**Project Report: Movie Recommendation using an AutoEncoder Network**

Introduction/Background

We chose to develop a movie recommendation network using data from the MovieLens Education and Development “small” dataset1. We used a smaller dataset to reduce the time and memory demands, but a larger set would give more accurate results. This dataset contains 100,836 ratings by 610 users of 9724 movies. Each rating contains a timestamp of when the rating was given, but we ignored them for this project. It also came with a file linking the movie ID’s with the movie title and genres that each movie falls into. For this project, we focused on the movie ratings as our feature space instead of the movie genres. The data is contained in a pandas DataFrame, with each row being a vector representing a single user’s ratings and each column corresponds to a specific movie. The ratings range from 1-5 and are half-integer steps (ie. 1, 1.5, 2, 2.5, etc.). The dataset contains null ratings, since not every user has given a rating for every movie, and we set these ratings to 0. This dataframe was then split into training/testing sets, where 80% of the users were used for training and 20% was used for testing. We implemented a single hidden layer, autoencoder network designed by Soumya Ghosh2, which is based off of a paper by Suvash Sedhain *et al*3. The general architecture design of this network can be seen in Figure 1 below, which comes from the previously mentioned paper.

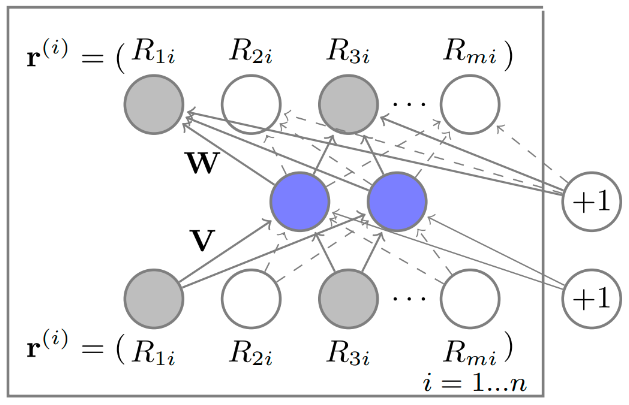


Figure 1: Network design of the autoencoder network used as a movie recommendation model3.

This autoencoder network is an unsupervised neural network with a single hidden layer with k-neurons, shown in blue. The model takes in a single user’s (*i*) array of movie ratings, r(i), as input, which has a high-dimensionality (the number of movies, *m*, in our case 9724). It encodes this array by projecting it onto the hidden layer’s lower-dimensional space, compressing the data. It then attempts to reconstruct the original array by decoding the hidden layer. Using this reconstructed array, it calculates the error between the output and input as the Mean Squared Error (MSE), and adjusts the network’s weights to decrease this error using the gradient-descent backpropagation method AdagradOptimizer, which has an initial learning rate parameter but adjusts its learning rate as it learns. Since the users don’t rate every movie, this model only updates the weights associated to the non-zero input neurons. The network is then fed the next user’s rating input, up to *n* users, and continually updates the weights. Once the network is trained, a test user’s ratings array can then be input into the network, and the output should produce an array where the nodes with higher values represent movies that the user is likely to enjoy, given their input ratings and the movies that other users with similar interests enjoyed.

Problem Statement

It is beneficial for many companies, such as Netflix or Amazon, to be able to learn the interests of their individual users in order to customize their experience to increase the likelihood of their repeat usage or business. Neural networks have become popular for making user recommendations on large datasets by learning the features that correspond to an individual’s interests or input. For this project, which is similar to the challenge Netflix faces, we have movie ratings given by users and we want to be able to use this information to predict which other movies that user would enjoy watching that they haven’t yet seen. This is very important for Netflix, because if they can reliably recommend movies that their customers enjoy, then their customers will be more likely to continue using their service and pay the monthly fee.

