CoE202 - Basics of Artificial Intelligence “Big data analysis and machine learning”

Mukanov Ayan 20170881

Activity#3

Movie recommender system

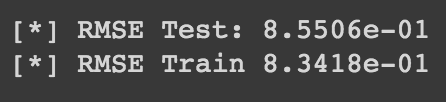
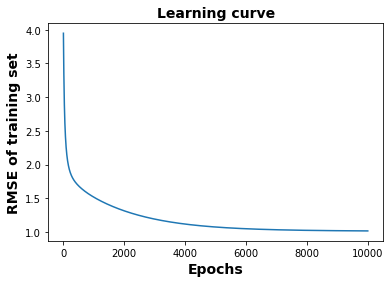
Report

**Part1: Momentum Optimizer**

1. Apply the GradientDescentOptimizer():

Optimizer = tf.train.GradientDescentOptimizer(lr).minimize(cost, global\_step = global\_step)

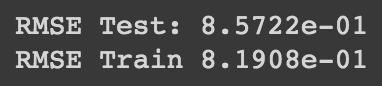
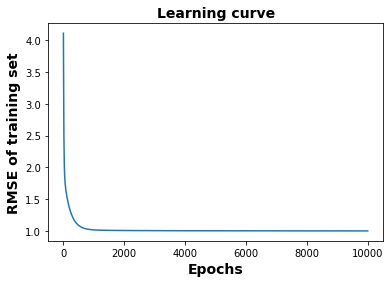
Result:



1. Apply the MomentumOptimizer() with momentum at 0.9

Optimizer = tf.train.MomentumOptimizer(learning\_rate, 0.9).minimize(cost, global\_step = global\_step)

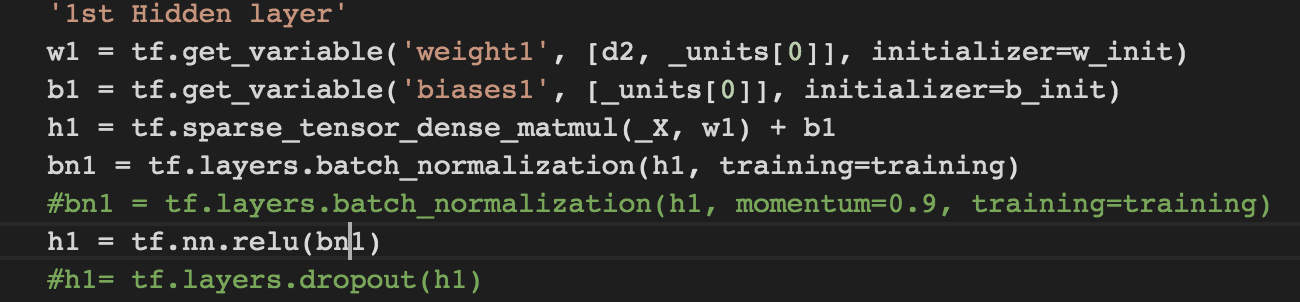
Result:

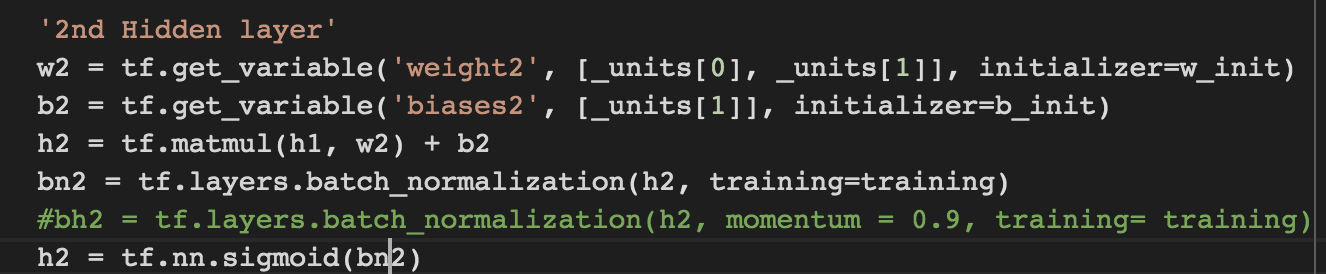


**Part 2: Batch Normalization**

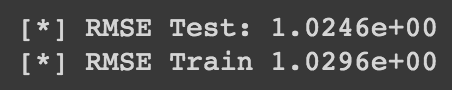
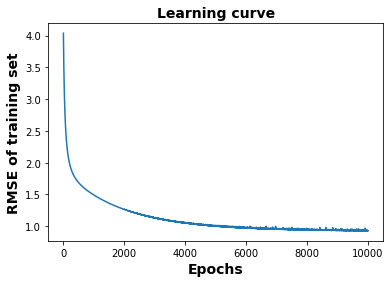
Normalize 1st and 2nd hidden layers with tf.layers.batch\_normalization()

Normalization for GradientDescentOptimizer:

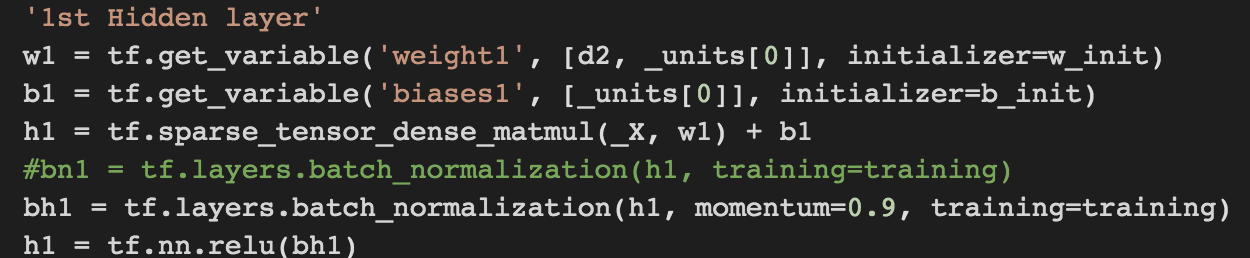
 for 1st layer

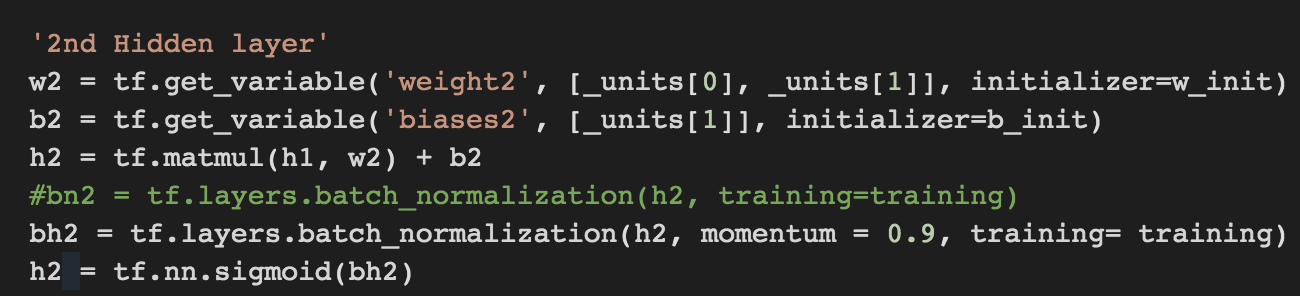
 for 2nd layer

Result:

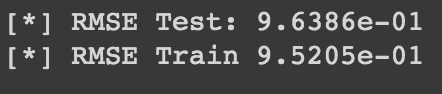
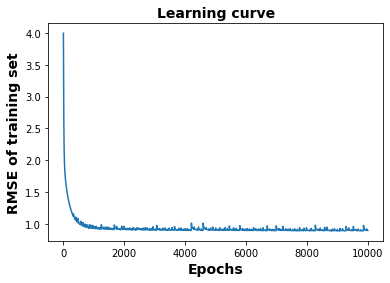


Normalization for MomentumOptimizer:

 for 1st layer

 for 2nd layer

Result:

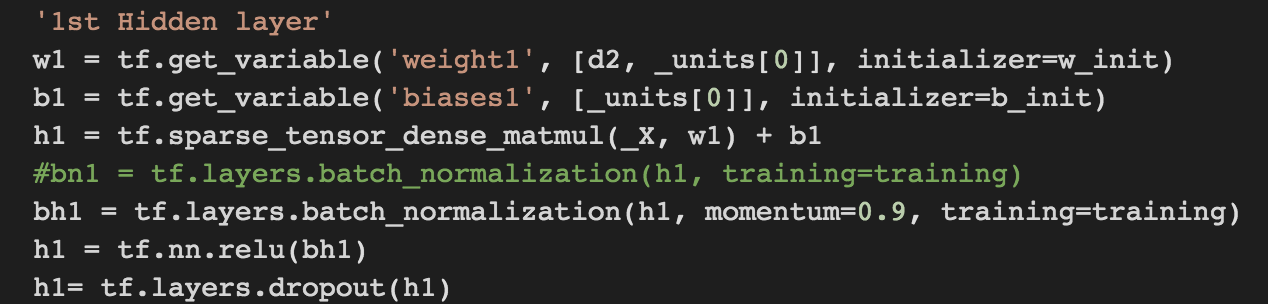


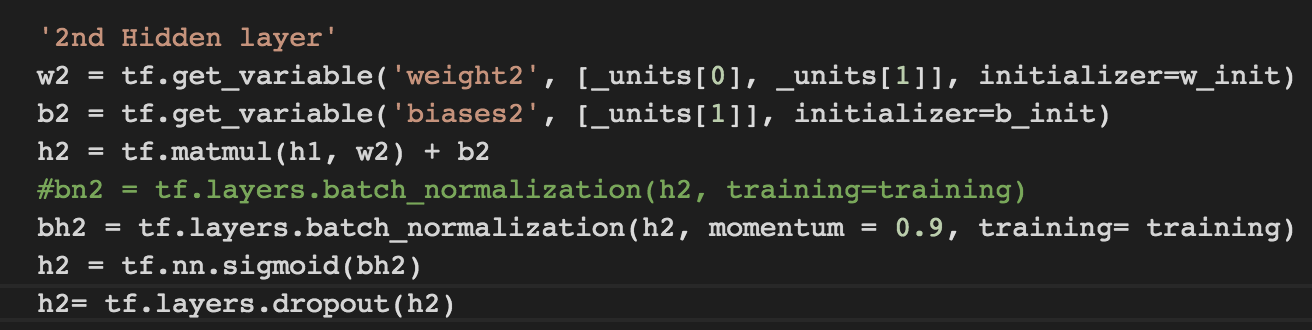
**Part 3: Dropout**

Apply tf.layers.dropout() to the 1st and 2nd hidden layers of autoendoder in Part 2(normalized MomentumOptimizer)

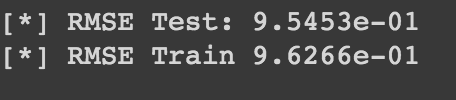
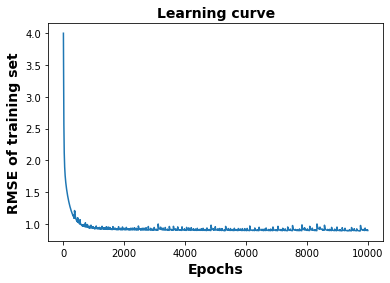
h1 = tf.layers.dropout(h1) - for 1st layer

h2 = tf.layers.dropout(h2) - for 2nd layer

 for 1st layer

 for 2nd layer

Result:



**Conclusion**

It is noticeable that MomentunOptimizer() outperforms the GradientDescentOptimizer() and the overall test and train RMSE of former is much more smaller than that of the latter. The reason for this is because MomentumOptimizer() accelerates GradientDescentOptimizeer and converges faster.

With the batch normalization, we can see that the trend is similar between two optimizers, and RMSE is not affected much.

With the “dropout” the trend is the same. The RMSE is almost the same as that of normalized MomentumOptimizer(), although the curve is a bit smoother, which probably causes a more stable/predictive behavior of gradients and also prevents overfitting.