CS 330 Autumn 2021/2022 Homework 0

Multitask Training for Recommender Systems
Due Monday September 27, 11:59 PM PST

SUNet ID:
Name:
Collaborators:

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

1 Overview

In this assignment, we will implement a multi-task movie recommender system based on the classic Matrix Factorization [1] and Neural Collaborative Filtering [2] algorithms. In particular, we will build a model based on the BellKor solution to the Netflix Grand Prize challenge and extend it to predict both likely user-movie interactions and potential scores. In this assignment you will:

1. Implement a multi-task neural network architecture and explore the effect of parameter sharing on model performance.
2. Debug the multi-task training pipeline.

The main goal of these exercises is to familiarize yourself with multi-task architectures, the training pipeline, and coding in PyTorch. These skills will be important in the course.

Submission: To submit your work, submit one pdf report and one zip file to GradeScope, where the report contains answers to the deliverables listed below and the zip file contains the code with your filled-in solutions.

Code Overview: The code consists of several files; however, you will only need to modify three:

- `main.py`: To run experiments, execute this file by passing the corresponding parameters.
- `models.py`: This file contains our multi-task prediction model **MultiTaskNet**, which you will need to finish implementing in PyTorch.
- `multitask.py`: This file contains the training pipeline. We have introduced an error in the **fit** function of the **MultitaskModel** class, which you will need to debug.

2 Dataset and Evaluation

Dataset. In this assignment, we will use movie reviews from the MovieLense dataset. The dataset consists of 100K reviews of 1700 movies generated by 1000 users. Although each user interaction contains several levels of meta-data, we'll only consider tuples of the type (**userID, movieID, rating**), which contain an anonymized user ID, movie ID and the score assigned by the user to the movie from 1 to 5. We randomly split the dataset into a **train** dataset, which contains 95% of all ratings, and a **test** dataset, which contains the remaining 5%.

Problem Definition. Given the dataset defined above, we would like to train a model $f(\text{userID}, \text{movieID})$ that predicts: 1) the probability p that the user would watch the movie and 2) the score r they would assign to it from 1 to 5. For some intuition on this setting, consider a user who only watches comedy and action movies. It would not make sense to recommend them a horror movie since they don't watch those. At the same time, we would want to recommend comedy or action movies that the user is likely to score highly.

Evaluation. Once we have our trained model, we evaluate it on the test set.

Score Prediction. We will evaluate the mean-squared error of movie score prediction on the held-out user ratings, i.e. $\frac{1}{N}\sum_{i=1}^{N}||\hat{r}_i - r_i||^2$, where $\hat{r}_i$ is the predicted score for user-movie pair $(\text{userID}_i, \text{itemID}_i)$. The summation is over all pairs in the test set. Better models achieve lower mean-squared errors.

Likelihood Prediction. By definition, our dataset contains ratings for movies the users have seen. To evaluate the quality of the likelihood model, we use the mean reciprocal rank metric. The metric is computed as follows: 1) for each user, rank all movies based on the probability that the user would watch them; 2) remove movies we know the user has watched (those in the training set); 3) compute the average reciprocal ranking of movies the user has watched from the held-out set. In mathematical terms:

$$\text{MRR} = \frac{1}{N_{\text{users}}}\sum_{i=1}^{N_{\text{users}}}\frac{1}{N_i}\sum_{j=1}^{N_i}\frac{1}{\text{rank}_{i,j}} \tag{1}$$

where N_{users} is the number of users in the dataset, N_i is the number of held-out movies user userID_i has seen and $\text{rank}_{i,j}$ is the rank of movie itemID_j for user userID_i given by our model. For example, consider the dataset in the following table:

userID	Held-out Movies	Model Ranking All	Mean Reciprocal Rank
1	A, B	B, C, D, A, G	$\frac{1}{2}\left[\frac{1}{4}+\frac{1}{1}\right]$
2	B	A, B, C, G, D	$\frac{1}{1}\left[\frac{1}{2}\right]$
3	A, B, C	A, D, B, C, G	$\frac{1}{3}\left[\frac{1}{1}+\frac{1}{3}+\frac{1}{4}\right]$

The MRR for the batch is 0.55. Better models achiever higher MRR scores.

