CSIT5210 Project Final Report

Implementation of Towards Universal Sequence Representation Learning for Recommender Systems

Project Type: Implementation
Group No: 9
Members Information
Student ID:20881383 Student Name: YAN, Xi
Student ID:20929070 Student Name: ZHONG, Lintao
Student ID:20928868 Student Name: WANG, Jiatong
Student ID:20927307 Student Name: CHANG Zijiang
Declaration Statement: This project is done solely within the course but not other scopes

Abstract

The paper we want to implement propose a model called UniSRec which can generate universal sequence representation for recommendation system. The Model utilizes BERT, parameter whiting and Mixture of Expert to generate universal item text representation. Then design two contrastive learning task for pre-training and fiun-tune the model for a new domain. This report mainly cover the research for the model and algorithm, we propose another pre-training framework for implementation and we set up the experiments and give our own conclusions.

1 Introduction

The sequential recommendation model has been one of the widely researched topics in the recommendation field. Its goal is to make accurate and appropriate item recommendations for users' historical activity or interaction records. The process of the sequence recommendation model can be roughly divided into two stages. The first stage is to convert the user's historical behavior into a sequence that can be learned. The second stage is to build an efficient model to capture the characteristic patterns in the historical sequence that can reflect user preferences. As a result, the model can recommend items according to the user's preferences, thus playing a huge role in shopping, music, games, and other fields.

Most current methods focus more on building models, and many effective models have emerged from these methods, such as Markov Chains (MCs) and sequential neural networks (GRU4Rec and Caser). Transformer based model [1] achieves outstanding results by introducing 'self-attention' to efficiently obtain syntactic and semantic patterns between words in a sentence.

However, most methods today rely heavily on item IDs to build sequence models, making it difficult to transfer the trained model to a new recommendation scenario. Even if the data types in different scenarios are exactly the same, the sets of item IDs in different recommendation domains are usually different, which limits the portability of item representations in these methods. Faced with this problem, we often need to retrain a model to better adapt to new recommendation scenarios, which is very tedious and resource-consuming. Therefore, the author proposes two main challenges in this paper: 1) Directly using the original text features for learning often brings terrible results, and we need to extract a more general text sequence representation; 2) Learning data and patterns from multiple domains can lead to conflicts and oscillations.

Recent research results show that the Pre-trained Language Model (PLM) can learn more general text features in different domains, thus effectively eliminating the semantic gap in different domains. Inspired by this, the authors propose **UniSRec**, a universal sequence representation learning method. Its core idea is to no longer rely on item IDs but to directly learn item-related texts (such as titles,

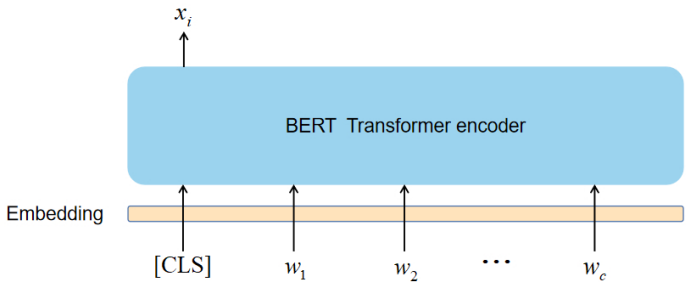

Figure 1: Item representation through BERT

descriptions, etc.) to obtain item representations and sequence representations that can be transferred between different domains and tasks. The article designs a lightweight architecture based on parametric whitening and mixture-of-experts enhanced adapter to integrate and adapt the text features learned by the pre-trained language model in different domains. Furthermore, two contrastive learning methods are introduced, sequence-item and sequence-sequence, to fine-tune the pre-trained model to adapt to new recommendation scenarios. This way, the pre-trained model can efficiently learn more general text sequences and quickly transfer to new recommendation scenarios. Subsequent experimental results also proved the efficiency and effectiveness of this approach.

2 Model

In this section we will discuss the models in the paper and we will explain these models from the following perspective: the techniques and algorithm behind the model, the purpose to design the model, the structure inside the model and how to train the model.

2.1 Universal Textual Item Representation

In this subsection, we will explain how to obtain the universal textual item representation through BERT and MoE model.

In order to build a universal sequential recommendation system, it is necessary to map items from various recommendation scenarios (different domains or platforms) to a semantic space that each item representation can be measured in this space.

This paper aims to learn the universal item representations based on the text describing the item attributes and features in natural language model. To enhance the transfer ability of item representation, they assume that item IDs is not avaliable and item representations will not consider ID embeddings. This is because ID might not be useful to indentify the item from different domain. Instead, we want to recognize one item according to its descriptions.

At first, the paper use a popular pre-trained language model BERT, which has achieved tremendous success in recent year, to learn the representations of items. Given an item and the text describing it, a special symbol [CLS] is concatenated to the words of item text, denoted as $w1, w2, ..., wc$ in order. The concatenated sequence is the input of the BERT model, and we have the following output from BERT model:

$$x_i = \text{BERT}([[\text{CLS}]; w_1, \ldots, w_c]), \tag{1}$$

where x_i is the final hidden vector of the first input token ([CLS]). The special symbol [CLS] is viewed as a special token, and it corresponding final output x_i can be viewed as the representation of the whole input sequence, and usually can be used for classification task. The calculation process of item representation from description text is shown in Figure 1.

