CSC311 Project Report

Ruochen Zhang

Dazhi Yuan

Changyan Xu

Department Of Computer Science

University of Toronto

December 2020

Introduction

In this section, we improve the Matrix Factorization based on Alternating Least Square (ALS) on two aspects. On the one hand, we want to reduce the run-time, and on the other hand, we need to enhance the algorithm with greater accuracy. Moreover, we suspect that there can be overfitting problem underlining the current algorithm. Hence, we extend the algorithm with two approaches, one is by adding learning rate schedules, and the other one is to apply L2 regularization to SGD optimization. The developed algorithm can be find mainly in `updated_mf_l2.py` and partially in `updated_mf_lr.py`

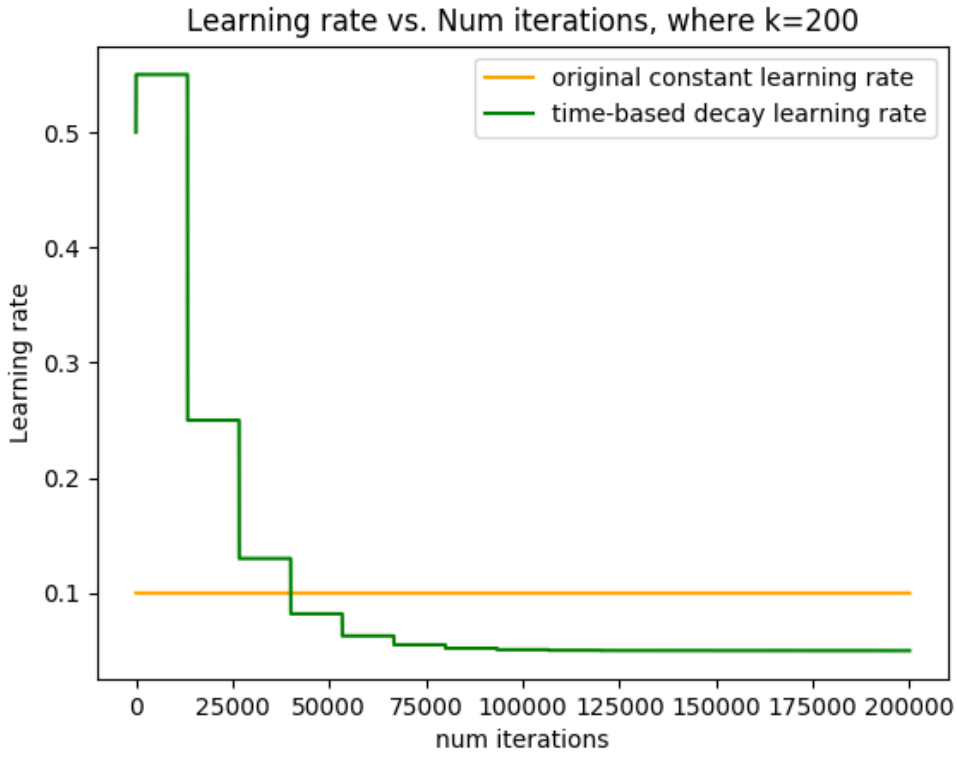
Proposed method 1: Learning Rate Schedules

Specifically, we applied a step decay for the learning rate to march on initially and later proceeded much slower as the U and Z matrices during training tend to stabilize. The mathematical expression supporting is as followed: [1]

$$lr = lr_0 * drop^{epoch/epochs\_drop} + c$$

, where lr is the learning rate, lr_0 is the initial learning rate, $epoch$ is the iteration number, $drop$ is the percentage of the drop, and $epochs\_drop$ is the interval number of iterations indicating it perform a drop after this number of iterations. c is a constant for the learning rate to converge to. See Figure.1a.

With the current training data, the runtime fluctuates around $16s$ to $22s$ for both original ALS and updated ALS. The time cost for the updated ALS with step decay is sometimes faster than the original, but not consistent. We expect that we need more rigorous tests and further adjustment on new hyper-parameters, $drop$ and



(a) Learning rate vs. Number of Iterations

```
===== hyperparameters =====
num_iterations = 200000
constant learning_rate = 0.1
initial learning rate = 0.5
lr decay constant = 5e-10
drop = 0.4
epochs_drop = 11111
chosen_k = 200
Updated ALS
     als_start=1.620153977, als_end=19.783359776, als_duration=18.163205799
     final validation accuracy: 0.6933389782670054
     final test accuracy: 0.6892464013547841
Original ALS
     als_start=19.794423007, als_end=41.11457623, als_duration=21.320153223
     final validation accuracy: 0.6941857183178097
     final test accuracy: 0.6917866215071973
```

(b) Comparison of two ALSs with or without step decay

Figure 1: Step decay for learning rate

epochs_drop. Furthermore, if we train with a larger dataset than the current training matrix with a size of 542×1774, the updated ALS with step decay will achieve a much better runtime than the counterpart. See Figure.1b.