3 Problems

Please read both problems before you begin. To install all required packages for this assignment you can run `pip install -r requirements.txt`.

3.1 Problem 1: Implement Multi-Task Model

In this problem, we will implement a multi-task model using Matrix Factorization [1] and regression-based modelling:

Matrix Factorization: Consider an interaction matrix M, where $M_{ij} = 1$ if userID$_i$ has rated movie with itemID$_j$ and 0 otherwise. We will represent each user with a latent vector $\mathbf{u}_i \in \mathbb{R}^d$ and each item with a latent vector $\mathbf{q}_i \in \mathbb{R}^d$. We model the interaction probability $p_{ij} = \log P(M_{ij} = 1)$ in the following way:

$$p_{ij} = \mathbf{u}_i^T \mathbf{q}_j + a_i + b_j \tag{2}$$

where a_i is a user-specific bias term and b_j is a movie-specific bias term. At each training step we sample a batch of triples (userID$_i$, itemID$_j^+$, itemID$_{j'}^-$) with size B, such that $M_{i,j} = 1$, while itemID$_{j'}^-$ is randomly sampled (indicating no user preference). Let

$$\begin{aligned} p_{ij}^+ &= \mathbf{u}_i^T \mathbf{q}_j + a_i + b_j \\ p_{ij'}^- &= \mathbf{u}_i^T \mathbf{q}_{j'} + a_i + b_{j'} \end{aligned} \tag{3}$$

and optimize the Bayesian Personalised Ranking (BPR) [3] pairwise loss function:

$$\mathcal{L}_F(\mathbf{p}^+, \mathbf{p}^-) = \frac{1}{B} \sum_{i=1}^{B} 1 - \sigma(p_{ij}^+ - p_{ij'}^-) \tag{4}$$

where σ is the sigmoid function.

Regression Model: For training the regression model, we consider only batches of tuples (userID$_i$, itemID$_j^+$, r_{ij}), such that $M_{i,j} = 1$ and r_{ij} is the numerical rating userID$_i$ assigned to itemID$_j^+$. Using the same latent vector representations as before, we will concatenate $[\mathbf{u}_i, \mathbf{q}_j, \mathbf{u}_i * \mathbf{q}_j]$ (where $*$ denotes element-wise multiplication) together and pass it through a neural network with a single hidden layer:

$$\hat{r}_{ij} = f_\theta([\mathbf{u}_i, \mathbf{q}_j, \mathbf{u}_i * \mathbf{q}_j]) \tag{5}$$

We train the model using the mean-squared error loss:

$$\mathcal{L}_R(\hat{\mathbf{r}}, \mathbf{r}) = \frac{1}{B} \sum_{i=1}^{B} ||\hat{r}_{ij} - r_{ij}||^2 \tag{6}$$

Your Implementation: The first part of the assignment is to implement the above model in `models.py`. First you need to define each component when the model is initialized.

1. Consider the matrix $\mathbf{U} = [\mathbf{u}_1|, \ldots, |\mathbf{u}_{N_{\text{users}}}] \in \mathbb{R}^{N_{\text{users}} \times d}$, $\mathbf{Q} = [\mathbf{q}_1|, \ldots, |\mathbf{q}_{N_{\text{items}}}] \in \mathbb{R}^{N_{\text{items}} \times d}$, $\mathbf{A} = [a_1, \ldots, a_{N_{\text{users}}}] \in \mathbb{R}^{N_{\text{users}} \times 1}$, $\mathbf{B} = [b_1, \ldots, b_{N_{\text{items}}}] \in \mathbb{R}^{N_{\text{items}} \times 1}$. Implement $\mathbf{U}$ and $\mathbf{Q}$ as `ScaledEmbedding` layers with parameter $d =$ `embedding_dim` and $\mathbf{A}$ and $\mathbf{B}$ as `ZeroEmbedding` layers with parameter $d = 1$ (defined in `models.py`). These are instances of PyTorch Embedding layers with a different weight initialization, which facilitates better convergence.