Although we can obtain textual item representation from BERT, they are not suitable for recommendation tasks. Because recent research has found that, BERT maps text to an anisotropic semantic space. Suppose we have two vector, $x \in \mathbb{R}^d$ and $y \in \mathbb{R}^d$, and the cosine similarity is calculated as:

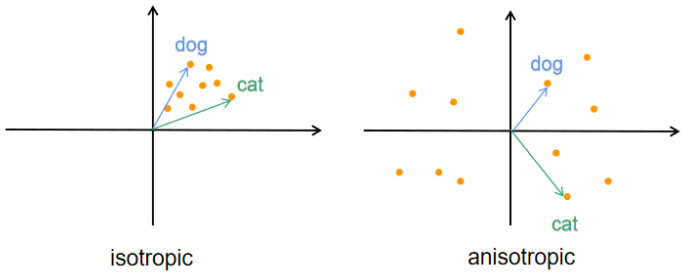

Figure 2: Illustration of istropic and anisotropic

$$\cos(x, y) = \frac{\sum_i^d x_i y_i}{\sqrt{\sum_i^d x_i^2}\sqrt{\sum_i^d y_i^2}}, \tag{2}$$

and Eqn. (2) only satisfied on standard orthogonal basis. But the similarity of word embedding produced by BERT can not be measured by cosine similarity, which means the coordinate basis of the BERT to generate vector is not a standard orthogonal basis, that is anisotropy.

The disadvantage of anisotropy is that text embeddings are distributed in a narrow vector space. As shown in Figure 2, dog and cat are totally different things. But in an anisotropy vector space, the cosine similarity of their embedding would be high.

In this task, multiple items texts with large semantic gaps need to be mixed which makes this question more serious. So the original output of BERT is not a good representation.

There are many ways to eliminate anisotropy of BERT, such as flow-based generative models and whitening. Flow-based generative models calibrates the original distribution to standard Gaussian distribution which is isotropic.

Another popular way is parameter whitening. Through linear transformation, whitening turns the average of the vector set into 0 and the covariance matrix into a unit matrix, which satisfies the spatial distribution of isotropic vectors, then we can obtain better generalizability on unknown domains. The formula for parameter whitening is:

$$\tilde{x}_i = (x_i - \mu)W \tag{3}$$

$$\mu = \frac{1}{M}\sum_{i=0}^{M} x_i \tag{4}$$

$$\Sigma = \frac{1}{M}\sum_{i=0}^{M}(x_i - \mu)^\top (x_i - \mu), \tag{5}$$

where M is the number of samples.

The transformed covariance matrix is $\tilde{\Sigma}$, and we want it to be unit matrix, so we have:

$$\tilde{\Sigma} = W^\top \Sigma W \tag{6}$$

$$W^\top \Sigma W = I \tag{7}$$

$$\Sigma = (W^{-1})^\top W^{-1} \tag{8}$$

$$\tag{9}$$

By SVD decomposition, we obtain the weight matrix:

$$\Sigma = U\Lambda U^\top \tag{10}$$

$$W = U\sqrt{\Lambda^{-1}} \tag{11}$$

$$\tag{12}$$

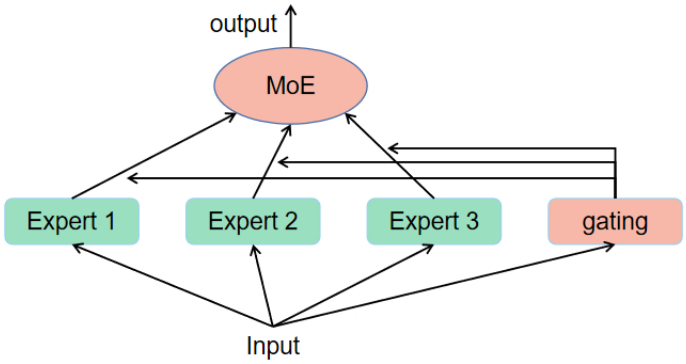

Figure 3: Structure of MoE

Different from the original whitening, parameters W and μ are not preset or calculated, but set as a learnable parameter and learned through training. We give another formula:

$$\tilde{x}_i = (x_i - b)W_1, \tag{13}$$

where $b \in \mathbb{R}^{d_W}$ and $W_1 \in \mathbb{R}^{d_W \times d_V}$ are learnable parameters, $\tilde{x}_i \in \mathbb{R}^{d_V}$ is the whited embedding.

There is usually a large semantic gap between different domains. To handle this problem, author set several parameter whitening embeddings for an item to learn different representations from them. Then they apply the mixture-of-expert (MoE) architecture to combine representations produce by each parameter whitening module and output a universal item representation.

Mixture of Experts(MoE), as shown in Figure 3, is an integrated learning technique. When prediction tasks are complex, they can be divided into subtasks for processing. The module that handles each subtask is called an expert. And MoE model will learn to combine the output of these experts to generate new representations.

In this task, the experts are different parameter whitening modules. The output of MoE module is:

$$v_i = \sum_{i=1}^{G} g_k \cdot x_i^{(k)}, \tag{14}$$

where $x_i^{(k)}$ is the output of the k-th parameter whitening module, and the weight g_k is from the gating router which controls the information give by k-th whitening module, it is defined as follows:

$$g = \text{Softmax}(x_i \cdot W_2 + \delta) \tag{15}$$

$$\delta = \text{Norm}() \cdot \text{Softplus}(x_i \cdot W_3), \tag{16}$$

where W_1, $W_2 \in \mathbb{R}^{d_W \times G}$ are learnable parameters and $g \in \mathbb{R}^G$ is the weights vector for experts. Norm() is used to generate Gaussian noise. In Eqn. (15), (16), we apply BERT embedding x_i as the input of the gate router, because it contains domain-specific semantic information. The gating router is similar to the gate unit in LSTM and GRU model.

There are several advantages of this approach. Firstly, it can learn more powerful representations of items form different parameter whitening module. Secondly, the module is learnable and adaptive, which can transfer and fuse information across domains better. Finally, it will cost less time when the model is adopted to a new domain as MoE is a lightweight module.

In paper, they do not fine-tune any paramsters in the BERT model, and only the parameters in MoE and parameter whiteing is updated. The training process is combined with the universal sequence representation training task.

