Machine Learning and Low-Rank Matrix Approximations

Bo Bleckel[1]
[1]*Bowdoin College, Brunswick, Maine, USA*
lbleckel@bowdoin.edu

Keywords: Neural Networks, low-rank matrix approximations, Singular Value Decomposition.

Abstract: In linear algebra, we use the singular value decomposition (SVD) to find low-rank approximations of high-rank matrices. These low-rank approximations can be used to store the data in a matrix in a more compact form. We know that the SVD method for approximating matrices is the best method. However, we would like to explore another way of approximating high-rank matrices. Using a Neural Network (NN), we can approximate a matrix based on how if acts on vectors. By limiting a crucial part of the network's structure, we can force the approximation of the matrix to be of a certain rank. In this paper we explore the accuracy of these approximations compared to the original matrix and the SVD approximation. When we set out, we hoped to find that the NN approximation is essentially equivalent to the SVD approximation. This paper enumerates the ways in which the two were the same, and also the ways in which they vary.

1 INTRODUCTION

Low rank approximations of matrices are essential to data compression. One of the standard ways of doing this is to calculate the singular value decomposition of a matrix, and truncate the singular values. Our theory is that we can get the same low rank version of the original matrix by using a Neural Network (NN) made up of carefully constructed layers (detailed in Section 3) to limit the rank of an output matrix determined by the neural network. A neural network is a complex set of machine learning algorithms that approximates patterns in data. We hoped to find that the neural network approximation was equivalent to the SVD approximation. We ran the neural network on random small matrices, large random matrices, images, and a few specific matrices — the identity matrix of different sizes, symmetric matrices, and others. We explored the differences of the SVD and NN approximations to determine if this was a viable option for matrix approximation.

2 SINGULAR VALUE DECOMPOSITION

The SVD creates a diagonalization of a matrix A of the form

$$A = U\Delta V^T$$

where Δ is a diagonal matrix with the singular values of A along its diagonal. Due to how this diagonalization is constructed, the higher singular values are more important to the fidelity of the matrix. Thus, the entries of Δ are ranked in decreasing order such that $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n$. For example, we might have Δ in 4-dimensional space such that

$$\Delta = \begin{pmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 \\ 0 & 0 & 0 & \sigma_4 \end{pmatrix}$$

(which indicates that the original matrix is of rank 4). To find the rank $k = 2$ approximation of the original matrix A, we truncate Δ, keeping only the first 2 singular values:

$$\Delta_2 = \begin{pmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

We can now recalculate A using the new SVD:

$$A = U\Delta_2 V^T,$$

where U and V are the same in both cases.

The SVD is calculated in a particularly clever way. If A is symmetric, the SVD is simply the diagonalization of A such that

$$A = PDP^{-1}$$

where P are the eigenvectors of A and D is the diagonal matrix of the associated eigenvalues. If A is not symmetric (square or not) we calculate the SVD in the following way.

1. Calculate the eigenvalues and eigenvectors of A^TA to get $V\Delta^2V^T$ where the columns of V are the eigenvectors of A^TA and Δ^2 contains the corresponding eigenvalues. Note A^TA is a symmetric matrix and thus this is the normal diagonalization of A^TA.
2. Calculate the eigenvalues and eigenvectors of AA^T to get $U\Delta^2U^T$ where the columns of U are the eigenvectors of AA^T and Δ^2 contains the corresponding eigenvalues. Again, note that AA^T is a symmetric matrix, and thus is the usual PDP^{-1} diagonalization of AA^T.
3. Normalize (make orthogonal) the columns of U, V and Δ to get the final SVD.

2.1 BEST POSSIBLE APPROXIMATION

While it is nice that the SVD provides an easy way to compute the diagonalization of a matrix A, is it the best possible way to do it? We can prove this through the Eckhart-Young theorem which states that A_k is the best rank k approximation to A using either the 2-norm or the Frobenius norm. We will briefly prove this.

1. First we must know the following lemma: Suppose V and W are subspaces of $\mathbb{R}^n$ & dim V + dim $W > n$. That implies that $V \cap W$ is non trivial. This can be proved by contradiction. Assume $V \cap W = \{0\}$. Let $\{v_i\}_{i=1}^k \& \{w_i\}_{i=1}^l$ be bases for V, W & note $W_i \notin \text{span}\{v_i\}_{i=1}^k$. Thus $\{w_i\}_{i=1}^l \cup \{v_i\}_{i=1}^k$ is a linearly independent set. Since $l + k > n$, this is impossible, and therefor the above statement holds.
2. Now we can prove the Eckhart-Young theorem. Assume B is a better approximation that A_k, i.e.

