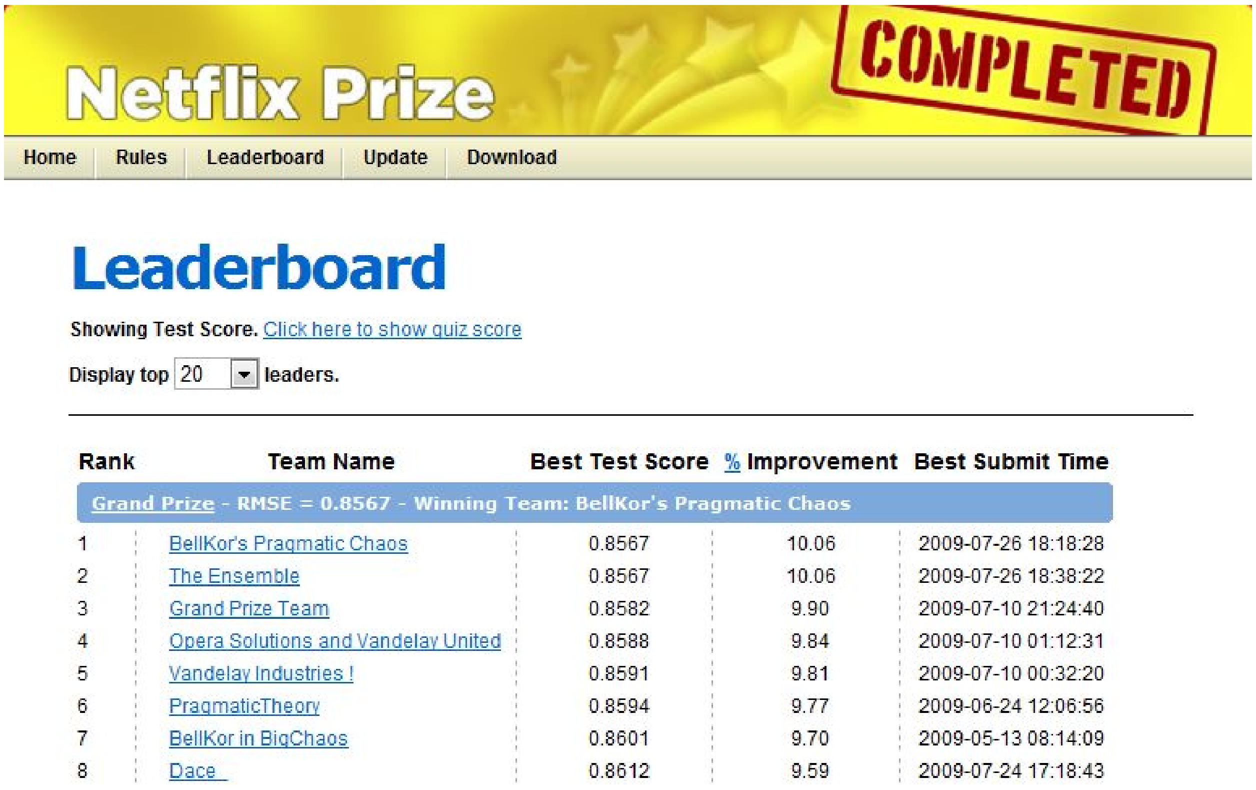
Networked Life Project Report

Netflix Recommendation System

Wang Siyuan 1000896
Loo Yi 1000937



Leaderboard

Showing Test Score. Click here to show quiz score

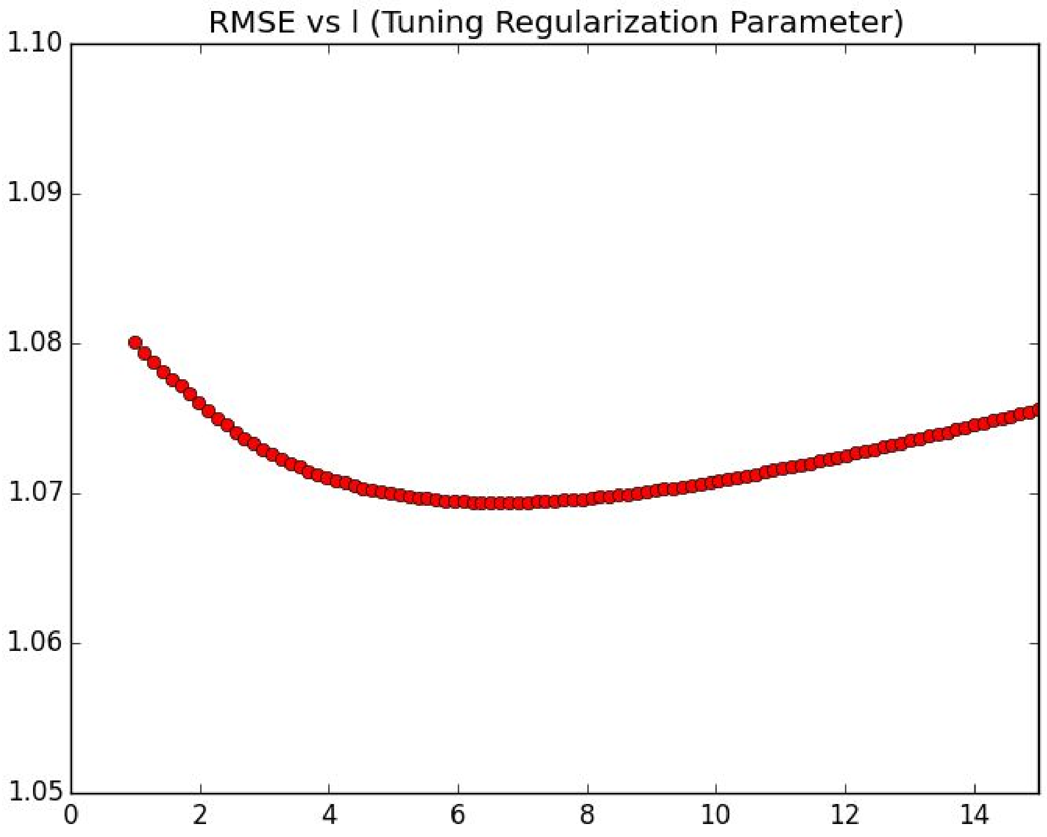
Display top 20 leaders.