Setup and Running of the Network

The program is run within a Python 3.7 Tensorflow environment. The external libraries necessary to run this program are pandas4, numpy5, tkinter6, and random7. Matplotlib8 was also used to create the plots shown in the Results/Analysis section. It is recommended to run this program in a python IDE, such as Spyder IDE, which can be accessed by downloading Anaconda9. You can then run the program by selecting the green “Play” button at the top of the IDE. The program will then train the network using the ratings from the MovieLens dataset. Once trained, a window will pop-up that prompts the user to give their own ratings on a random set of movies. Choose a rating from the provided options by clicking in its corresponding checkbox and select “Next Movie” to submit the rating and go to the next movie. If you have not seen the movie and don’t wish to give a rating, select “Next Movie” without selecting a checkbox. Once you have rated as many movies as you’d like, select “Finished Rating” and a new window will open displaying the top five movies that the network has recommended based on your rating input and the learned features from training.

Results/Analysis

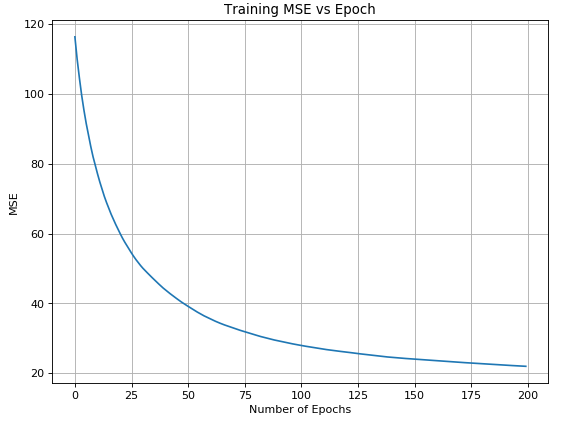
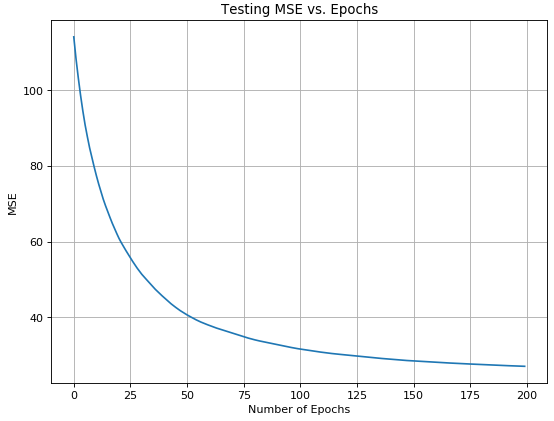
 

Figure 2: Plots of the Mean Squared Error during training (left) and testing (right) vs. the number of epochs. These results were produced using the initial conditions of 256 hidden layer neurons, a batch size of 100, and a learning rate of 0.1.

As can be seen in Figure 2 above, the MSE is decreasing as the number of epochs increases, showing that the network is in fact learning and is improving its accuracy as it updates its weights. However, as the curve begins to plateau around 200 epochs, the MSE is still relatively high at about 22 and 33 for training and testing, respectively.

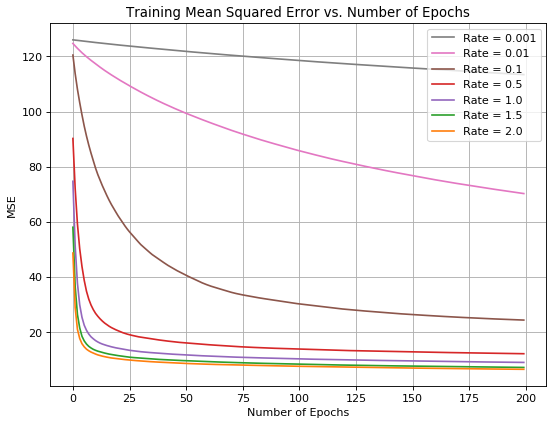
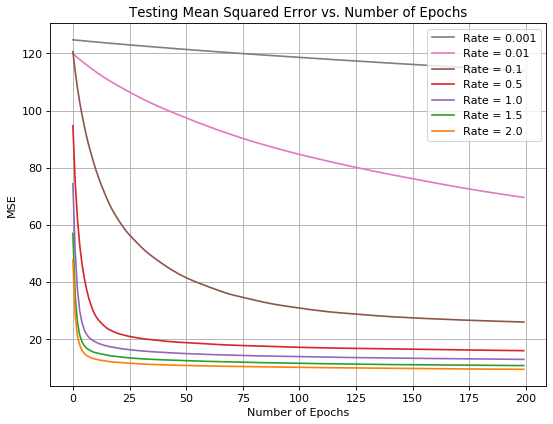
 