$$\|A - B\|_2 < \|A - A_k\|_2 = \sigma_{k+1}$$

(a similar argument follows for the Frobenius norm). Since B is rank k, the dimension of the kernel of B is equal to $n - k$. Thus there exists a $n - k$ dimensional subspace of the domain such that $Bw = 0$ if $w \in W$, where W is the kernel of B. Thus

$$Aw = (A - B)w \quad \text{if } w \in W.$$

Thus

$$\|Aw\|_2 = \|(A - B)w\|_2 \leq \|A - B\|_2 \, \|w\|_2$$

$$\|Aw\|_2 < \sigma_{k+1} \, \|w\|_2 \, \forall w \in W.$$

Now let's examine $X = \text{span}\{v_1, \ldots, v_{k+1}\}$ where $\{v_i\}$ are columns of V. Then

$$\|Av\|_2 \geq \sigma_{k+1} \, \|v\|_2 \, \forall v \in \text{span}\{v_1, \ldots, v_{k+1}\}.$$

We can see that the dimension of $X = k + 1$, and the dimension of $W = n - k$. Thus the dimension of X plus the dimension of W is equal to $n + 1$ which is greater than n. By our lemma listed above, there exists a vector $z \in X \cap W$ that is non-zero. Therefore

$$\|Az\|_2 < \sigma_{k+1} \, \|z\|_2 \;\&\; \|Az\|_2 \geq \sigma_{k+1} \, \|z\|_2$$

which is impossible.

3 THE NEURAL NETWORK

The "neural network" (also referred to as "NN" or "neural net") is a machine learning algorithm that attempts to mimic the structure and processes of the brain in order to effectively learn about a given problem. A neural net consists of several layers, each of which contributes to forming inferences about the given data, and the underlying rules we seek to learn. A given layer in the network is made up of a certain number of *neurons*, each of which represents a different part of the input layer (in many cases, neurons in inner layers are actually made up of some combination of previous neurons, and may not directly represent the input data). Most of the time a neural net is used with a vector as the input, and another vector as the output. In our case, we are dealing with matrices, and how they act on vectors. We designed our neural net to take in a random vector, and we told it that the output was the result of multiplying that vector by a matrix. We let it learn on some large number of random vectors, and hoped that in the end it would recognize the pattern of the effect that the matrix had on the vector. The input vector and the output vector both have size n so we made a matrix that had input and output layers with n neurons. We used Tensorflow and Keras to build and train our network running on Python. Tensorflow is an open source machine learning software designed by a team at Google. Keras is a high-level library designed to make Tensorflow easier to use. Keras is what is called a "front-end" while Tensorflow is the "back-end."

3.1 LAYERS OF THE NEURAL NETWORK

For our purposes, the input of the neural net will always be a vector (dimension n), and the output will be a vector of the same dimension, but lower altered by the matrix. We have used four total layers: an input layer of size n, a hidden layer of size n, a small center layer of size k (for a rank k approximation), and an output layer of size n. The decision to use this layer design was not done in vain. Section 5.1 explains how we got to this network design. A visualization of a general neural network is shown in Figure 1.

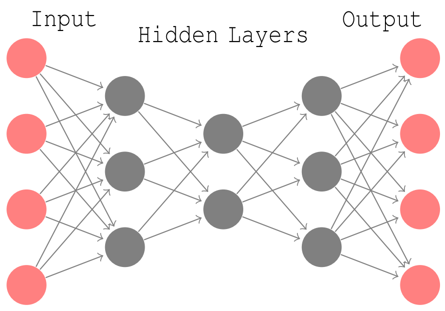

Figure 1: Dense neural network with 3 hidden layers.

3.1.1 FORCING THE RANK OF THE APPROXIMATION

As mentioned above, we have a smallest layer of size k. This layer limits the rank of the output matrix, and is referred to as the “choke-point” in the layering of the neural net. This layer is key to forcing our approximation to be of desired rank k. So the network depicted in Figure 1 would take a 4-dimensional matrix and force a rank 2 approximation of it.

4 PERFORMANCE

As we tested these approximations, we needed a metric to know how well our neural net was doing. We explored two things: how well the neural net was doing compared to the original matrix, and how both the neural net and SVD were doing compared to the original matrix. We measured this in two ways:

- Error: Error is calculated by the formula,

  $$\text{Error} = \frac{\|A - A_k\|_2}{\|A\|_2},$$

  where A_k is the rank k approximation of the matrix A, and $\|A\|_2$ is the 2-norm of A. We calculated this error for both the NN and SVD approximations, and then compared the two hoping that they wouldn't be too off from each other. While the 2-norm was most likely to be enough, we also looked at the Frobenius norm. For almost every test we ran, the difference between the two was negligible.

