Session Based Recommender System

Name: Thummanoon Kunauntakij
Student ID: 12122522

Problem Setting

A Session-Based Recommender System is a task in a recommender system where a system tries to recommend an item to an unknown user based on the list of things the user views or interacts with during a session. This problem differs from a conventional recommender system in that the system is unaware of information about their background (long-term preference). In other words, the systems don't know items the user rated or purchased in the past.

This task is practical as it appears on all e-commerce websites when a random user visits a website (or before a member signs in). It is also helpful when user tastes change quickly, such as in the case of electronic devices, e.g., a popular smartphone from previous years may not be interesting for customers this year. Because the system doesn't take user history into account, it can recommend items based on user interest based on the current session. Not only its practical usage, but this task is also pleasing in its technical part. We can use many machine learning methods to solve it, including traditional methods like collaborative filtering and Markov chain, as well as recent deep learning methods such as recurrent neural networks, transformers, and graph neural networks.

This project framed the session-based recommender task as a next-item recommendation task. In other words, The system recommends a ranking of items that a user would interact with. An ideal system recommends such items at the top of the list.

Dataset: eCommerce events history in electronics store

(https://www.kaggle.com/datasets/mkechinov/ecommerce-events-history-in-electronics-store)

This original dataset contains 7 months of user interaction data (October 2019 to February 2020) on products listed on an online electronic store. Interactions can be item viewing, adding or removing an item from the shopping cart or purchasing. Each user session can have multiple item interactions on one or more items. They can contain multiple purchases as well.

In this project, I preprocessed the data by the following steps.

1. Because this project focused on a session based recommender. The user ID is removed.
2. Removed rows without user session ID.
3. Sorted the dataset by session ID and timestamp. Repeated interactions on the same items are removed.
4. Removed sessions that contained only one interaction left.
5. Use sessions in January 2020 as validation and February 2020 as a testing set. All sessions that lasted over the splitting sessions are removed.

The result dataset has 68,004 sessions with 26,884 unique items. The average length of all sessions is 3.5 interactions. The shortest session has 2 interactions and the longest one has 418 sessions. About 40,000 of the sessions are in the training set while both the validation and testing set have about 14,000 sessions.

Evaluation Metrics:

The system is evaluated by giving the sequence of all items except the last one from each session. The models have to recommend the top 20 items from all available items.

There are 2 metrics used.

1. **Recall@20**
   The percentage of the next-item appears in the top-20 ranking.

2. **MRR@20** (Mean Reciprocal Rank)
   It is the reciprocal of the ranking of the next-item in the top-20 ranking (1/Rank) averaged over the batch. If it is not in the rank, the value is 0.

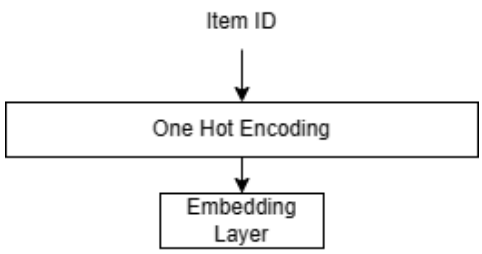
Target evaluation metrics are **Recall@20 > 0.2** and **MRR@20 > 0.2**. In other words, a target recommender should recommend the next item within the top 5.

Implemented Solution

In this project, I have implemented 2 different models. The first one is an RNN using GRU units and the other is a transformer model. All of the deep learning models are done using Pytorch and Pytorch Lightning.

Item Embedding

First, we give a unique item ID to each item in the inventory. We treat each item as a token, just like how NLP represents a word as a token in a sentence. Then, we convert the number to an array using one hot encoding. Then, we feed the array to an embedding layer to learn the lower-dimension feature representation of each item.



Item Embedding

Recurrent Neural Network

General Idea

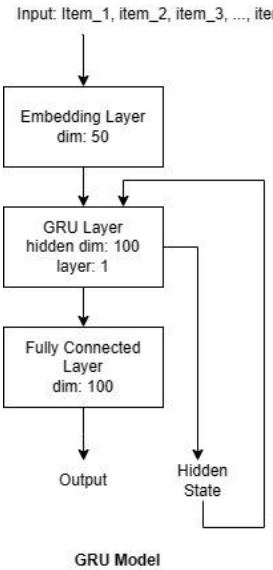
The first approach is by framing this task as a next-word prediction task in NLP. We can train a machine learning model by feeding it with a sequence of items one by one and asking it to predict the next item in the session. Then, we can sort the items by probability and calculate the loss (using Top1 loss according to [1]) by considering the rank of the actual next item from the model output. The result is the probability of all items in the inventory.