Figure 3: Plots showing the decrease in the MSE during training (left) and testing (right) vs. the number of epochs for multiple different learning rates. This was run with 256 hidden layer neurons and a batch size of 100.

As can be seen from Figure 3, the MSE decreased as the learning rate was increased, and began to plateau within fewer epochs. The network uses the AdagradOptimizer, which adjusts the learning rate as it goes. A lower learning rate corresponds to the parameters having frequently occurring features, whereas a higher learning rate corresponds to infrequently occurring features10. Observing that a higher initial learning rate results in a lower MSE implies that the parameters have infrequently occurring features, which makes sense due to the sparsity of the data as well as its high-dimensionality. Based on these plots, we have decided to use a learning rate of 2 for this network and that 25 epochs appears to be where the curve begins to plateau.

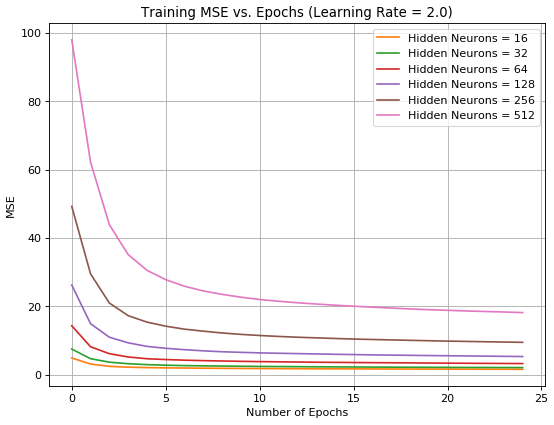
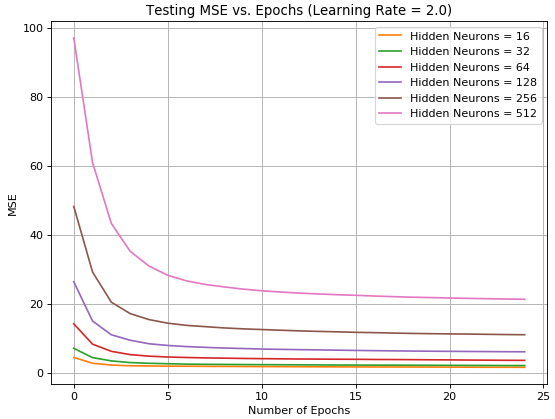
 

Figure 4: Plots showing the MSE during training (left) and testing (right) vs. the number of epochs using different numbers of hidden layer neurons. This was run with a batch size of 100 and a learning rate of 2.

As can be seen from Figure 4 above, the MSE curves decrease with a decrease in the number of hidden neurons. Based on this, we have chosen to use 16 neurons for the hidden layer of our model.