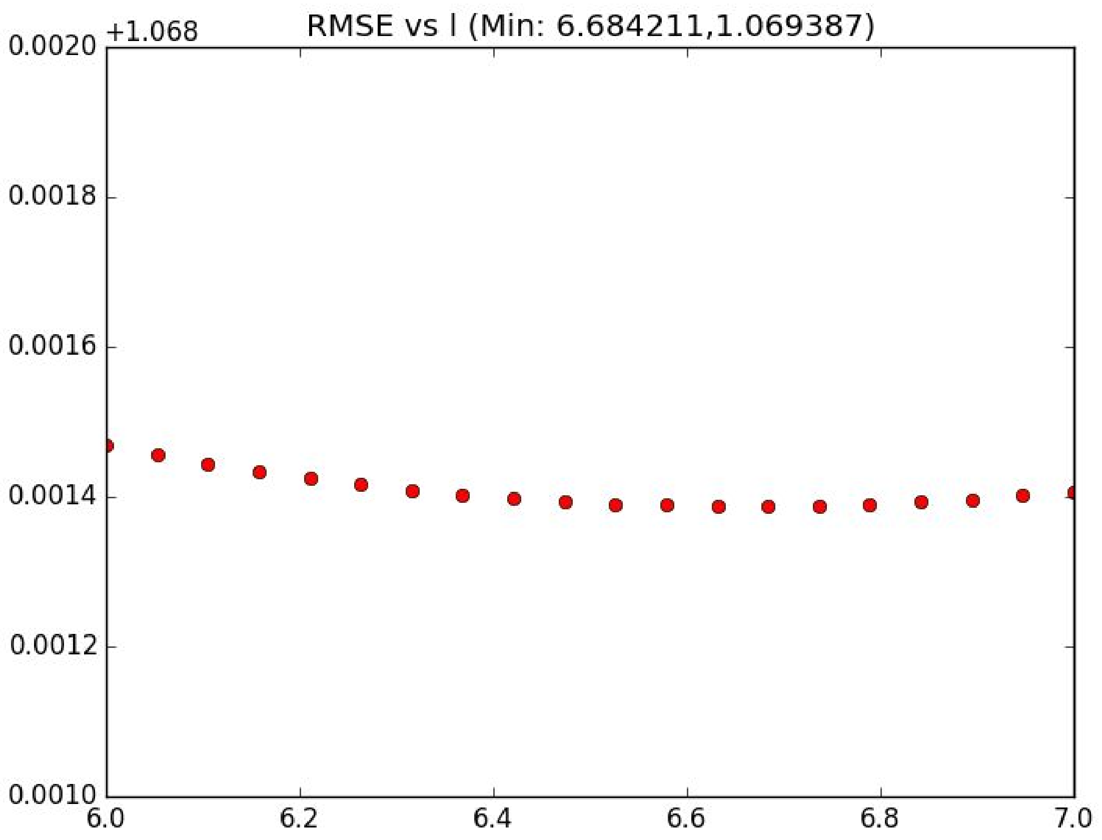
Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace	0.8612	9.59	2009-07-24 17:18:43

1 Linear Regression

1.1 Linear Regression with Regularization

Firstly, we completed the code skeleton based on the linear regression method written for homework. The only variable parameter is the regularization parameter L. With an add-on script, we plotted the correlation of validation RMSE with L ranging from 1 to 15, as shown in the first plot below. Then we zoomed into the range from 6 to 7 with denser spacing. As shown in the second plot below, validation RMSE reaches the minimum 1.0694 when L=6.68.