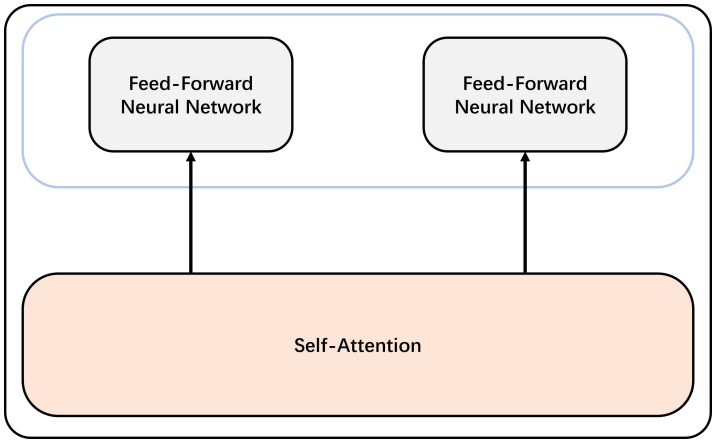

Figure 4: Architecture of Transformer encoder

2.2 Universal Sequence Representation

In this s bsection we will explain how to obtain the universal sequence representation through transformer encoder model.

Through the BERT and MoE model, we obtained a universal item representation v_i, but the recommender system does not built on a single item, it is rather give the recommended item according to the sequential behavior. So the next step is to obtain a sequence representation to represents the sequential behavior pattern accross multiple domains. In the paper, the author uses a method similar to SASRec [1] to encode the behavior sequence, using the Transformer encoder as the user behavior encoder to establish a sequence pattern according to the learned universal item representation. As an efficient self-attention architecture, the Transformer encoder consists of stacks of multi-head self-attention layers and point-wise feed-forward neural networks. The architecture is shown in Figure 4.

The encoding process is shown in following:

$$f_j^0 = v_i + p_j \tag{17}$$

$$F^{l+1} = \text{FFN}(\text{MHAten}(F^l)), \tag{18}$$

where $F^l = [f_0^l; ...; f_n^l]$ represents the concatenated representations at layer l in the user behaviour encoder. The initial behavior sequence expression f_i^0 is the sum of the universal item representation v_i and the absolute position embeddings p_j as input at position j, and the behavior sequence expression at each position constitutes the initial input F^0. FFN represents the point-wise feed-forward neural network, and $MHAttn$ represents the calculation process of the multi-head attention layer.

Finally, we take f_n^L, which is the output of the final hidden layer of the behavior encoder (a total of L layers) at position n (a total of n positions) as the sequence representation.

2.3 Training Process

The training process consist of two stages, the pre-training stage and fine-tune stage. In this subsection, we will explain how to train the models in pre-training stage and fine tune stage. We think this idea is similar to upstream tasks and downstream tasks in computer vision domain.

2.3.1 Pre-training

If we simply sample interaction sequences from multiple domains for pre-training, then it might to *seesaw phenomenon*, that is there might be a conflict between different behavior pattern. So contrastive learning is introduced to ease the seesaw phenomenon and improve the infusion between different domain in semantic space.

Next we want to introduce some basic concept about contrastive learning and why we use the contrastive learning is utilized for training transformer encoder. Contrastive learning is a kind of

unsupervised learning in machine learning. It aims to learn the common features between the similar instances and distinguish different instances. In our task, we aim to use the contrastive task to train a good encoder, which can encode the similar sequence to similar representation from multiple domains.

In contrastive learning, given an training instances x, we will construct two types of instances, positive instance, denoted as x^+, and negative instance, denoted as x^-. Positive instances are instances similar to x, and negative instances are instances different from x. One natural and traditional way to seperate instances in space is cluster algorithm. But unlike cluster algorithm, in contrastive learning, we do not aim to learning which cluster x belong to, but the representation of the x in space, and we achieve this purpose by reducing the distance from positive instances and increase the distance from negative instances. For example, for a person, in cluster algorithm, we might want to learn which profession he belongs to, but in contrastive learning, we want to leaning the description (representation in semantic space) of his profession, and this description should be similar to those persons with similar professions and different from those with different professions. The object in contrastive learning can be expressed by:

$$d(f(x), f(x^)) \ll d(f(x), f(x^-)) \quad \text{or} \tag{19}$$

$$s(f(x), f(x^)) \gg s(f(x), f(x^-)), \tag{20}$$

where $f(x)$ is the space representation of the data point, d is the distance function and s is the similarity function.

We want to train the transformer encoder to learn a good and universal sequence representation s. So the paper designs two contrastive learning task. Sequence-item contrastive task and Sequence-sequence contrastive task, the sequence to item task intends to capture the structure inside the sequence representation, and sequence to sequence task intends to discriminate the representations of sequences from other sequences from multiple domains. For example, what description should appear to describe a person's profession and how to distinguish one profession description to another.

There are two key points in contrastive learning, loss function and how to construct positive and negative instances. We introduce the loss function first.

In two tasks, the paper both adopt the InfoNCE Loss function, the InforNCE loss function is the most popular loss function in contrastive learning in recent years, which has the following form:

$$l = \mathbb{E}[-\log \frac{\exp(f(x) \cdot f(x^+))}{\exp(f(x) \cdot f(x^+)) + \exp(f(x) \cdot f(x^-))}], \tag{21}$$

which is similar to cross entropy loss function, $f(\cdot)$ is normalized, so $f(x_i) \cdot f(x_j)$ is the cosine similarity. By minimizing this loss, we will increase the similarity between $f(x)$ and $f(x^+)$, and decrease the similarity between $f(x)$ and $f(x^-)$.

Then the paper utilize the framework in simCLR [2] to construct the positive instances and negative instances. There are two keys points in simCLR: 1) the number of reference instance; 2) data augmentation. SimCLR assumes that the location of the the given data point is decided by other instances (positive and negative), so in order to obtain a correct representation in semantic space, we need enough instances, which is 4k to 8k in simCLR and requires 128 TPU. Besides, for a instance, the representation should be same for different view of that instance, the different view of the instance is the augmented operation on the original instance. For example, random cropping, gaussian blur, color filtering for a image.