2. Next implement $f_\theta([\mathbf{u}_i, \mathbf{q}_j, \mathbf{u}_i * \mathbf{q}_j])$ as an MLP network. The class `MultiTaskNet` has `layer_sizes` argument, which is a list of the input shapes of each dense layer. Notice that by default `embedding_dim=32`, while the input size of the first layer is 96, since we concatenate $[\mathbf{u}_i, \mathbf{q}_j, \mathbf{u}_i * \mathbf{q}_j]$ before processing it through the network. Each layer should be followed by a ReLU activation. The final layer should output the final user-item predicted score in and have an output size of 1.

3. The `MultiTaskNet` class has an `embedding_sharing` attribute. Implement your model in such a way that when `embedding_sharing=True` a single latent vector representation is used for both the factorization and regression tasks and vice versa.

In the second part of the problem you need to implement the `forward` method of the `MultitaskNet` module. The `forward` method receives a batch of $(\text{userID}_i, \text{itemID}_j)$ of user-item pairs. The model should output a probability p_{ij} that the user would watch the movie, given by Eq. 2 and a predicted score $\hat{r}_{ij}$ the user would assign to the movie, given by Eq. 5. **Be careful with the output tensor shapes!**

3.2 Problem 2: Debug Model Training

The main algorithm training loop is implemented in `multitask.py`. Given a batch of tuples $(\text{userID}_i, \text{itemID}_j^+, \text{itemID}_{j'}^-, r_{ij})$, such that $M_{i,j} = 1$ and $M_{i,j'} = 0$, we user our model to predict $p_{ij}^+, p_{ij'}^-, \hat{r}_{ij}$ as defined in Eq. 3 and Eq. 6. We optimize a weighted joint loss

$$\min_\theta \lambda_F \mathcal{L}_F(\mathbf{p}^+, \mathbf{p}^-) + \lambda_R \mathcal{L}_R(\hat{\mathbf{r}}, \mathbf{r}) \tag{7}$$

where $\mathcal{L}_F$ is as defined in Eq. 4 and $\mathcal{L}_R$ in Eq. 6. We have introduced a deliberate bug in the loss function implementations in the `fit` method of `multitask.py`, which you have to debug. In your work, consider the model outputs and specific loss implementations above. It might be helpful to check the loss function implementations in `losses.py`, but you should not modify those.

4 Write-up

To execute experiments run the `main.py` script, which will automatically log training MSE loss, BPR loss and test set MSE loss and MRR scores to TensorBoard. Once you're done with your implementation run the following experiments:

1. Evaluate a model with shared representations and task weights $\lambda_F = 0.99, \lambda_R = 0.01$. You can run this experiment by running:

   ```
   python main.py --factorization_weight 0.99 --regression_weight 0.01
   --logdir run/shared=True_LF=0.99_LR=0.01
   ```

   Here the `--factorization_weight` and `--regression_weight` arguments correspond to λ_F and λ_R respectively.

2. Evaluate a model with shared representations and task weights $\lambda_F = 0.5, \lambda_R = 0.5$. You can run this experiment by running:

   ```
   python main.py --factorization_weight 0.5 --regression_weight 0.5
   --logdir run/shared=True_LF=0.5_LR=0.5
   ```

3. Evaluate a model with **separate** representations and task weights $\lambda_F = 0.5, \lambda_R = 0.5$. You can run this experiment by running:

   ```
   python main.py --no_shared_embeddings --factorization_weight 0.5
   --regression_weight 0.5 --logdir run/shared=False_LF=0.5_LR=0.5
   ```

For each experiment include a screenshot of Tensorboard graphs for the training and test set losses in your write up. Answer the following questions:

1. Does parameter sharing outperform having separate models? By comparing train and test set performance can you briefly explain these results?

2. In the shared model setting compare results for $\lambda_F = 0.99$ and $\lambda_R = 0.01$ and $\lambda_F = 0.5$ and $\lambda_R = 0.5$, can you explain the difference in performance?

References

[1] Koren Yehuda, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42, 2009.

[2] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[3] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *Conference on Uncertainty in Artificial Intelligence*, 2009.