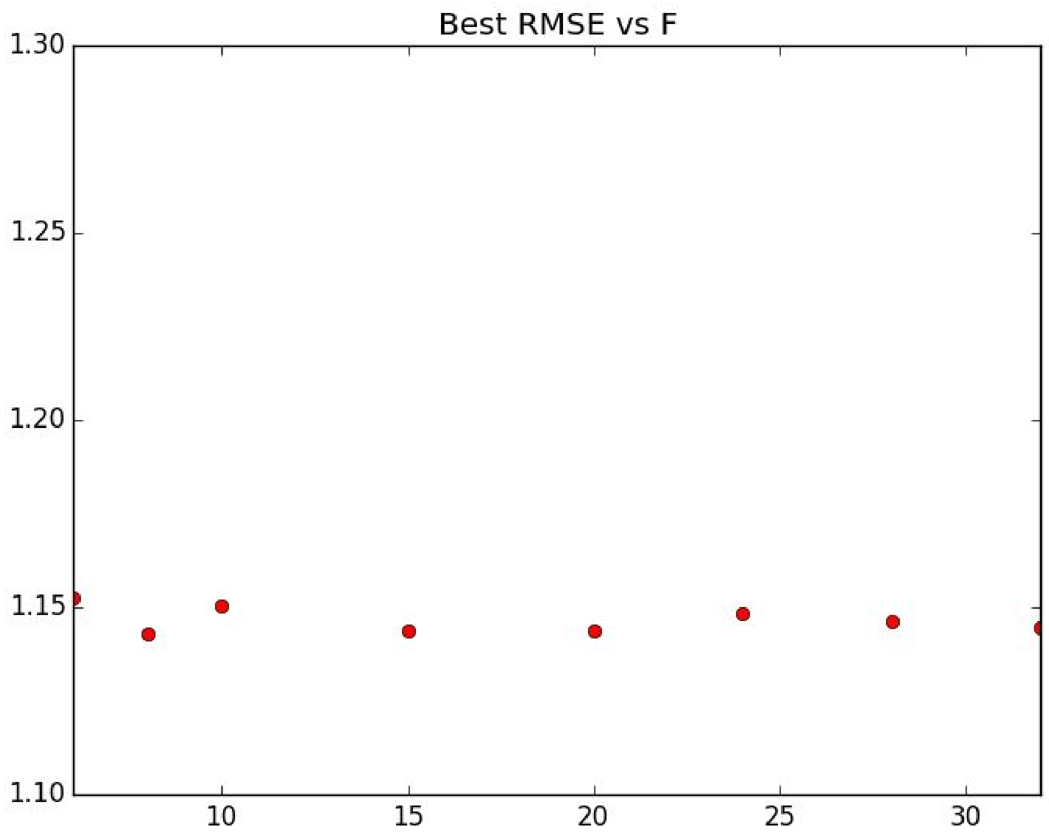
1.2 Extension: Neighborhood

We also implemented the neighborhood method (similar to homework) on top of regularized linear regression. However, the best result is only 1.1672 with L=6.71, which is lower than basic linear regression method. From what we understand about the neighborhood method, the reason might be that the training data is so sparse that the neighborhood information is not helpful.

2 Restricted Boltzmann Machine (RBM)

2.1 Basic RBM

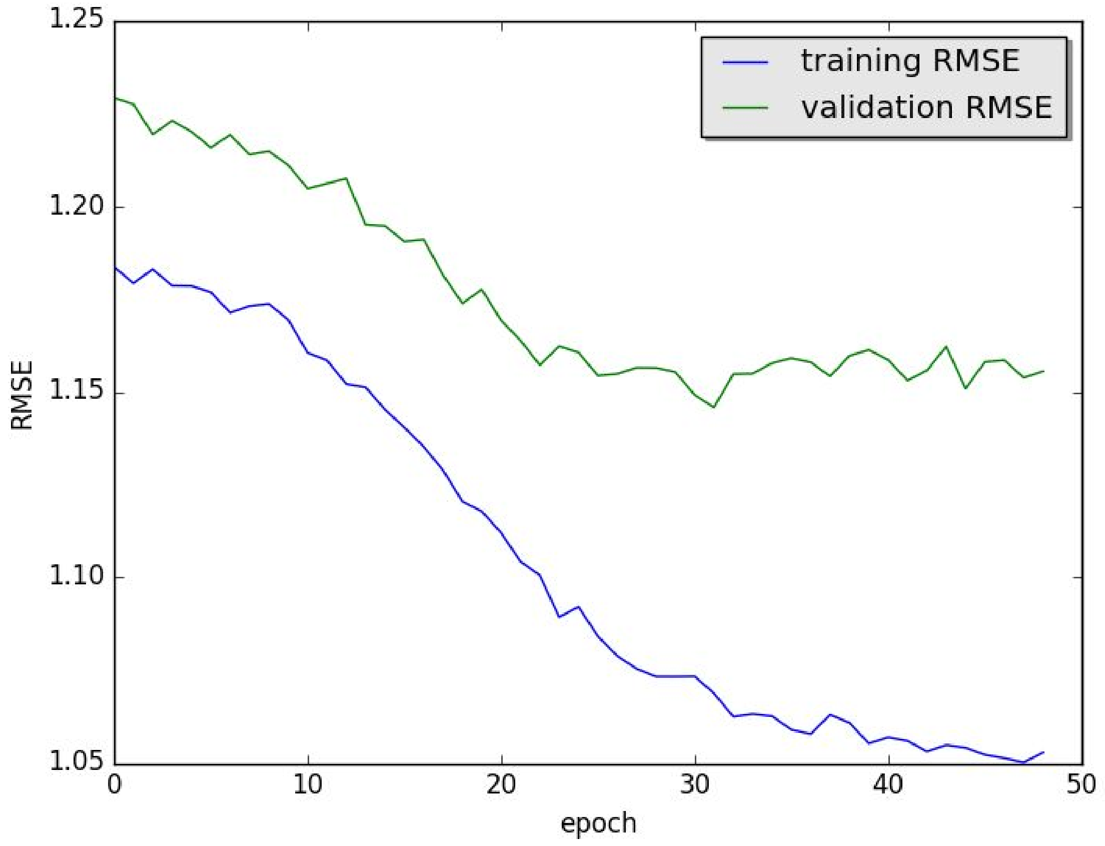
2.1.1 RSME vs F



With momentum=0.5 and max(epoch)=30, we experimented with a series of number of hidden units (F). However, from the plot above, it seems pretty random without a clear pattern. Probably F does not matter too much. Hence, we chose 8 as F tentatively because it gave the lowest RMSE.