For both tasks, the paper set a batch of B training instances, each training instance consist of sequential context and the next item and encodes the batch to embedding representation, $\{(s_1, v_1), \ldots, (s_B, v_B)\}$. The batch is constructed by instances from multiple domains, which is helpful for semantic fusion and adaptation across domains. In experiment conducted in paper, the batch size in two tasks is set to 8192.

In sequence to item contrastive task, the next item v_j for s_j is the positive instances and other across-domain item $v_{j'}$ is the negative instance, the loss function is:

$$l_{S-I} = \sum_{j=1}^{B} [-\log \frac{\exp(s_j \cdot v_j/\tau)}{\sum_{j'}^{B} \exp(s_j \cdot v_{j'}/\tau)}] \tag{22}$$

In sequence to sequence contrastive task, the positive instance is the augmented sequential context. There are two augmentation strategies in paper, randomly drops items in context and randomly drop words in item text. The negative instances are the remaining sequence representation in batch, so the the loss function is:

$$l_{S-S} = \sum_{j=1}^{B}[-\log \frac{\exp(s_j \cdot \tilde{s}_j/\tau)}{\sum_{j'}^{B} \exp(s_j \cdot s_{j'}/\tau)}], \tag{23}$$

where $\tilde{s}_j$ is the augmented instance representation.

At the pre-training stage, we optimize both sequence-item task and sequence-sequence task, the jointly loss function is:

$$\mathcal{L}_{PT} = l_{S-I} + \lambda l_{S-S} \tag{24}$$

2.3.2 Fine-tune

In order to make the pretrain model take less time to apply to a new domain, and enhance the flexibility of the model. When the model is adapted to a unknown domain, only a small proportion of parameters in the MoE module will be fine-tuned, while parameters of other structure of the network will be fixed.

There are two training settings in the fine-tuning stage:inductive settings and transductive setting.Transductive setting is base on the item IDs of the target domain while Inductive not.

Inductive setting don't relay on item IDs.When we have little knowledge about new domain and the item IDs is unknown,we can use this setting to finetune our model.

The probability of next item can be predict as:

$$P_I(i_{t+1}|s) = \text{Softmax}(s \cdot v_{i_{t+1}}), \tag{25}$$

where the softmax probability is calculated over the candidate set (the positive item and a set of sampled negatives).

If the training set contains near all the items of the new domain,we can add $ID_emdedding$ to the representation of an item.

The probability can be predicted as:

$$P_T(i_{t+1}|s) = \text{Softmax}(\tilde{s} \cdot (v_{i_{t+1}} + e_{i_{t+1}})), \tag{26}$$

where $\tilde{s}$ is the new universal sequence representation after we add $ID_embedding$ to each item's textual representation.

Since in above equations, we calculate the probability distribution of next item to be recommended, we can use cross-entropy loss to train the parameters in MoE module.

3 Implementation

For the model building part, we will describe the construction method and implementation of some modules. For the Parametric Whitening Module, we used a simple linear transformation and learnable parameters to adjust the model input. The class contains the parameter weight initialization and forward process of the whitening layer.

The framework code is shown as following:

```
class PWLayer(nn.Module):
    def __init__(self, input_size, output_size, dropout=0.0):
        super(PWLayer, self).__init__()
        ...
        self.apply(self._init_weights)
    def _init_weights(self, module):
        module.weight.data.normal_(mean=0.0, std=0.02)
    def forward(self, x):
        return self.lin(self.dropout(x) - self.bias)
```

For the mixture-of-expert, MoE module, we use the gating function to control the fusion weight, and use the Gaussian distribution noise to balance the load. The MoE class includes the creation of gating and noise and the forward process of model parameters.

The framework code is shown as following:

```
class MoEAdaptorLayer(nn.Module):
    def __init__(self, n_exps, layers, dropout=0.0, noise=True):
        super(MoEAdaptorLayer, self).__init__()
        ...
    def noisy_top_k_gating(self, x, train, noise_epsilon=1e-2):
        clean_logits = x @ self.w_gate
        ...
        return gates
    def forward(self, x):
        ...
        return multiple_outputs
```

Using the basic modules, we constructed UnisRec with the following structure to reproduce the paper. SASRec Class includes configuration and loading of basic parameters, model forward process, S-I and S-S comparison learning process, parameter pre-training, parameter fine-tuning and recommendation system prediction modules. The complete network results are presented in the Appendix. Among them, the forward process includes the combination of universal commodity code and position embedding, data LN, dropout and data enhancement process. The Comparative Study part is divided into sequence items comparison task and sequence sequence comparison task. The positive examples of the former come from the items at the next moment of the sequence, the negative examples come from the items in other in-batch domains, and the positive and negative examples of the latter come from the data enhancement, that is, use the mask to generate positive examples, and the other in-batch sequences are negative examples. Finally, a multi-task strategy is used to jointly optimize the proposed two comparative losses.

The framework code is shown as following:

```
class UniSRec(SASRec):
    def __init__(self, config, dataset):
        super().__init__(config, dataset)
        ...
    def forward(self, item_seq, item_emb, item_seq_len):
        ...
        input_emb = item_emb + position_embedding
        ...
        return output # [B H]
    def seq_item_contrastive_task(self, seq_output, same_pos_id, interaction):
        ...
        loss = -torch.log(pos_logits / neg_logits)
        return loss.mean()
    def seq_seq_contrastive_task(self, seq_output, same_pos_id, interaction):
        ...
        loss = -torch.log(pos_logits / neg_logits)
        return loss.mean()
    def pretrain(self, interaction):
        ...
        loss = loss_seq_item + self.lam * loss_seq_seq
        return loss
    def calculate_loss(self, interaction):
        return loss
    def full_sort_predict(self, interaction):
        scores = torch.matmul(seq_output, test_items_emb.transpose(0, 1))
        return scores
```

For training implementation, at the pre-training stage, we implement a different training framework from paper.