Architecture

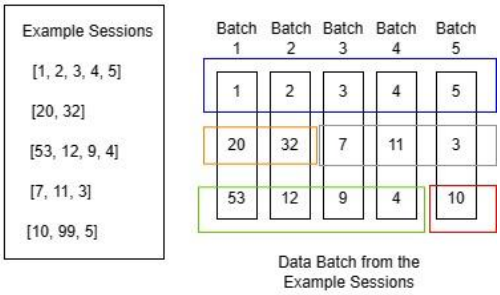
The first approach is the recurrent neural network. It is implemented according to work by [1]. The model contains three layers; an embedding layer, a gated recurrent unit (GRU) layer, and a fully connected layer. In this project, my optimized model has an embedding layer with 50 dimensions, a GRU layer with a hidden state of 100, and a fully connected layer with 100 dimensions. Finally, a ReLU activation layer is applied to the output of the fully connected layer.

Data Batch

To train the model with batches of size N, the first N sessions are stacked. The first batch is the first item of the sessions. The next batches are the next items of each batch accordingly. When a session reaches the end, items from the next unused session replace the slot and the hidden state of the RNN model is reset. For example, in the diagram below, I have demonstrated how to construct batches of size 3. The first 3 sessions are stacked, and we have the first batch with item 1, 20, and 53. Then, the second batch is the next item from each session. In the third batch, the second session is exhausted. We can replace it with the fourth session. So the third batch contains items 3, 7, and 9. We then continue similarly until all of the sessions are used. This will conclude an epoch.



GRU Model



Data Batch from the Example Sessions

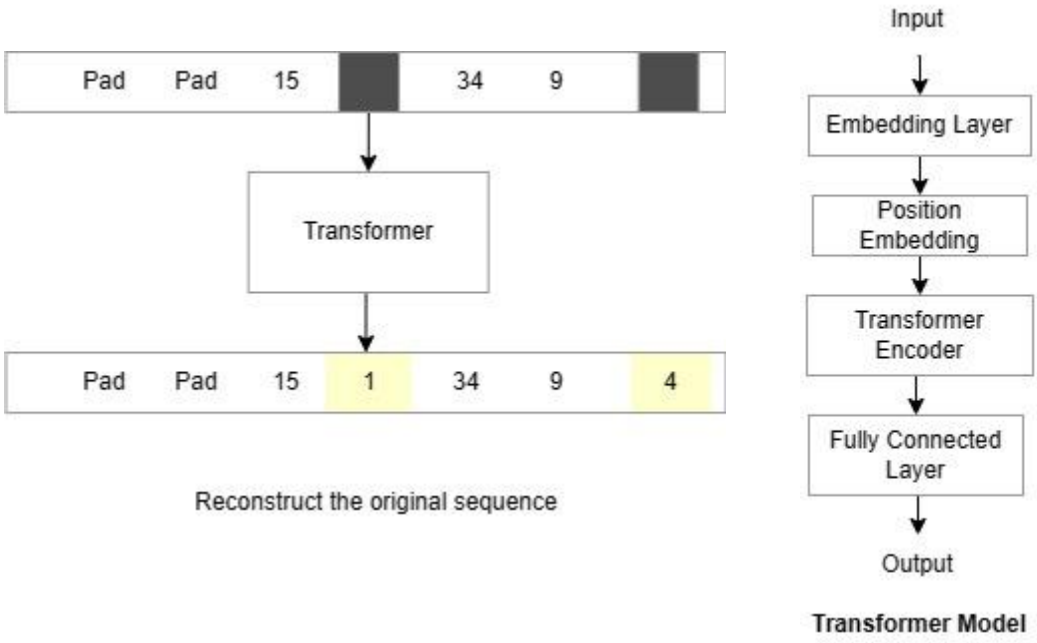
Transformer

General Idea

The second approach is to use a transformer to learn item sequences. There could be many ways to frame our task to be learned by a transformer mode. In this project, we train a transformer by feeding it with masked item sequences according to [2]. Then, we train the model to reconstruct the original sequence. Unlike RNN, the transformer model doesn't need to learn the interaction of items concerning the ordering. The transformer should be able to select the interaction freely by self-attention.