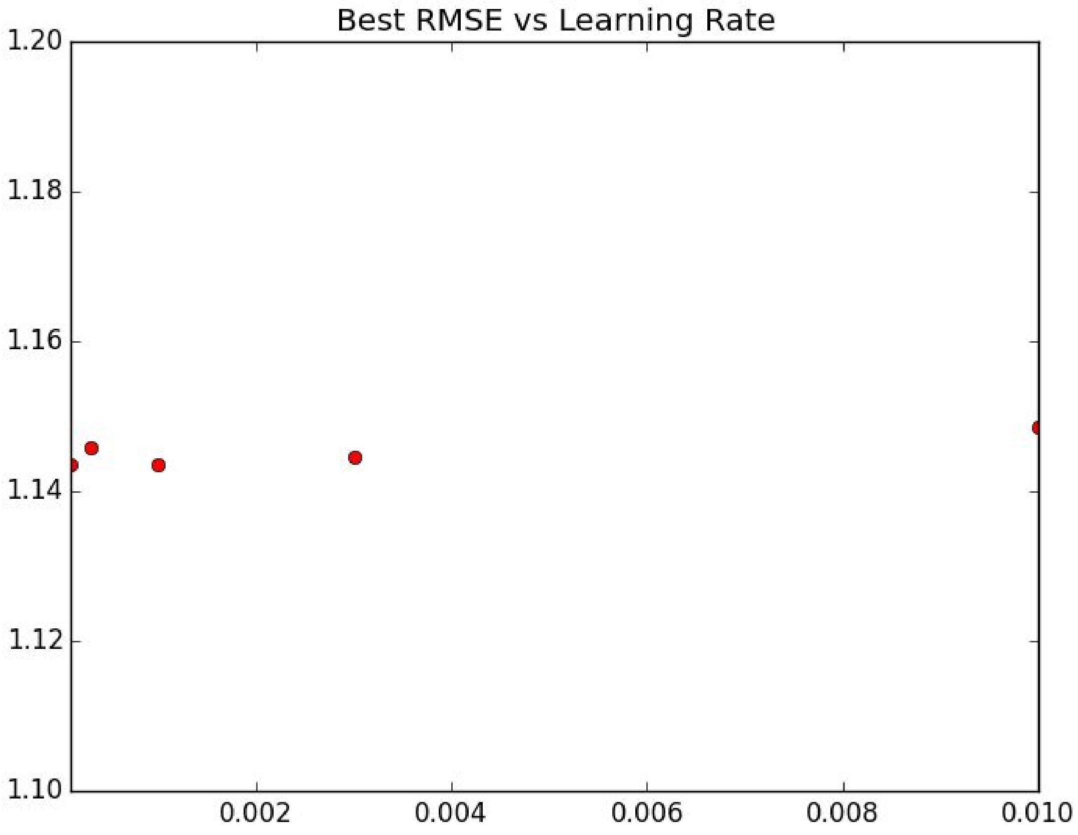
2.1.2 RSME vs epochs

As we can see from the RSME-epoch plot, more epochs does not always give better RMSE, which shows the importance of implementing early stopping.



2.1.2 RSME vs Learning Rate

As shown in the plot below, we experimented with learning rates of [0.0001, 0.0003, 0.001, 0.003, 0.01] (x3 increment). It is hard to determine which one performs better. Hence, would be tuning the learning rate with other extensions integrated later.



2.2 Extension: Momentum

2.2.1 Literature Research

Source: http://sebastianruder.com/optimizing-gradient-descent/index.html#momentum

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another [1], which are common around local optima. In these scenarios, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum as in Image 2.

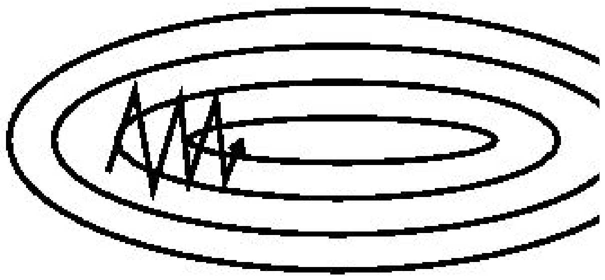
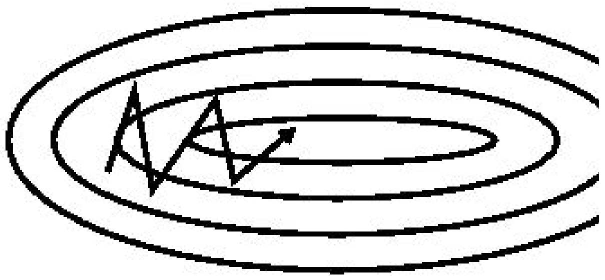



Image 2: SGD without momentum Image 3: SGD with momentum

Momentum [2] is a method that helps accelerate SGD in the relevant direction and dampens oscillations as can be seen in Image 3. It does this by adding a fraction γγ of the update vector of the past time step to the current update vector:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta).$$

$$\theta = \theta - v_t.$$

Note: Some implementations exchange the signs in the equations. The momentum term γγ is usually set to 0.9 or a similar value.