Motivation. As mentioned in above section, the paper set a batch size of 8192 to construct the training instances, For a complete epoch training, the batch size of 8192 requires about 66G of GPU RAM to achieve. In the experimental testing phase, we used three GPUs including NVIDIA RTX 3090, NVIDIA TITAN RTX, and NVIDIA TITAN RTX in parallel. It takes about 23 minutes to train an EPOCH in the test. We can know that most local experiments cannot provide such corresponding load and computing power. If the batch size is simply reduced to adapt to local training, the performance of the model will be greatly reduced due to the reduction of negative examples. In order to improve the feasibility of local experiments, we try to find an alternative to implement the pre-training with a small batch size without affecting the training performance.

Inspired by MoCo (Momentum contrast, v-1 [3] and v-2 [4]), another popular and effective framework in contrastive learning which only need the batch size of 256 and outperforms than simCLR, we utilize the idea in MoCo to implement the pre-training stage.

In MoCo, there are three key points: 1) enough number of negative instances; 2) query encoder and momentum encoder; 3) data augmentation and project head. MoCo considers that if the negative instances is enough, we can still learn a satisfying representation in semantic space. So MoCo maintain a queue (first in first out) to record negative instances representation. There are two encoder in MoCo, the query encoder and the key encoder, the query encoder is used to encode training instance x, and key encoder is used to encode the positive instance, x^+, all instances in queue are viewed as negative instances, the loss function is

$$l_{MoCo} = \mathbb{E}[-\log \frac{\exp(q(x) \cdot k(x^+))}{\sum_{i=0}^{K} \exp(q(x) \cdot k_i)}] \tag{27}$$

where q is the query encoder, k is the key encoder, k_i is the negative instance in the queue, and K is the length of queue. Then all outputs of key encoder are inserted into the queue, so for next training batch, representations stored in the queue are different from training instances, so can be viewed as negative instances representation. In MoCo, the loss will not pass to the key encoder, which means the key encoder will not update its parameters according to gradient, instead, it updates its parameters according to query encoder:

$$\theta_k = m \cdot \theta_k + (1 - m) \cdot \theta_q, \tag{28}$$

where θ_k is the parameters in key encoder, θ_q is the parameters in query encoder, and m is the momentum parameter. So key encoder also called as momentum encoder. By momentum updates, key encoder can keep consistency with the query encoder and the instance representations in the queue will be updated to stable learning process. It is a popular trick used in reinforcement leaning. Similar to simCLR, MoCo considers that the data augmentation and a project head can help improve the leaning performance. The data augmentation strategy is same as in paper. We add a project head, which is MLP layer to calculate the similarity, the project head is used to recover the lost information in BERT model and Parameter whitening.

In our pre-training stage tasks, the transformer encoder is the query encoder. The pre-training process is shown in Figure 5.

In sequence to item task, we do not need the momentum encoder, and we insert each positive item representation into the item queue, the loss function is:

$$l_{S-I} = \sum_{j=0}^{B} [-\log \frac{\exp(s_j \cdot v_j^+)}{\sum_{i=0}^{K} \exp(s_j \cdot v_i)}] \tag{29}$$

The code is shown as following:

```
def seq_item_moco(self, seq_output, interaction):
        # item embedding, v_i
        pos_items_emb = self.moe_adaptor(interaction['pos_item_emb'])
        pos_items_emb = F.normalize(pos_items_emb, dim=1)
        # calculate the logits between training instance and positive instance
        pos_logits = (seq_output * pos_items_emb).sum(dim=1, keepdim=True) / self.temperature
```

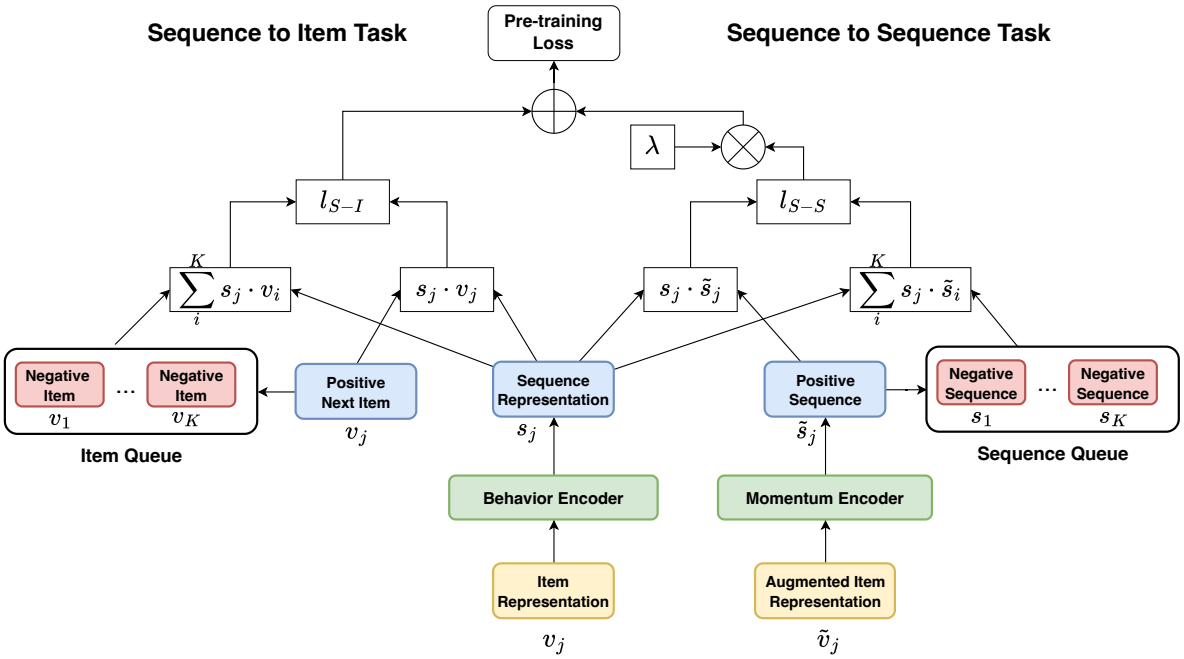

Figure 5: Pre-training Stage

```
        pos_logits = torch.exp(pos_logits)
        # calculate the logits between training instance and negative instance
        neg_logits = torch.matmul(seq_output, self.item_queue.clone().cuda().detach()) /
            self.temperature
        neg_logits = torch.exp(neg_logits).sum(dim=1)
        # calculate the loss
        loss = -torch.log(pos_logits / neg_logits)
        # insert the keys into queue
        self._item_dequeue_and_enqueue(pos_items_emb)
        return loss.mean()
```