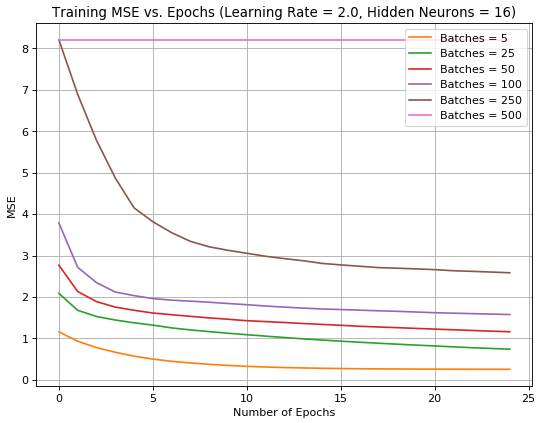
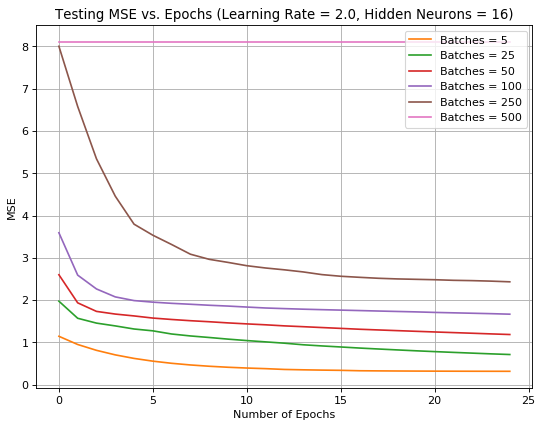
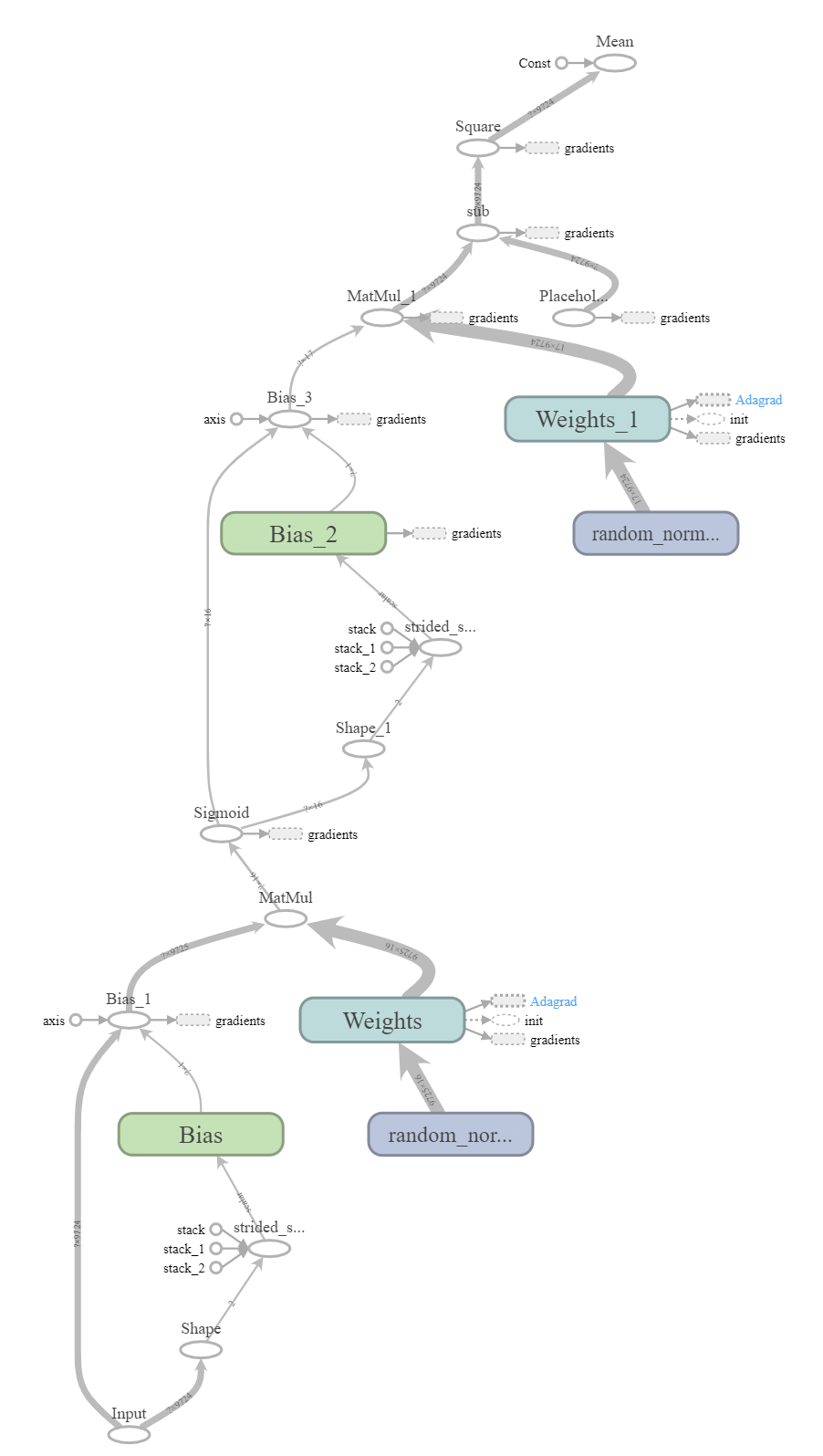
 