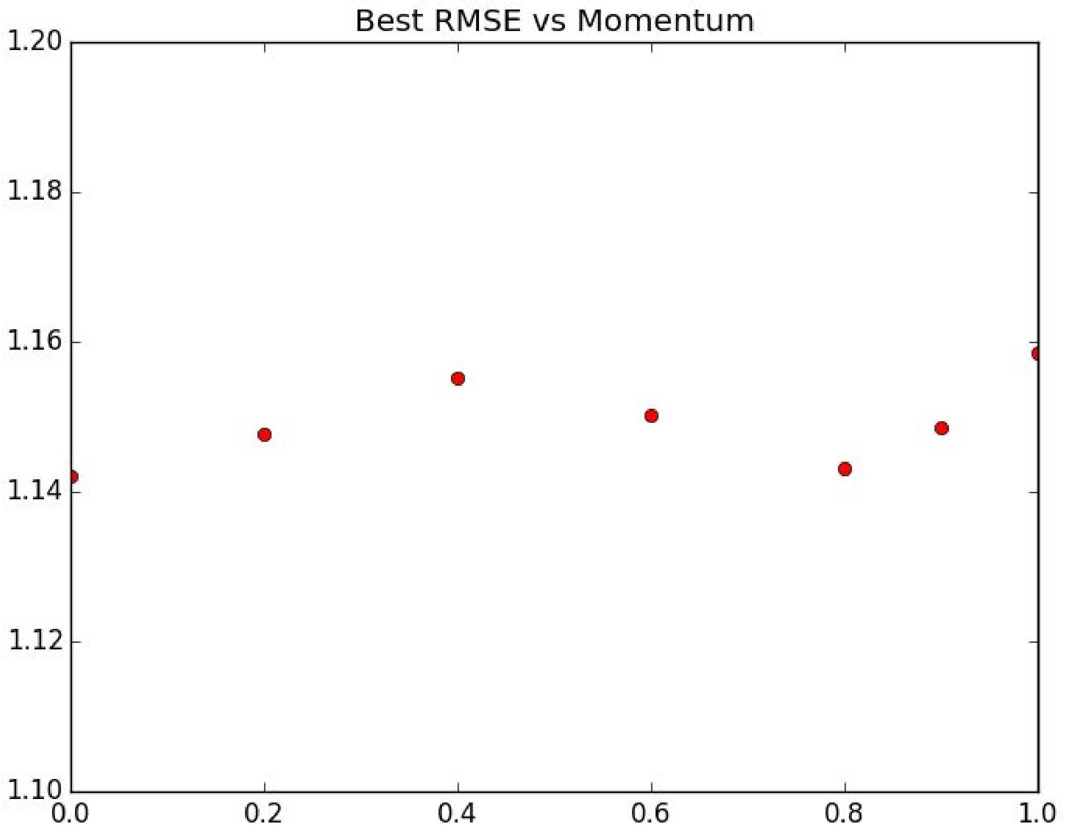
Essentially, when using momentum, we push a ball down a hill. The ball accumulates momentum as it rolls downhill, becoming faster and faster on the way (until it reaches its terminal velocity if there is air resistance, i.e. γ<1γ<1). The same thing happens to our parameter updates: The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, we gain faster convergence and reduced oscillation.

2.2.2 Implementation and Results

Following the tutorial above, we simply added one term to factor in the previous **grad.** Also, we had to initiate **grad** as **`np.zeros(W.shape)`**

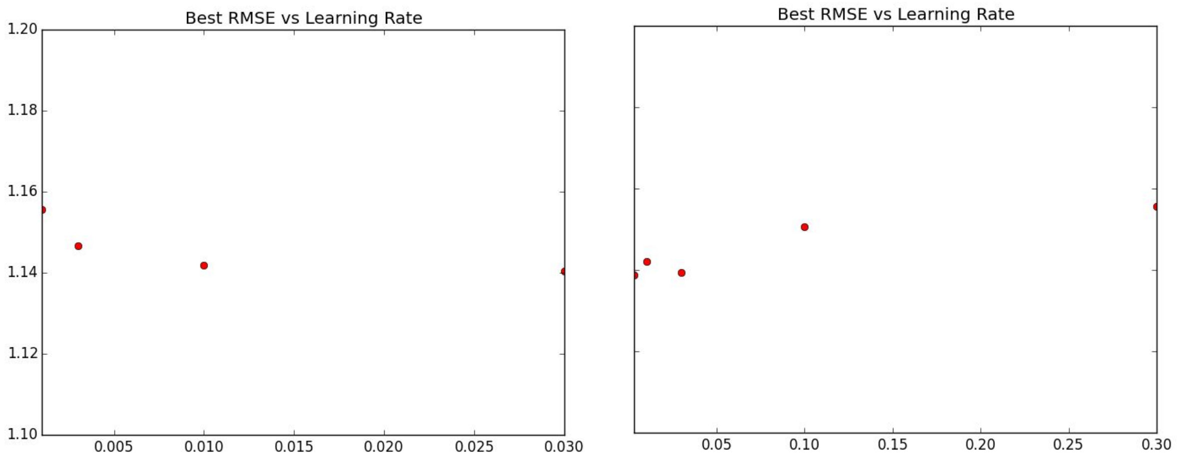
```
grad = momentum*grad + gradientLearningRate * (posprods - negprods)/trStats["n_users"]
```

As shown in the plot below, we experimented momentum parameter ranging from 0 to 1 on top of the basic RBM. From the preliminary results, it seems 0.8 performs the best. However, we still need to tune it together with other extensions in place.

