Recommendation system: Collaborative filtering using Pytorch

By:
Vishwas
Tong

Outline

1 **Model and Features**

Pytorch model architecture and features implemented

2 **Techniques**

- Negative sampling
- Cold-start problem
- Regularization

3 **Experimental results**

Hyperparameter tuning:Learning rate, embedding layer, dropout rate

4 **Lessons learned**

Summary of models, techniques, experiments etc.

Submission 1 **Matrix factorization model**

Random negative sampling and learning the user and item embeddings

Submission 2 **Neural network model**

Added linear and dropout layer on top of embedding layer for users and items

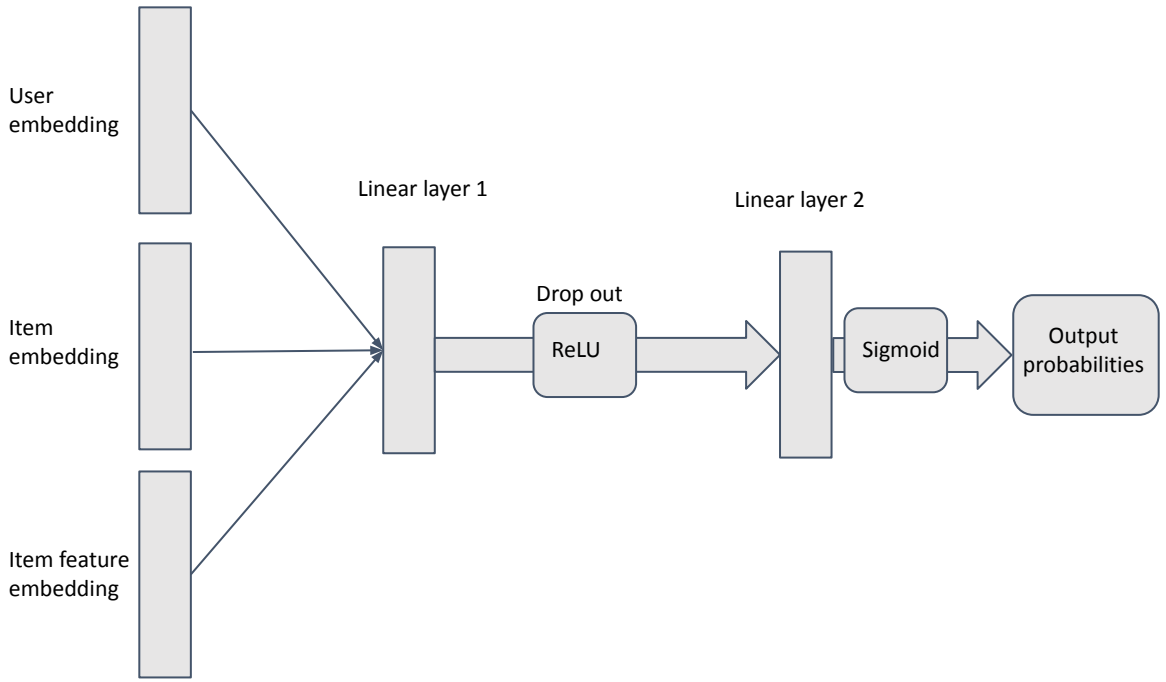
Submission 3 **Neural network model with popularity based sampling**

- Cold start problem tackled
- Probability based item ID selection for negative sampling

Model and features:

- User embedding layer and item embedding layer to learn the user and item interactions.
- Incorporation of item feature ID embedding layer in the model.
- These embedding layers were followed by a linear layer and Relu activation function.
- We incorporated a dropout layer followed by another linear layer before applying the sigmoid to get the final probability from the model output.
- Binary cross entropy was used for evaluating model loss.

Model architecture for submission 3:



Techniques

- Negative sampling based on item popularity and number of items a user has interacted with.
  - The inverse of popularity is used as probabilities for sampling items
  - The number of negative samples is based on number of items a particular user has interacted.
- Random user ID dropout to learn the new user embedding that the model can train on to better handle cold start users.