Proposed method 2: L2 Regularization

The original ALS algorithm requires run $200,000$ iterations to reach it optimal performance, which has a total cost of around 20 seconds. After implement L2 regularization upon the current ALS algorithm, we make remarkable achievements on the runtime, which is now only 1 to 2 seconds.

The following way is how L2 Regularization is approached: [3]

As $\mathbf{C}$ is the training matrix of size $n \times m$ or user_id $\times$ question_id, $\widehat{\mathbf{C}}_{nm}$ is the prediction matrix applied with L2 regularization, which can be expressed by

$$\widehat{\mathbf{C}}_{nm} = \mu + b_n + b_m + u_n^T \cdot z_m$$

b_n is a vector of size $n \times 1$, which is the user bias vector, and in this case b_n is the n^{th} entry of b_n. Similarly, b_m is a vector of size $m \times 1$, which is the question bias vector, and in this case b_m is the m^{th} entry of b_m.

The loss function we are using:

$$L = \frac{1}{2}\left[\sum_{n,m} (\mathbf{C}_{nm} - \widehat{\mathbf{C}}_{nm})^2 + \lambda_{ub} \sum_n ||b_n||^2 + \lambda_{zb} \sum_m ||b_m||^2 + \lambda_{nf} \sum_n ||u_n||^2 + \lambda_{mf} \sum_m ||z_m||^2 \right]$$

, where λ_{ub} is the regularization term for user bias, λ_{zb} is the regularization term for question bias, λ_{nf} is the regularization term for user latent factor, λ_{mf} is the regularization term for question latent factor, and they are all hyperparameters.

To apply stochastic gradient descent (SGD), we found the update rules for the gradient descent by applying partial derivatives to the loss function and set fixed point to 0. For example,

$$b_n \leftarrow b_n + \alpha \frac{\partial L}{\partial b_n}$$

where

$$\frac{\partial L}{\partial b_n} = -(\mathbf{C}_{nm} - \widehat{\mathbf{C}}_{nm}) + \lambda_{ub} b_n$$

and α is the learning rate. Hence, we got the following four gradient descent update

rules:

$$
\begin{aligned}
b_n &\leftarrow b_n + \alpha[(\mathbf{C}_{nm} - \widehat{\mathbf{C}}_{nm}) - \lambda_{ub} b_n] \\
b_m &\leftarrow b_m + \alpha[(\mathbf{C}_{nm} - \widehat{\mathbf{C}}_{nm}) - \lambda_{zb} b_m] \\
u_n &\leftarrow u_n + \alpha[(\mathbf{C}_{nm} - \widehat{\mathbf{C}}_{nm}) z_m - \lambda_{uf} u_n] \\
z_m &\leftarrow z_m + \alpha[(\mathbf{C}_{nm} - \widehat{\mathbf{C}}_{nm}) u_n - \lambda_{zf} z_m]
\end{aligned}
\tag{1}
$$

The following is basically how we are training U and Z matrices:

Algorithm 1 *ALS with SGD Optimization applied with L2 Regularization*

Randomly initialize $\mathbf{U}$ and $\mathbf{Z} \longleftarrow \mathbf{U}_{[N\times K]}, \mathbf{Z}_{[M\times K]}$ where $N, M \in \mathcal{R}$.shape
Initialize $\mathbf{b}_{users}, \mathbf{b}_{ques} \longleftarrow$ `zeros`(N), `zeros`(M)
for *i = 0 to i = max_iter* **do**
    `lr = update_lr(lr0,drop,epochs_drop,i)` #step decay learning rate
    Randomly select $(n, m) \in O$
    $train\_C \longleftarrow train\_data[n, m]$
    prediction $\longleftarrow$ `prediction`$(\mathbf{b}_{users}, \mathbf{b}_{ques}, \mathbf{U}[n], \mathbf{Z}[m])$ #*equation* $\widehat{\mathbf{C}}_{nm}$
    error $\longleftarrow$ ($train\_C-$ prediction)
    $\mathbf{b}_{users}, b_{ques}, \mathbf{U}[n], \mathbf{Z}[m]) \longleftarrow$ `als_updates` #*equation* gradient decent update
$user\_bias \leftarrow \mathbf{b}_{users}$`.reshape((N,1))` $\cdot$ `ones((1,M))`
$ques\_bias \leftarrow$ `ones((N,1))` $\cdot \mathbf{b}_{ques}$`.reshape((1,M))`
$global\_bias \leftarrow$ `ones((N,M))` $\times$ *mu* #*hyper_parameter*
$C \longleftarrow global\_bias + ques\_bias + user\_bias + \mathbf{U} \cdot \mathbf{Z}$
return C

The rational that we add the user bias regularization and the question bias regularization inspired by the article is that there can be certain users that have better competence to answer more question correctly, and also there can be certain questions that are particularly difficult to be correctly answered for most users. For example, since the data covers the student users from 7 to 18 years old, primary school questions can then be exceptionally easy for those students from the senior high school. In this case, the high school student users who is capable to answer lower-grade questions, does not mean they can have the same possibility to answer hard questions.