- Visuals: Matrices can be plotted as images, giving us a more tangible sense of what the values of the matrix are in a more readable format than just printing out the values of the matrix. For our tests we plotted the NN approximation, next to the SVD approximation, next to the original matrix to compare. This was especially helpful when we were trying to approximate actual images. Figure 2 shows how the random matrix

  $$\begin{pmatrix} 0.41 & 0.40 & 0.60 & 0.85 & 0.06 \\ 0.46 & 0.30 & 0.97 & 0.51 & 0.37 \\ 0.77 & 0.99 & 0.93 & 0.51 & 0.98 \\ 0.41 & 0.41 & 0.72 & 0.94 & 0.94 \\ 0.71 & 0.88 & 0.01 & 0.89 & 0.47 \end{pmatrix}$$

  can be represented visually where high values correspond to the lighter parts in the image, and low values correspond to the darker parts of the image.

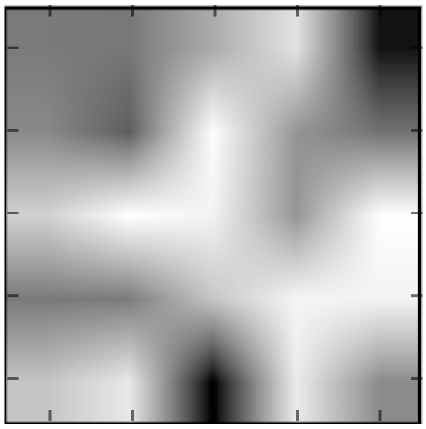

Figure 2: Random 5×5 matrix represented visually.

4.1 CORRECTNESS

As we started this project, we spent a long while running the tests without checking to make sure that our answer was the correct rank. As we were running the tests and only looking at the errors, we never noticed that in fact our original tests were all producing rank 1 approximations no matter what rank we told the network to approximate. While working on approximating images (see Section 5.4), I noticed that no matter the rank we told the machine to learn, it was always giving us the same result. This led to some rearranging of the neural network, and us implementing a rank check function that we ran every time to make sure that our approximations matched up with what we desired.

5 EXPERIMENTS

We explored many aspects of neural networks and matrices for this project. We explored how well the neural network would approximate a random matrix of dimension smaller than 10, how performance changed with special matrices (i.e. symmetric matrices, the identity matrix, and others), how well it approximated medium sized images and small images, and how the design of our neural net helped or didn't help the neural net approximate the matrix.

5.1 NETWORK DESIGN

The first thing we explored was design. We started with a neural net that had n nodes in the first and second layer, where n is the dimension of the $n \times n$ matrix. We then set the number of nodes in the third layer to be k where k was the rank of the low-rank matrix that we set. Finally the output layer was set to n nodes. See Figure 3 for an example where $n = 4$ and $k = 2$.

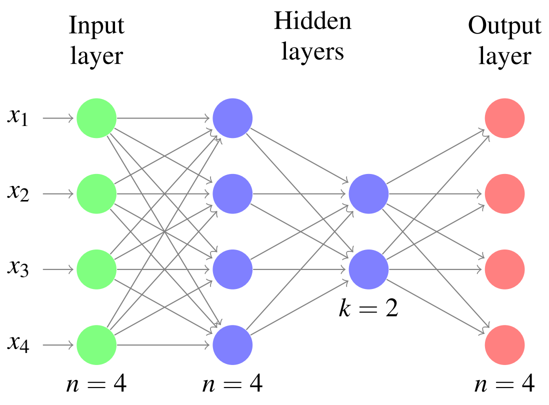

Figure 3: Network Design: n-n-k-n. Vector v with components x_1, x_2, x_3, x_4 to vector Av.

We tested multiple designs on a 4×4 matrix, with rank 2. Some data scientists argue that building the hidden layers so that they make their way down from the original size, to the rank by powers of 2 — as shown in Figure 4 — is optimal. Because this doesn't make much sense to test on a 4×4 matrix with rank k, we tested it on larger matrices. We found that not only was this not optimal, it often gave us approximations that didn't do what we wanted. Most commonly, we found that it made all of the approximations rank 1, instead of rank k.