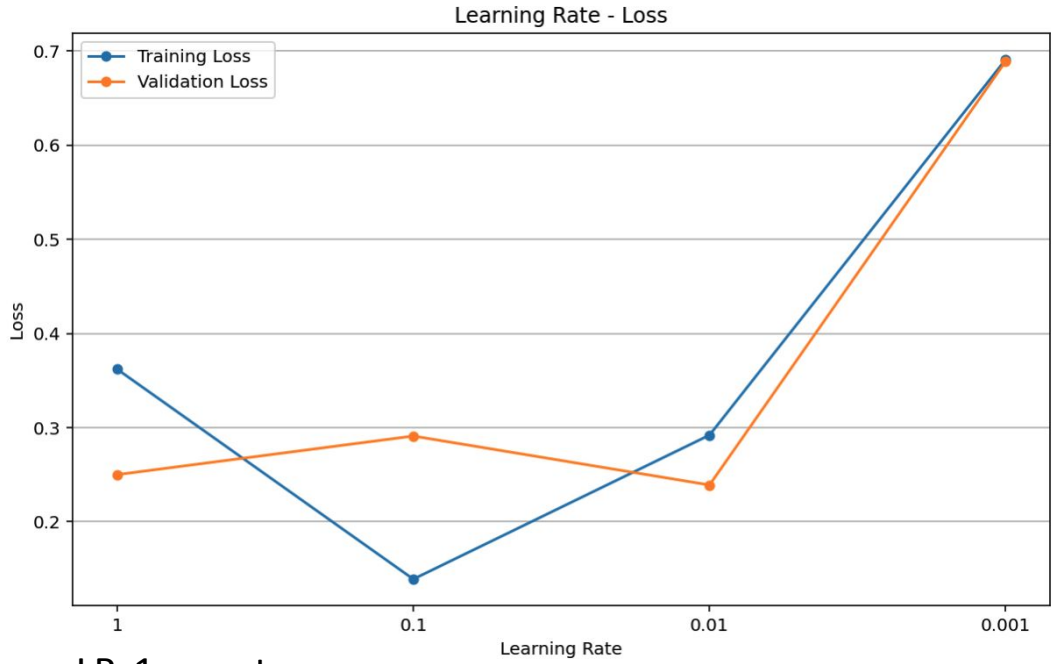
- Dropout layer in the model to regularize the model parameters and make it more generalizable

- Dataloader implemented to introduce batch wise gradient descent which creates shuffled batches.

Experimental Results

- Hyperparameter search:
  - **Learning Rate** (embedding size 3, dropout rate 0.5)
  - Embedding Size
  - Dropout Rate

Learning Rate	Training Loss	Validation Loss
1	0.362	0.250
0.1	0.139	0.291
0.01	0.292	0.239
0.001	0.691	0.689

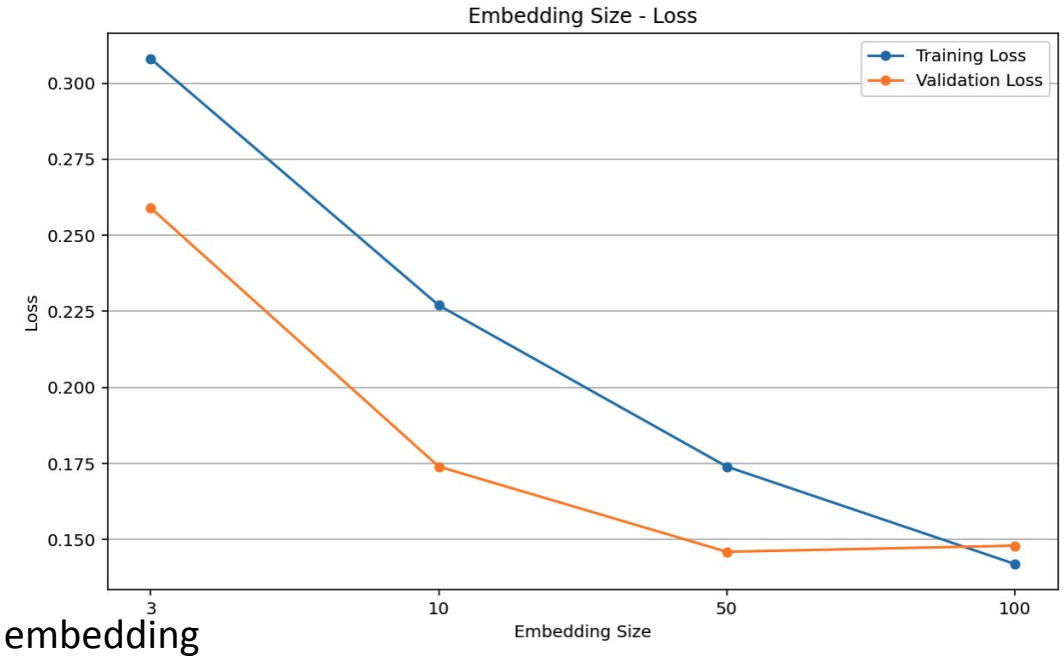


Observations: Considering both training loss and validation loss, LR 1 was too large, LR 0.1 lead to overfitting, and LR 0.001 was too small to converge to minimum loss. Therefore, we chose **0.01** as the **Learning Rate** in our experiments.