In sequence to sequence task, we insert each augmented sequence representation into the sequence queue, the loss function is:

$$l_{S-S} = \sum_{j=0}^{B} [-\log \frac{\exp(s_j \cdot \tilde{s}_j)}{\sum_{i=0}^{K} \exp(s_j \cdot s_i)}] \tag{30}$$

The code is shown as following:

```
def seq_seq_moco(self, q_seq_output, interaction):
        # augmented item representation
        item_seq_aug = interaction[self.ITEM_SEQ + '_aug']
        item_seq_len_aug = interaction[self.ITEM_SEQ_LEN + '_aug']
        item_emb_list_aug = self.moe_adaptor(interaction['item_emb_list_aug'])
        # calculate the positive instance representation, (B, D), B is Batch Size and D is
            the output dim of transformer encoder
        k_seq_output = self.key_forward(item_seq_aug, item_emb_list_aug, item_seq_len_aug)
        k_seq_output = F.normalize(k_seq_output, dim=1)
        # calculate the logits between training and positive instance, (B, D)*(B, D) -> (B)
        pos_logits = (q_seq_output * k_seq_output).sum(dim=1, keepdim=True) / self.temperature
        pos_logits = torch.exp(pos_logits)
        # calculate the logits between training and negative instance, (B, D)*(D, K) -> (B, K)
        neg_logits = torch.matmul(q_seq_output, self.queue.clone().cuda().detach()) /
            self.temperature
```

```
        neg_logits = torch.exp(neg_logits).sum(dim=1)
        # calculate the loss
        loss = -torch.log(pos_logits / neg_logits)
        # insert keys into queue
        self._seq_dequeue_and_enqueue(k_seq_output)
        return loss.mean()
```

4 Evaluation

4.1 Experimental Setup

In terms of experimental settings, we adopted the experimental settings that are the same as those in the original paper and conducted experiments on cross-domain and cross-platform, respectively.

4.1.1 Datasets

We adopted the same settings as the original paper. Datasets are collected for three proposes:

1. Datasets for pre-train: similar to the original paper, we choose five categories from Amazon review datasets, "Grocery and Gourmet Food", "Home and Kitchen", "CDs and Vinyl", "Kindle Store" and "Movies and TV".

2. Datasets for cross-domain: similarly, we use another five categories from Amazon review datasets, which are "Prime Pantry", "Industrial and Scientific", "Musical Instruments", "Arts, Crafts and Sewing" and "Office Products".

3. Datasets for cross-platform: in order to evaluate the cross-platform performance of proposed method, we select an Online Retail transaction dataset from a UK-based online retail platform.

4.1.2 Compared Methods

Due to the limitation of project time and experimental device, we focused the experiment on the comparison with the experimental results of the method proposed in the original paper, so we only compared the experimental data of the UniSRec method in the original paper. In the original paper, the author first pre-trained a general sequence expression model on five Pre-trained datasets and then used two different settings, Inductive and Transductive, to fine-tune (respectively by $UniSRec_t$ and $UniSRec_{t+ID}$ express).

4.1.3 Evaluation Settings

In order to compare the experimental results more effectively, we used the same evaluation settings as the original paper, using Recall@N and NDCG@N, where N is set to 10 and 50. More specifically, we also used a leave-one-out strategy in the evaluation. That is, the last item in the user interaction sequence is used as test data, the penultimate item is used as evaluation data, and all remaining interaction records are used as training data. We calculate the average score of test users by ranking the ground-truth item of each sequence on the test set.

4.1.4 Implementation Details

In order to truly simulate the training results in the paper, we adopted almost the same training settings as the original paper. To prevent overfitting, We used the Adam optimizer and stops at early stage with the patience of 10 epochs, and NDCG@10 is set as the indicator. Like the original paper, the learning rate is set to 0.0003, 0.001, 0.003, 0.01, embedding dimension is set to 64, 128, 300.

In the pre-training phase, we found that the original paper sets the batch size to 8192, which requires nearly 66.37GB of video memory. Since our experimental device cannot provide such a vast video memory requirement, we use the pre-trained universal sequence representation model provided by the original paper to fine-tune. In the subsequent improvement, we used the MoCo framework for pre-training, and the batch size only needed to be set to 256. We pre-trained the proposed method for