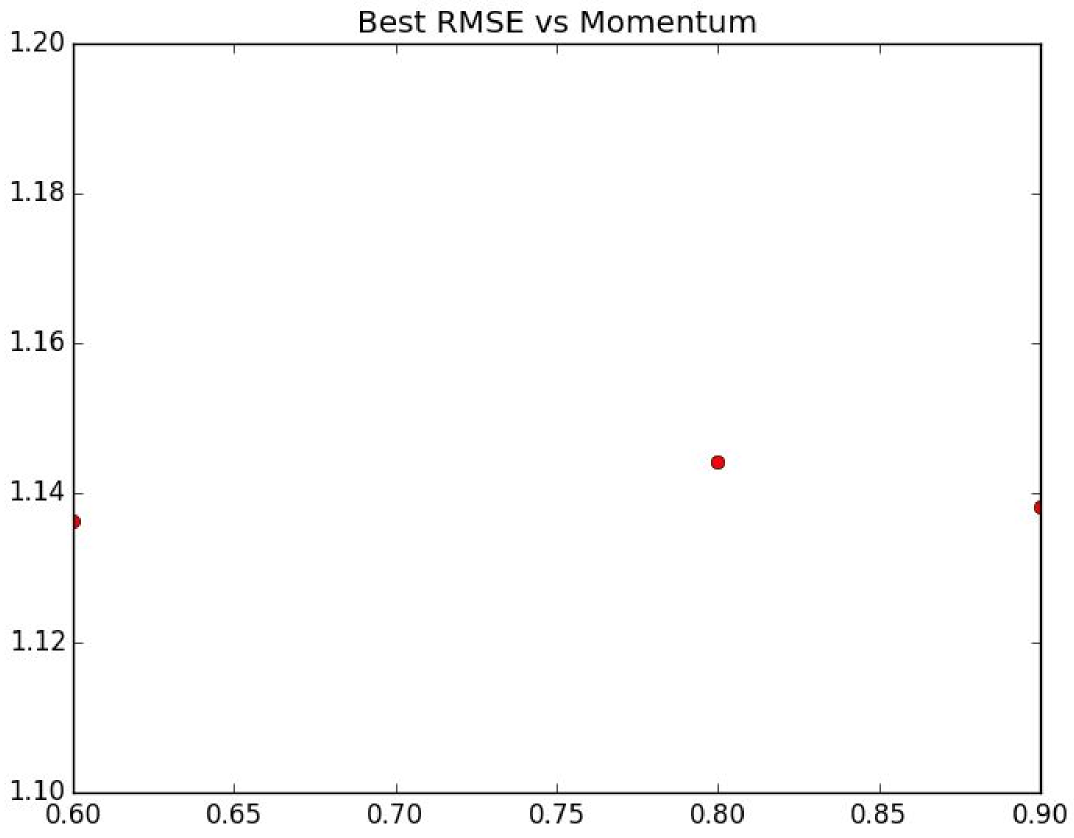


2.3 Extension: Adaptive Learning Rate

We used a simple yet effective method of adaptive learning rate. For each epoch, the learning rate is equal to $alpha/epoch$, where alpha is the initial learning rate. We experimented with a series of initial learning rate: [0.003, 0.3] (x3 increment) and found out 0.03 showed the best performance, as shown in the combined plots below.

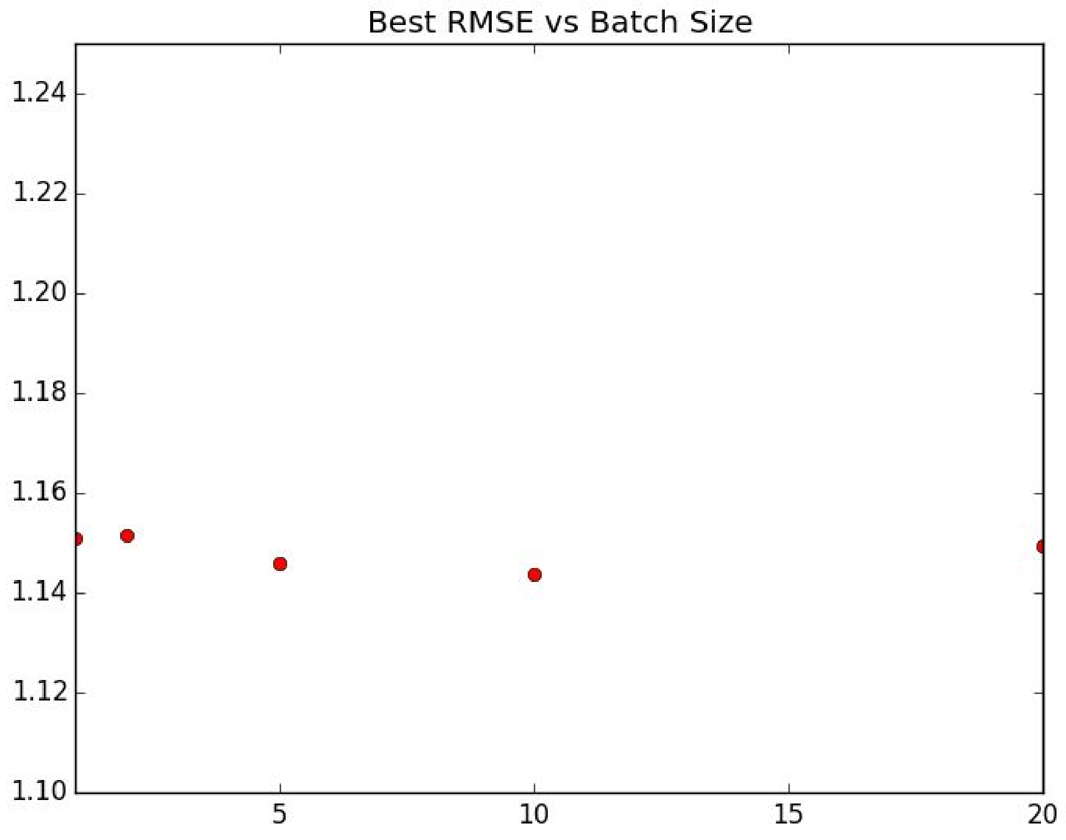


In fact, we also tried other formulas such as $alpha/\sqrt{epoch}$ and $alpha/epoch^2$. The former performed poorly. However, the latter performed slightly better than $alpha/epoch$, as shown in the plot below (tested with three different momentum parameters, two resulting in a noticeable margin under 1.14). Hence, we would be trying out both formulas for the final tuning.



2.4 Extension: Mini Batch

From the book *A Practical Guide to Training Restricted Boltzmann Machines* by Geoffery Hinton, we learned that to avoid having to change the learning rate when the size of a mini-batch is changed, it is helpful to divide the total gradient computed on a mini-batch by the size of the mini-batch, so when talking about learning rates we will assume that they multiply the average, per-case gradient computed on a mini-batch, not the total gradient for the mini-batch. Based the above principles, we ran the program with batch size [1, 2, 5, 10, 20] and it shows that **B=10** has the best performance, which coincides with the value suggested in the guidebook.