With L2 regularization, we can avoid overfitting and converge much faster. The result we got is in Figure 2.

```
Updated ALS with L2 Regularization and step-decay learning rate
===== hyperparameters =====
num_iterations = 145000
inital learning_rate = 0.3
chosen_k = 380
global bias = 0.5
user bias reg = 0.15
question bias reg = 0.3
user factor reg = 0.4
question factor reg = 0.15
drop = 0.15
epochs_drop = 29000

     als_start=2.491311985, als_end=18.109913536, als_duration=15.618601551000001
     final validation accuracy: 0.7075924357888794
     final test accuracy: 0.7075924357888794



Original ALS
===== hyperparameters =====
num_iterations = 200000
constant learning_rate = 0.1
chosen_k = 200

     als_start=18.12056531, als_end=35.061438191, als_duration=16.940872881
     final validation accuracy: 0.6852949477843635
     final test accuracy: 0.6937623482924076
```

Figure 2: Console results for the Updated ALS with L2 regularization and step-decay learning rate vs. Original ALS. The updated ALS has better accuracy and faster runtime.

Comparison or Demonstration

The following figure 3 is the test accuracy table for all models we implemented. The algorithm we implement for part B has the greatest test accuracy with approx. 70.8% correctness on prediction. See more in *PartA.pdf* enclosed in *code.zip*.

models	test accuracy (3d.p.)
kNN	0.684
Item Response Theory	0.705
SVD	0.650
ALS	0.696
Neural Network	0.691
Ensemble (kNN, IRT, NN)	0.699
Enhanced ALS with L2 Regularization and Step-decay learning rate	0.708

Figure 3: Comparison on test accuracy performance for all models we implement.

Based on previous descriptions, we used L2 regularization for faster convergence and less probablePartA.pdf to overfit the model.

Limitations

Too Many Hyperparameters

Our optimized ALS with L2 regularization and step-decay learning rates is exposed to hyperparameter tuning difficulties. As it can be seen that there is a total 10 hyperparameters in the existing algorithm, it takes long time and meticulous checking to reach the optimal performance for the algorithm itself. Furthermore, even we obtained what we think is the optimal value for a hyperparameter, we cannot guarantee it is definitely the global optimal one rather than only being the local optimal ones. Hence so far, we still believe, if we have more time and energy, it can achieve much higher accuracy and less time cost.

Unreliable Observed Data

If observed data is not useful for training but actually deployed as a part of training, all existing models would fail quietly and inevitably. A general case can be that a user choose wrong answer in a multiple choice question, because this user accidentally wrongly typed, rather than due to mistakenly understanding or poor knowledge. Or in extreme cases, for example, if some user(s) play pranks on the platform by randomly entering the answer rather pondering over them, it can potentially make the dataset invaluable. While the technicians who take the dataset and use as a part of recommendation system construction, the accuracy of the results are then doubtful but oblivious.

Possible Solution

To address the dataset issue, we may need a larger and more comprehensive dataset for training, possibly with a training matrix in the size of Netflix competition dataset for Recommender System. Then the 'accidentally wrongly typed' wrong answers may be learning by the current bias hyperparameters. In addition, we may need write a binary classification system to learn which of those observed data may be created out of malice.

For resolving the hardness of tuning an exhibition of hyperparameters dilemma, we may need to deploy better strategies for finding the optimal values for all the hyperparameters we have. As it is still relying on manual efforts so far, applying Hill climbing and/or Bayesian optimization may be more efficient approaches. [4]

References

[1] Suki Lau, https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1 Online

[2] CS231n: Convolutional Neural Networks for Visual Recognition https://cs231n.github.io/neural-networks-3/update Online

[3] Insight, Explicit Matrix Factorization: ALS, SGD, and All That Jazz https://blog.insightdatascience.com/explicit-matrix-factorization-als-sgd-and-all-that-jazz-b00e4d9b21ea Online

[4] George Lawton, How to optimize hyperparameter tuning for machine learning models https://searchenterpriseai.techtarget.com/feature/How-to-optimize-hyperparameter-tuning-for-machine-learning-models Online