Table 1: Performance comparison of proposed method and our reproduced one

Scenario	Dataset	Metric	$UniSRec_t$	$UniSRec_{t+ID}$	$UniSRec_t^{CLR}$	$UniSRec_{t+ID}^{CLR}$
Cross-Domain	Scientific	Recall@10	0.1188	0.1235	0.1167	0.1189
		NDCG@10	0.0641	0.0634	0.0635	0.0649
		Recall@50	0.2394	0.2473	0.2355	0.2405
		NDCG@50	0.0903	0.0904	0.0895	0.0914
	Pantry	Recall@10	0.0636	0.0693	0.071	0.0698
		NDCG@10	0.0306	0.0311	0.0331	0.0328
		Recall@50	0.1658	0.1827	0.1843	0.1838
		NDCG@50	0.0527	0.0556	0.0576	0.0573
	Instruments	Recall@10	0.1189	0.1267	0.1241	0.1252
		NDCG@10	0.0680	0.0748	0.0703	0.0706
		Recall@50	0.2255	0.2387	0.2367	0.2394
		NDCG@50	0.0912	0.0991	0.0947	0.0953
	Arts	Recall@10	0.1066	0.1239	0.1023	0.1241
		NDCG@10	0.0586	0.0712	0.0564	0.0679
		Recall@50	0.2049	0.2347	0.2000	0.2339
		NDCG@50	0.0799	0.0955	0.0776	0.0919
	Office	Recall@10	0.1013	0.1280	0.1052	0.1277
		NDCG@10	0.0619	0.0831	0.0633	0.0818
		Recall@50	0.1702	0.2016	0.1760	0.2025
		NDCG@50	0.0769	0.0991	0.0787	0.0981
Cross-Platform	Online Retail	Recall@10	0.1449	0.1537	0.1481	0.1553
		NDCG@10	0.0677	0.0724	0.0705	0.0727
		Recall@50	0.3604	0.3885	0.3714	0.3877
		NDCG@50	0.1149	0.1239	0.1194	0.1236

100 epochs with $\lambda = 1e^{-3}$ and $G = 8$ experts. In the finetune phase, we used the same experimental settings as the original paper, setting the batch size to 2048.

4.2 Results

In this subsection, we first use the pre-trained universal sequence representation model provided by the paper to fine-tune the datasets, and compare the resultswith the experimental data in the paper. Then we used the MoCo framework to pre-train the model to get a new pre-trained model and performed a fine-tune again. And then we compare the result with the one pretraining on simCLR framework. Finally we came to our conclusion.

4.2.1 Fine-tune comparison

In order to verify the authenticity of the data in the paper, we first used the same pre-trained universal sequence representation model as the paper to perform finetune on five cross-domain datasets and one cross-platform dataset. The results are shown in Table 1.

By comparison, we can see our reproduced method (represented by $UniSRec_t^{CLR}$ and $UniSRec_{t+ID}^{CLR}$) and the experimental data of the original paper (represented by $UniSRec_t$ and $UniSRec_{t+ID}$) has almost no significant difference, which is less than 12%. Among them, the model we reproduced on the "Prime Pantry" dataset performed better than the original paper. $UniSRec_t^{CLR}$ has an improvement of 11.64 % in the Recall@10 compared to $UniSRec_t$. It is clear that the method reproduced in this paper can be effectively transferred to different fields and platforms through universal sequence representation pre-training, and achieved excellent results.

4.2.2 Pre-train comparison

In order to more completely simulate and evaluate the experimental results of the paper, we tried to conduct a complete pre-training based on the experimental settings of the original paper. However, the original paper used simCLR for contrastive learning during pre-training, and the batch size was set to 8192, which required nearly 66.37GB of video memory, and our experimental device could not meet this requirement. Therefore, we used the MoCo framework instead of the simCLR framework for pre-training, which enabled us to pre-train with a batch size of 256 on our experimental device. We

Table 2: Performance comparison of pre-trainning on simCLR and MoCo

Scenario	Dataset	Metric	$UniSRec_t^{CLR}$	$UniSRec_{t+ID}^{CLR}$	$UniSRec_t^{MoCo}$	$UniSRec_{t+ID}^{MoCo}$
Cross-Domain	Scientific	Recall@10	0.1167	0.1189	0.1092	0.1139
		NDCG@10	0.0635	0.0649	0.0597	0.059
		Recall@50	0.2355	0.2405	0.2301	0.238
		NDCG@50	0.0895	0.0914	0.0835	0.0836
	Pantry	Recall@10	0.071	0.0698	0.0653	0.071
		NDCG@10	0.0331	0.0328	0.0281	0.0286
		Recall@50	0.1843	0.1838	0.1756	0.1925
		NDCG@50	0.0576	0.0573	0.0454	0.0483
	Instruments	Recall@10	0.1241	0.1252	0.1189	0.1267
		NDCG@10	0.0703	0.0706	0.0643	0.0711
		Recall@50	0.2367	0.2394	0.2305	0.2437
		NDCG@50	0.0947	0.0953	0.0896	0.0975
	Arts	Recall@10	0.1023	0.1241	0.1012	0.1185
		NDCG@10	0.0564	0.0679	0.0532	0.0658
		Recall@50	0.2000	0.2339	0.1978	0.2276
		NDCG@50	0.0776	0.0919	0.0742	0.0898
	Office	Recall@10	0.1052	0.1277	0.0951	0.1218
		NDCG@10	0.0633	0.0818	0.0552	0.0764
		Recall@50	0.1760	0.2025	0.1667	0.1981
		NDCG@50	0.0787	0.0981	0.0729	0.0951
Cross-Platform	Online Retail	Recall@10	0.1481	0.1553	0.1398	0.1486
		NDCG@10	0.0705	0.0727	0.0621	0.0668
		Recall@50	0.3714	0.3877	0.3543	0.3824
		NDCG@50	0.1194	0.1236	0.1089	0.1179

used the pre-trained universal sequence representation model provided by the original paper and the universal sequence representation model trained through the MoCo framework to perform finetune on five cross-domain datasets and one cross-platform dataset respectively. The results are shown in Table 2.