Input Data

As it can be seen from the diagram, an input sequence is a sequence of 10 most recent items. Shorter sequences are padding in the front. Then, the final item of the sequence is masked. Optionally, we choose to randomly mask other items in the sequence as well.



Architecture

The transformer model contains 4 types of layers. The first layer is an embedding layer with 150 dimensions. The second part is a positional encoder. It calculates features to denote the position of each item in a sequence. Then, we stack both outputs of the embedded and positional features and feed them to the third layer, a transformer encoder with 5 layers, 5 attention heads and 150 hidden state dimensions. This layer learns the interaction between each item by self-attention. The output of this layer is a feature representation of the input sequence. We feed this representation into the final fully connected layer for classification.

Result

RNN with GRU is trained with 30 epochs and the transformer model is trained with 5 epochs. The number of epochs is decided by the lowest loss value from the validation set.

For validation set (Sessions data from January 2021)

Model	**Recall@20**	**MRR@20**
RNN with GRU	0.3115	0.1317
Transformer	0.0647	0.0198

For testing set (Session data from February 2021)

Model	**Recall@20**	**MRR@20**
RNN with GRU	0.2922	0.1242
Transformer	0.0581	0.0173

Discussion

We can see that RNN with GRU model delivered a good result in **Recall@20**, although **MRR@20** is still lower than the target. On the other hand, the transformer model cannot achieve anything as near. This might be due to an inappropriate way that I set up the model for this task and dataset.

It is clear that deep learning is a promising method for session-based recommender systems. The model can learn to deliver such a good result with only a sequence of item ids without additional information about the item. In the demo application, we can see clearly that the recommended items are relevant as they are in a related category. However, we have to take into account multiple desired properties of recommendation and test further if we want to deploy the model to real users. For example, a user might find that the recommended list is not attractive as there is nothing new to them. Also, when new items are added, the model might need to be adapted faster to recommend them to users.

Lesson Learned

1. **Different way to batch data into the model**
   In the RNN model, the way we feed the data allows the session length to be variable. We don't need to trim the session length to a shorter one as we did for the transformer model.

2. **Different way to use transformer for the task**

Although the transformer model is powerful, there are multiple ways to set up a task for it, and in this case, I could not train it successfully as I initially hoped.

3. **Loss/Metric is a proxy to the goal but may not be the ultimate goal.**
   In this case, we use Recall@20 and MRR@20 to measure the performance of our recommenders. But in the real world, there are many answers for what items a user might be interested in. I realized when I saw the demo application output that there are many relevant items in the recommended list, except the next item does not appear. The metric for this list will be 0, but in the real world, it provides some value.

4. **The drawback of RNN might be useful for this task**
   RNN usually have trouble with learning interactions in long sequences. But in this task, it can be a positive feature as users might change their mind during the session and RNN can adapt quicker than a more complex model.

Project Management Reflection

During the project there are 2 obstacles that consume a significant portion of the time budget.

1. In the early part of the implementation, I realize that my assumption about the data is incorrect. Firstly, I want to approach this project with purchase-item prediction. But the quality of the dataset I picked wasn't as good as I assumed. After cleaning and reformatting it to fit the original task, there were only a thousand sessions left which needed to be larger for the training. Thus, I change my approach and work on next-item prediction instead.

2. I have little experience implementing RNN and Transformer models before this task. The hardest part is framing the task to feed into the models and enable them to learn correctly. I have been successful with RNN but not with the Transformer. Implementing and debugging the data loader and calculating each model's loss and evaluation metrics consume a lot of time. Because both models work and produce output differently, it takes a long time to figure out which part of the output is relevant and how to compute loss and evaluation metrics from them.

In the end, I spent about 60 hours on the project. They are divided roughly equally amongst data cleanup (including confusion from point 1.), the RNN model, and the transformer model. Because the transformer didn’t work as well as expected, I didn’t spend time on the part related to the optimization and implementation of the state-of-the-art model as planned.

Reference

[1] Hidasi, Balázs, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. “Session-Based Recommendations with Recurrent Neural Networks.” arXiv, March 29, 2016. http://arxiv.org/abs/1511.06939.

[2] Lu, Yichao, Zhaolin Gao, Zhaoyue Cheng, Jianing Sun, Bradley Brown, Guangwei Yu, Anson Wong, Felipe Pérez, and Maksims Volkovs. “Session-Based Recommendation with Transformers.” In RecSys Challenge 2022, 29–33. Seattle WA USA: ACM, 2022. https://doi.org/10.1145/3556702.3556844.