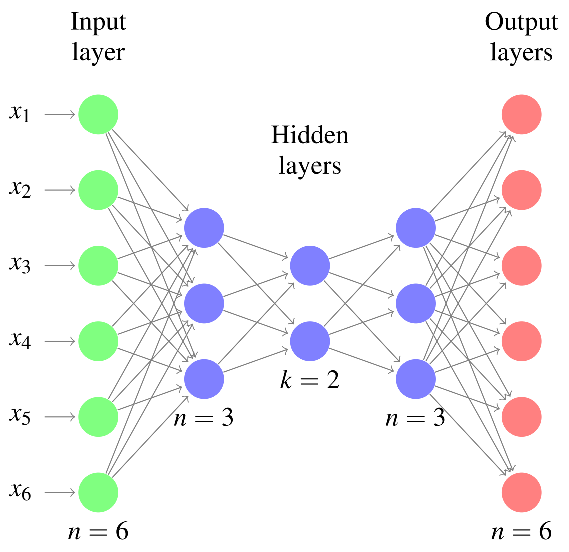

Figure 4: Network Design with decreasing/increasing hidden layer size to/from inner most hidden layer.

We also tried varying the structure in a less calculated way for the smaller matrices. We ran the neural net on networks with hidden layers that had size anywhere between n and k. To make our lives simple, we limited the number of hidden layers to 6 and tried many of the possible options. While there were variations, we concluded that there was actually no tangible difference. This led us to believe that using the original design, noted in Figure 3 was the best.

We concluded that there was no faster or more accurate design than the simple $n \rightarrow n \rightarrow k \rightarrow n$ design that we started with originally.

It is important to note that there are other things that can be edited in the network design that aren't just the number and size of layers. We can also edit the activation function for the network itself, and the initialization methods for each layer. While changing those variables most likely would have produced more intentional results, it was beyond the scope of our project. We stuck to the linear activation function and fiddled around with the initializations before settling with the initialization for Keras layers called "he_normal."

5.2 APPROXIMATING THE IDENTITY MATRIX

As a very simple way of testing the neural net, we subjected it to figuring out the identity matrix and it's low rank approximation. The identity matrix is defined as the matrix with all 1s down the diagonal

$$I = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

with the property

$$IA = A$$

for all matrices A. We ran the neural net on the identity matrix of size 50, looking for a different rank approximation.

The neural net ended up giving a very interesting result. Figures 5 & 6 show an attempt to do this approximation for rank $k = 20$.

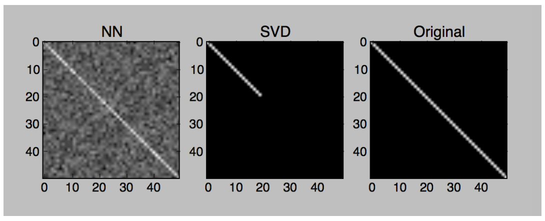

Figure 5: NN, SVD, and Original versions of 50 $\times$ 50 identity matrix with rank 20 approximation.

SVD Error (2-norm)	1.0
NN Error (2-norm)	1.00146298773
Error Difference	0.00146

Figure 6: Errors for matrix described in Figure 5.

It is clear that the neural network did not pick up on the most important part of the rank 20 approximation, i.e. the first 20 singular values of the matrix. Interestingly the difference between the error for the SVD and NN approximations was 0.00146 where each error was around 1.0. That is incredibly low for such a difference visually.

I then tried to approximate the matrix equal to the diagonal matrix that I call M_n where each entry down the diagonal is equal to the row number that that entry is in. This matrix looks like

$$M_n = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & n \end{pmatrix}$$

and the results of the neural net's approximation are shown in Figures 7 & 8.

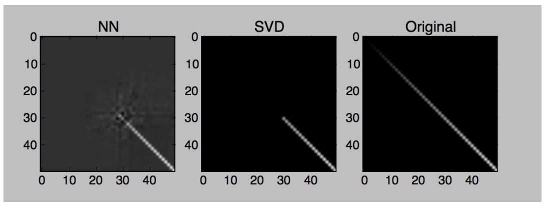

Figure 7: NN, SVD, and Original versions of 50 $\times$ 50 M_n matrix with rank 20 approximation.

SVD Error (2-norm)	0.591836734694
NN Error (2-norm)	0.641697317379
Error Difference	0.0499

Figure 8: Errors for matrix described in Figure 7.

This shows that the neural net is very good at approximating this matrix, at least visually. Even more interesting, however, is that the difference in the errors is actually higher than it was with just the identity matrix. This test yielded an error of 0.5918 for the SVD approximation, and an error of 0.6417 for the neural net approximation which gives a difference of 0.0499. This is somewhat expected, however, because this is a more complex matrix than the identity.

