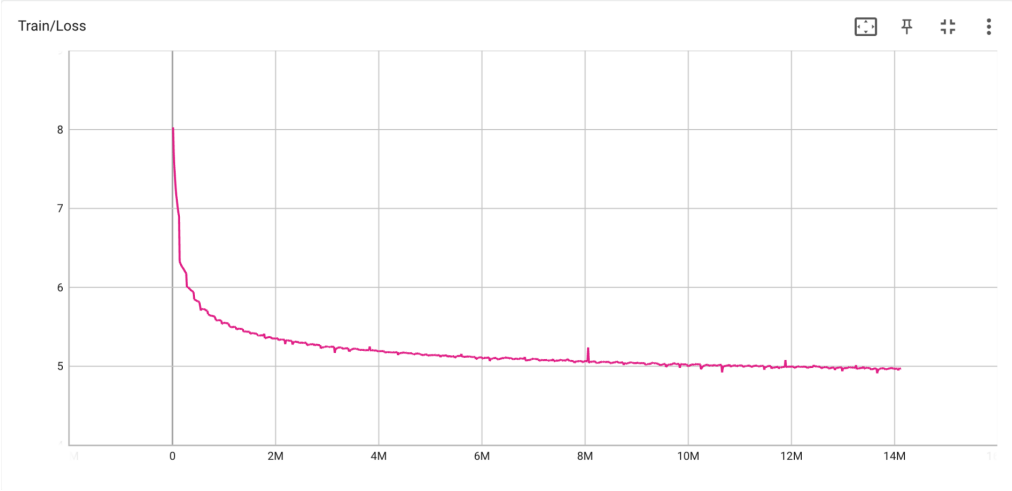

Figure 6: Loss

Results

To test our model, we implement our model and use movies from original data and convert model output to readable recommendation set.

```
seq = ["Harry Potter and the Sorcerer's Stone (a.k.a. Harry Potter and the Philosopher's Stone) (2001)",
       'Harry Potter and the Chamber of Secrets (2002)',
       'Harry Potter and the Prisoner of Azkaban (2004)',
       'Harry Potter and the Goblet of Fire (2005)'
      ]
res = pridct(model,seq,smap,reverse_smap,moive2id,id2movie,10)
```

✓ 0.3s

```
['Harry Potter and the Order of the Phoenix (2007)', 'Adventure|Drama|Fantasy|IMAX']
['Harry Potter and the Half-Blood Prince (2009)', 'Adventure|Fantasy|Mystery|Romance|IMAX']
['Lord of the Rings: The Two Towers, The (2002)', 'Adventure|Fantasy']
['Chronicles of Narnia: The Lion, the Witch and the Wardrobe, The (2005)', 'Adventure|Children|Fantasy']
['Lord of the Rings: The Fellowship of the Ring, The (2001)', 'Adventure|Fantasy']
['Harry Potter and the Deathly Hallows: Part 2 (2011)', 'Action|Adventure|Drama|Fantasy|Mystery|IMAX']
['Pirates of the Caribbean: The Curse of the Black Pearl (2003)', 'Action|Adventure|Comedy|Fantasy']
['Twilight (2008)', 'Drama|Fantasy|Romance|Thriller']
['Spirited Away (Sen to Chihiro no kamikakushi) (2001)', 'Adventure|Animation|Fantasy']
['Dark Knight, The (2008)', 'Action|Crime|Drama|IMAX']
['Shawshank Redemption, The (1994)', 'Crime|Drama']
```

Figure 7: Result

For input, we choose a set of four movies that has sequence in series and time. Based on this, what model recommends for us is a set of 10 movies. From the result, we can see that the first two movies are also in series of input movies and these two movies also follow the time sequence pattern. Besides,

other movies from results are mostly in genre of adventure and fantasy. Even though we do not provide any genre information as input, but based on previous phenomenon we observed, we conclude that the model in fact learnt genre by understanding relationship between movies. This is where self-attention taking effect, which helps the model explore importance of relationship between movies. Although, there is a special case, which is the 6th output of movie. It seems to jump out of sequence pattern of input and does not have a high importance relationship with input data. From our perspectives, we realize the insufficient of a BERT model that number of transformer layers may varies in different situations. To optimize the model, more attempts and trainings are required. Therefore, we can say that the BERT model has the ability performing well as a recommendation system in most cases and needs to be varied based on specific feature of dataset.

Conclusions

BERT4Rec is a promising method for recommendation systems that leverages the power of BERT to encode item descriptions and user behavior into a high-dimensional space. It has advantages such as learning sequential patterns and importance of relationship from dataset. While based on different features of dataset, architecture of model and training process are required to changed to gain better performance. BERT is still a great example of implementing deep neural network in recommendation system. With further exploration, BERT will show its important roles in different areas.

Contribution

Yang Yao: 1. BERT Model Construction; 2. Data Preprocessing
Haoping Lin: 1. Training code 2. Test Result