2.5 Extension: Early Stopping

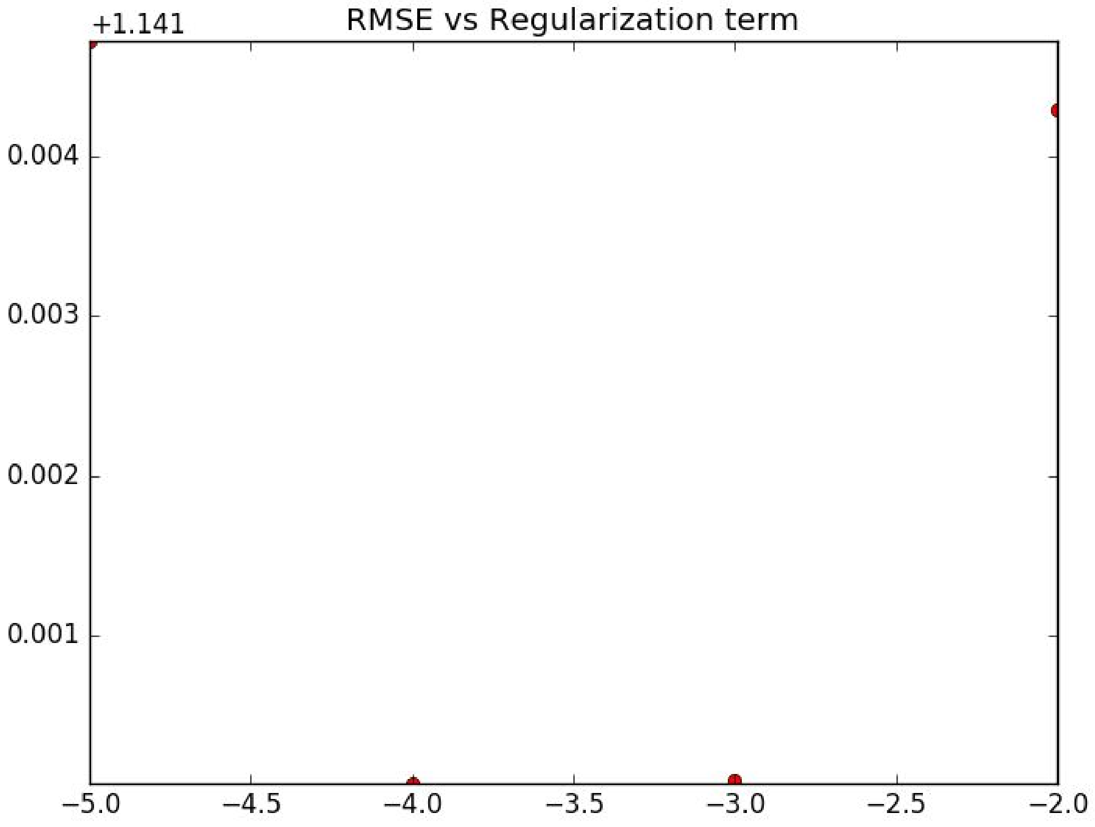
An early stopping method is also used to obtain the epochs with weights with the best RMSE.At each epoch, the lowest RMSE for the validation data is tracked and its respective weights for that epoch are stored. At the end of the specified number of epochs, the weights with the lowest RMSE is used for the prediction instead.

2.6 Extension: Regularization

To prevent overfitting of weights to the training data, a regularization term is added each time the gradient is updated based on weight decay:

$$\Delta W_{ij}^{k} = \gamma(< v_i^k h_j >^{+} - < v_i^k h_j >^{T} - \beta W_{ij}^{k})$$

The goal of weight decay is to penalize the larger weights for each user. The term β is the weight decay term which is tested in range of 0.01 to 0.00001 in the implementation and each value of β is compared based on its RMSE value:



In the figure above we plot the log of the regularization term versus the validation RMSE, which shows a regularization term of 0.0001 produces the best RMSE.

2.7 Extension: Biases

Another possible improvement which unfortunately we did not manage to implement in time was the inclusion of bias terms in each of the visible and hidden units for different users. Due to the situation where each user rated a differents sets of movies. Therefore, a universal bias in the Restricted Boltzmann machine may not be the most ideal model in predicting the ratings for each separate user. Ideally there should be a separate bias for each visible unit which represents all the different movies:

$$p(h_j = 1|\mathbf{V}) \quad = \quad \sigma(b_j + \sum_{i=1}^{m} \sum_{k=1}^{K} v_i^k W_{ij}^k)$$

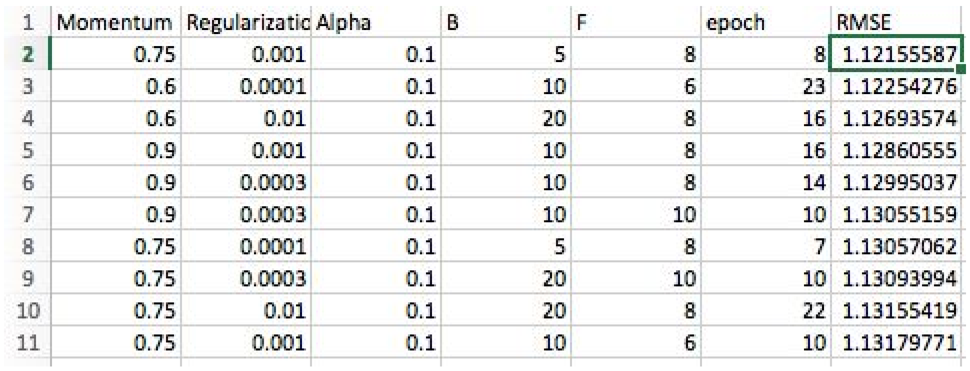
Where b_j is the additional bias term for each different movie the user rated in the visible unit.

2.8 Final Tuning of All Parameters

As discussed above in each extension section, we do not have a definite idea of which parameter is the absolutely the best one for each extension. In fact, during testing, we realized the best parameter for one extension alone may not be the best one with other extensions together. In a way, they may affect each other. Therefore, we could only narrow down each extension parameter to several candidates and perform grid search to tune all the parameters together.