We also fine-tune the pre-trained universal sequence representation model obtained through the MoCo framework in Inductive and Transductive ways on the same datasets, and the results are represented by $UniSRec_t^{MoCo}$ and $UniSRec_{t+ID}respectively^{MoCo}$ respectively. Because the batch size is small, each epoch in pre-training takes about 3 hours, and it will take 900 hours to complete a pre-training. The training setting in the original text is that when the loss does not decrease within 10 epochs, the training stopped. The maximum value of the epoch is 300, that is, the threshold is 300. In the MOCO framework, we adjust the threshold to 100, which is one-third of the pre-training settings in the original paper. We can see that the performance after 100-epoch threshold of pre-training using the MoCo framework is about 2.84 % to 13.21 % lower than the performance of training 300 epochs using the simCLR framework, but still achieved a relatively good result. Among them, $UniSRec_{t+ID}^{MoCo}$ even performed better than $UniSRec_{t+ID}^{CLR}$ on the "Musical Instruments" dataset, with about 1.8 % improvement on the Recall@50. The experimental results show that using the MoCo framework is a lighter pre-training solution, which can guarantee the normal performance of the model while ensuring that it occupies less computing resources.

5 Conclusion

In this project, we aim to implement the model called UniSRec, which is a unversal sequence representation model and can recommend the item according to the behavioral pattern in multiple domains. In the paper, the training dataset can be viewed as the sequential interactions consist of items, and each items is identified with the text description. To generate the item representation in semantic space, the paper utilizes the BERT model to generate ID-agnostic item representations. But since the training data comes from multiple domains, there will be anisotropy in BERT output embeddings. To alleviate this problem, the paper utilizes parameters whiting to set both mean vector and covariance matrix to learnable parameters. Considering that a big senmantic gap exists between different domains, the Mixture of Experts is introduced to generate universal item representation. Then the paper utilizes the transformer encoder and concatenate the sequential item embedding with

a special token as the input to generate the universal sequence (behavior pattern) representation.

To better enhance infusion and adaptation of sequence representation in multiple domains, at the pre-training stages, two contrastive learning tasks is proposed in paper, which is similar the simCLR. But this training framework requires a enormous batch size which can only be afforded by organizations or laboratories, we could not run the pre-training under this batch setting, and also the framework requires at least 4k batch size to make the training effective, it is meaningless for us to reduce the batch size to a small number. So in pre-training stage, we replace the training framework with the one in MoCo, which requires only 256 batches. At the fine-tune stages training, which is used to adapt the model to a new domain, we implement the stetting in paper.

In evaluation, we set up two main experiments, the first one is to directly use the parameters of pre-trained model given by paper and run the fine-tune training. The second one is to run the pre-training by MoCo framework. In the fine-tuning experiment of reproducing the paper, we used the same hyperparameter settings as the pre-trained model in the original paper, and the final results were almost the same as those shown in the paper. After being improved by the MoCo framework, the fine-tuning experiment based on the MOCO pre-training model showed slightly lower performance than the original paper, which is already a satisfactory result under the premise of being limited by the epoch threshold and training time.

In summary, UniSRec can generate universal sequence representation effectively, but its pre-training stage consumes huge computing resources which is hard to be satisfied, so we optimize it with a more batch-friendly training framework and implement other parts in the model.

6 Contribution

- CHANG, Zijiang
  In our teamwork, I was responsible for obtaining items' representation from text description of the them. Firstly, I learned the mechanism, structure, input and output of the pretrain language model BERT. With the help of the framework called recbole,I completed and run the code to got the output of mark [CLS]. Then I read some literature to know the anisotropy of original output of BERT and several approaches to handle this problem.I understand how parameter whitening module works in this model and put some key points to the report. Next,I learn the technique of MoE used in complex prediction tasks. Experts and gating router are two core concepts of MoE.I treat each parameter whitening module as a expert and finally get the textual representation of items.

- ZHONG, Lintao
  Throughout the project, I was mainly responsible for the implementation of the behavior encoder of the model and two fine-tune methods, and the integration and analysis of experimental data. In the code part, I mainly cooperate with YAN Xi to complete the implementation of the sequence representation part. More specifically, I apply multi-head attention to the transformer encoder and integrate it into the model's behavior encoder. For the two fine-tune methods proposed in the paper, I provided code implementations and integrated them into the model. In the proposal part, I was mainly responsible for the collection and integration of papers. In the final report part, I was mainly responsible for the Introduction and Evaluation parts, and also completed the algorithm definition and formula of the behavior encoder part.

- WANG, Jiatong
  In this teamwork, I mainly take the role of model construction integration and experimentation, that is, communicate with the three teammates about the corresponding modules and complete the code together. In the project, I discussed the sequence encoder process with ZhongLintao, communicated with ChangZijiang to learn the realization of the whitening layer and Moe module, asked YanXi for the content of comparative learning, and discussed the motivation and feasibility of model migration. The codes of the three students were integrated and debugged by me, and I also contributed some codes to the model. In the experimental part, I was responsible for the operation of the entire experiment, and the recording results and analysis were handed over to the rest of the students for summary. In the thesis part, I was mainly responsible for the Implementation and Conclusion parts.

- YAN, Xi
  In this project, I was mainly responsible for the research and implementation of pre-training stage, and after reading some references and papers about contrastive learning and simCLR, I understand why to utilize contrastive task for pre-training and found that the batch-size in original framework is unrealistic, and inspired by MoCo, so I propose to replace the training framework with MoCo and implement MoCo framework, and discuss the feasibility with other group members. In the proposal part, I took the role of researching the algorithm mentioned in the paper. In the final report part, I was mainly responsible for Training Process and MoCo optimization in Implementation, besides, I was responsible for Latex writing, integrating the parts written by each members to the report, checking for syntax errors and spell errors, and adjusting the format.

References

[1] Kang W C, McAuley J. Self-attentive sequential recommendation[C]//2018 IEEE international conference on data mining (ICDM). IEEE, 2018: 197-206.

[2] Chen T, Kornblith S, Norouzi M, et al. A simple framework for contrastive learning of visual representations[C]//International conference on machine learning. PMLR, 2020: 1597-1607.

[3] He K, Fan H, Wu Y, et al. Momentum contrast for unsupervised visual representation learning[C]//Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2020: 9729-9738.

[4] Chen X, Fan H, Girshick R, et al. Improved baselines with momentum contrastive learning[J]. arXiv preprint arXiv:2003.04297, 2020.