Figure 5: Plots showing the MSE during training (left) and testing (right) vs. the number of epochs using different batch sizes. This was run with a learning rate of 2 and 16 hidden layer neurons.

As can be seen from Figure 5 above, the MSE decreases with a decrease in the batch size. For a batch size of 500, the network was not able to learn at all and the MSE remained constant. Based on these results, we have decided to use a batch size of 5 for our model.

**Main Graph**



**Auxiliary Nodes**

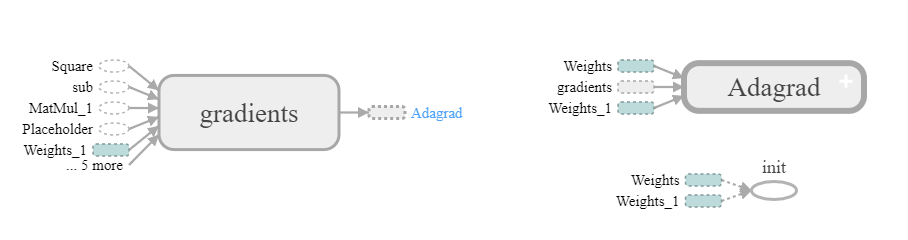


Figure 6: The above images display the diagram of the end result of our network, which was produced using Tensorboard.

Conclusion

As can be seen from the results above, our network is able to learn features within our dataset of movie ratings and can improve the Mean Squared Error between the input and output of the model by adjusting its weights using the gradient-descent backpropagation algorithm AdagradOptimizer. We were also able to improve these results greatly from a test MSE of roughly 33 after 200 epochs to an MSE of approximately 0.5 after only 10 epochs. This was achieved by changing the initial learning rate from 0.1 to 2, changing the number of hidden neurons from 256 to 16, and changing the batch size from 100 to 5.

References

1. GroupLens - MovieLens Dataset  
   [Accessed Dec. 12, 2018]  
   <https://grouplens.org/datasets/movielens/>
2. Soumya Ghosh, *Recommender System on the MovieLens dataset using an Autoencoder and Tensorflow in Python*, Medium.com, March 17, 2018  
   [Accessed Dec. 12, 2018]  
   <https://medium.com/@connectwithghosh/recommender-system-on-the-movielens-using-an-autoencoder-using-tensorflow-in-python-f13d3e8d600d>
3. Suvash Sedhain *et al*., *AutoRec: Autoencoders Meet Collaborative Filtering*, Australian National University  
   [Accessed Dec. 12, 2018]  
   <http://users.cecs.anu.edu.au/~akmenon/papers/autorec/autorec-paper.pdf>
4. Python Data Analysis Library (Pandas)  
   <https://pandas.pydata.org/>
5. Numpy Library, Scipy  
   <http://www.numpy.org/>
6. Tkinter Library, Python  
   <https://docs.python.org/3/library/tkinter.html>
7. Random Library, Python  
   <https://docs.python.org/2/library/random.html>
8. Matplotlib Library  
   <https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html>
9. Anaconda – Python 3.7  
   <https://www.anaconda.com/download/>
10. Sebastian Ruder, *An Overview of Gradient Descent Optimization Algorithms: Adagrad*, Jan 19, 2016  
    [Accessed Dec. 15, 2018]

<http://ruder.io/optimizing-gradient-descent/index.html#gradientdescentvariants>