Experimental Results

- Hyperparameter search:
  - Learning Rate
  - **Embedding Size** (learning rate 0.01, dropout rate 0.5)
  - Dropout Rate

Embedding Size	Training Loss	Validation Loss
3	0.308	0.259
10	0.227	0.174
50	0.174	0.146
100	0.142	0.148

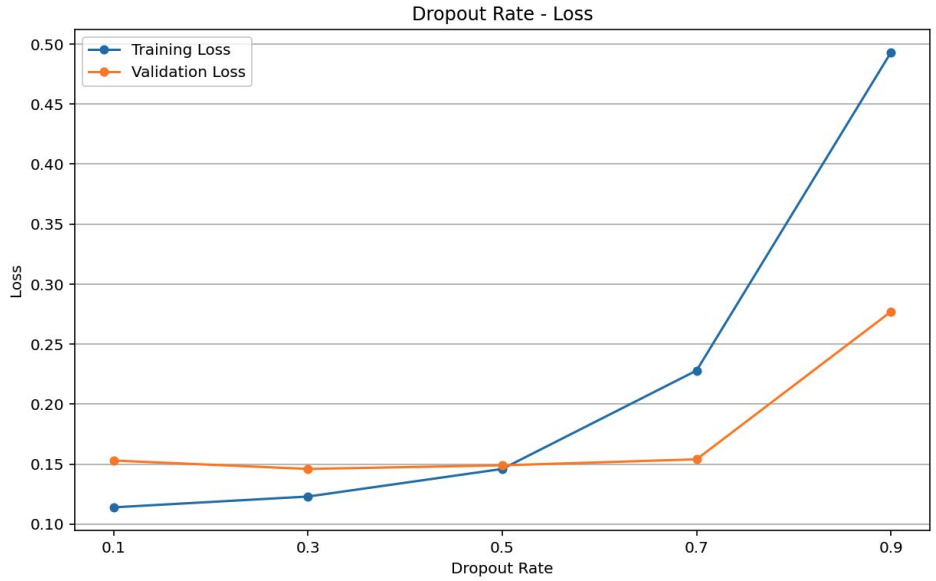


Observations: Considering both training loss and validation loss, embedding sizes 50 and 100 seem to converge to similar minimum losses. But increasing embedding size from 50 to 100 did not significantly increase the model performance. Therefore, we choose **50** as the **embedding size**.

Experimental Results

- Hyperparameter search:
  - Learning Rate
  - Embedding Size
  - **Dropout Rate** (learning rate 0.01, embedding size 50)

Dropout Rate	Training Loss	Validation Loss
0.1	0.114	0.153
0.3	0.123	0.146
0.5	0.146	0.149
0.7	0.228	0.154
0.9	0.493	0.277



Observations: Dropout rate 0.9 and 0.7 did not fit the training data well compared to the other candidate dropout rates. With DR 0.1, the model fits the training data better than other candidates However, its associated the validation loss was not better than DR 0.5 or 0.3. We also exclude DR 0.3 to avoid the risk of overfitting. In conclusion, we selected **0.5** as our **dropout rat**e.

Lessons Learned

- Implementing a Pytorch based neural network model for creating a recommendation system based on implicit feedback.
- Implementing Dataloader helps in getting a more generalizable model because of data shuffle in each batch.
- Adding more linear layers on top of user embeddings and item embeddings improve the performance of the models.
- Large learning rate may lead to oscillations affecting the validation scores. Small learning rate could lead to failure in converging to minimum loss.
- Increasing embedding sizes leads to higher computation time.
- Negative sampling technique affects the model performance.
- Regularization techniques like dropout rates can improve model performance.
- Cold start problem can be tackled by training the model on user ID in the training randomly assigned as new users to train the embedding for new user.