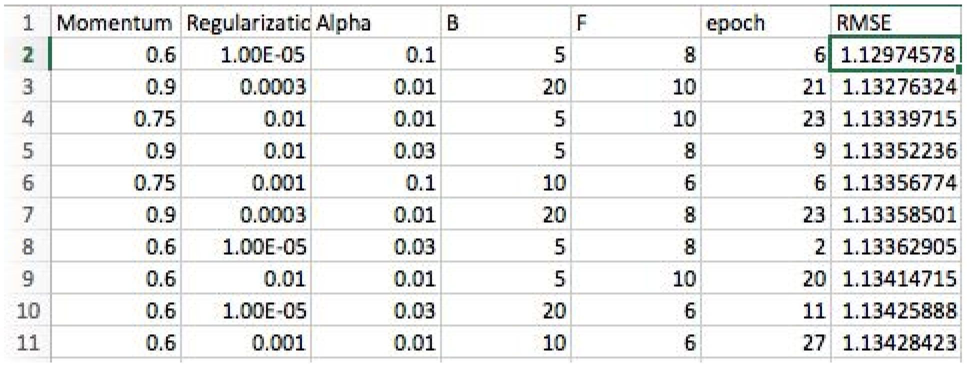
Momentum: mrange = [0.6, 0.75, 0.9]
Regularization: rrange = [0.00001, 0.0001,0.0003, 0.001, 0.01]
Alpha: arange = [0.01, 0.03, 0.1]
Batch Size (B): brange = [5, 10, 20]
#Hidden Units (F): frange = [6, 8, 10]

$alpha/epoch^2$ for Adaptive Learning Rate



1	Momentum	Regularizatic	Alpha	B	F	epoch	RMSE
2	0.75	0.001	0.1	5	8	8	1.12155587
3	0.6	0.0001	0.1	10	6	23	1.12254276
4	0.6	0.01	0.1	20	8	16	1.12693574
5	0.9	0.001	0.1	10	8	16	1.12860555
6	0.9	0.0003	0.1	10	8	14	1.12995037
7	0.9	0.0003	0.1	10	10	10	1.13055159
8	0.75	0.0001	0.1	5	8	7	1.13057062
9	0.75	0.0003	0.1	20	10	10	1.13093994
10	0.75	0.01	0.1	20	8	22	1.13155419
11	0.75	0.001	0.1	10	6	10	1.13179771

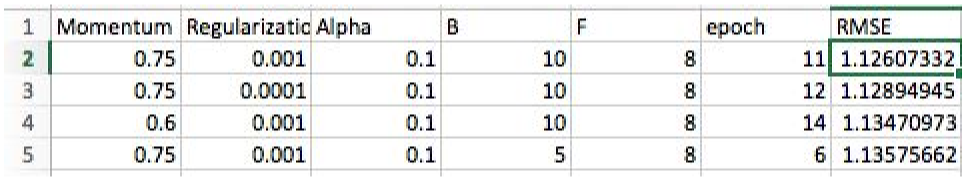
$alpha/epoch$ for Adaptive Learning Rate



1	Momentum	Regularizatic	Alpha	B	F	epoch	RMSE
2	0.6	1.00E-05	0.1	5	8	6	1.12974578
3	0.9	0.0003	0.01	20	10	21	1.13276324
4	0.75	0.01	0.01	5	10	23	1.13339715
5	0.9	0.01	0.03	5	8	9	1.13352236
6	0.75	0.001	0.1	10	6	6	1.13356774
7	0.9	0.0003	0.01	20	8	23	1.13358501
8	0.6	1.00E-05	0.03	5	8	2	1.13362905
9	0.6	0.01	0.01	5	10	20	1.13414715
10	0.6	1.00E-05	0.03	20	6	11	1.13425888
11	0.6	0.001	0.01	10	6	27	1.13428423

Above are the top 10 results for $alpha/epoch^2$ and $alpha/epoch$ Adaptive Learning Rate. Obviously $alpha/epoch^2$ performs better in general and exhibits more consistency in certain

parameters (such as Alpha, B and F). However, Momentum and Regularization are not very consistent. Hence, we fix Alpha=0.1, F=8 and only vary Momentum, Regularization and B.
mrange = [0.6, 0.75, 0.9]
rrange = [0.0001, 0.001]
arange = [0.1]
brange = [5, 10]
frange = [8]



1	Momentum	Regularizatic	Alpha	B	F	epoch	RMSE
2	0.75	0.001	0.1	10	8	11	1.12607332
3	0.75	0.0001	0.1	10	8	12	1.12894945
4	0.6	0.001	0.1	10	8	14	1.13470973
5	0.75	0.001	0.1	5	8	6	1.13575662

So with the more narrowed-down tuning, it shows more consistency in the top performers. We can conclude that the best parameters are:

F = 8
Momentum = 0.75
Regularization = 0.001
B = 10
Alpha = 0.1

3. Conclusion

Despite implementing all the extensions except the bias and tuning the parameters to the best, the performance of RBM is still not as good as linear regression. Probably biases is a very significant extension. This project has been a fulfilling learning journey and definitely aroused our interest in RBM and recommendation system beyond what we learned in class. In the future, with more time, we would try implementing biases and more advanced extensions to further improve the RBM recommendation system.

References

i) R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In Proceedings of the 24th international conference on Machine learning, pages 791–798. ACM, 2007.

ii) G. Louppe. Collaborative filtering: Scalable approaches using restricted Boltzmann machines. Master's thesis, Université de Liège, Belgique, 2010.

iii) An overview of gradient descent Algorithm
http://sebastianruder.com/optimizing-gradient-descent/index.html#momentum

iv) Geoffrey Hinton, *A Practical Guide to Training Restricted Boltzmann Machines*