5.3 APPROXIMATING RANDOM LOW DIMENSION MATRICES

Through our tests, the neural net gave the most variation in results when we ran it on low rank approximations of low dimension matrices. The tests enumerated in this section focus on matrices of size $n = 10$ or smaller. We were able to successfully get the rank 1 approximation within an average error between the neural net approximation and the SVD approximation equal to 0.00046. Figure 9 shows the plot of a random matrix's rank 1 approximation done by the neural net and the SVD approximation.

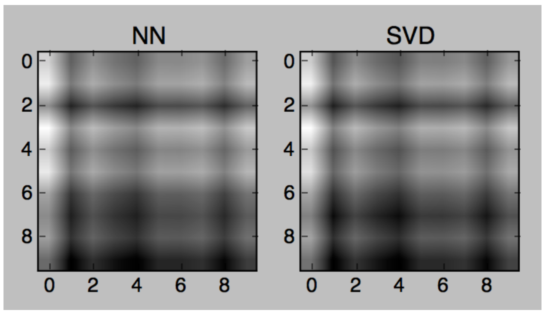

Figure 9: Rank 1 neural net and SVD approximation.

The tests that we ran on rank 1 approximations proved that the neural network was incredibly good at matching the SVD for $k = 1$.

As the rank got bigger the SVD failed. We saw that the disparity between the SVD and the NN approximations grew. Figure 10 shows the mean error difference between the two approximations from running the neural network on 10 random matrices for each rank approximation.

Rank 1	0.0005
Rank 2	0.0519
Rank 3	0.0773
Rank 4	0.0768
Rank 5	0.0923
Rank 6	0.0919
Rank 7	0.1351
Rank 8	0.1373
Rank 9	0.1550

Figure 10: Rank and absolute value of the difference in error between the neural net approximation and the SVD approximation for random 10×10 matrices.

Figure 11 shows a random 10×10 matrix approximated to rank 5 with it's neural net and SVD approximation side by side next to the original. It is clear that a rank 5 SVD approximation is much more accurate than a rank 5 neural net approximation.

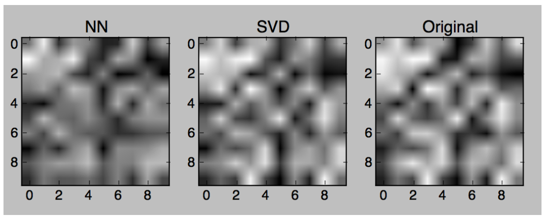

Figure 11: Rank 5 approximations of a 10×10 matrix and the original matrix.

5.3.1 SYMMETRIC MATRICES

We also ran this test on random symmetric matrices by generating random matrices and making them symmetric. This yielded the results displayed in Figure 12

Rank 1	0.0199
Rank 2	0.0478
Rank 3	0.0646
Rank 4	0.0658
Rank 5	0.1128
Rank 6	0.1238
Rank 7	0.1439
Rank 8	0.1279
Rank 9	0.1596

Figure 12: Rank and absolute value of the difference in error between the neural net approximation and the SVD approximation for random 10×10 matrices.

Figure 13 shows a random 10×10 symmetric matrix approximated to rank 5 with it's neural net and SVD approximation side by side next to the original. Like the non-symmetric version of this test, it is clear that a rank 5 SVD approximation is much more accurate than a rank 5 neural net approximation. However, the data in Figure 12 shows that the neural net was on average slightly more accurate at approximating symmetric matrices than random matrices, but not exactly all of the time.

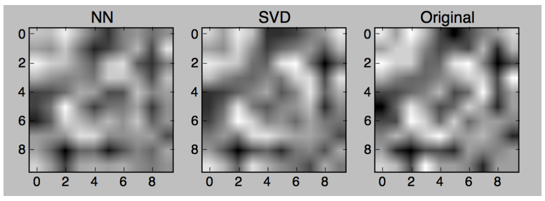

Figure 13: Rank 5 approximations of a 10×10 symmetric matrix and the original matrix.

5.4 APPROXIMATING IMAGES

5.4.1 BILL GATES

Finally we tried approximating images. We started with a 400×400 black and white verson of the image of Bill Gates shown in Figure 14. We struggled to approximate such a large matrix with the same accuracy as the singular value decomposition. We had to majorly up our processing power on the machine learning algorithms, and seriously up the number of time that we allowed for the neural net to learn. Although it doesn't matter much, for comparison's sake that change meant that we went from training the neural net on 10,000 random vectors during our tests in Section 5.3 and ran 10 "epochs" (a machine learning term that is really irrelevant to the understanding of this problem) to 100,000 and the number of epochs to 100 for the Bill Gates image. And still yet this produced results with a very large error and an unrecognizable image. The results of that test are shown in Figure 15 where we tried a rank 10 approximation.

Figure 14: Bill Gates 400 × 400 pixel matrix.

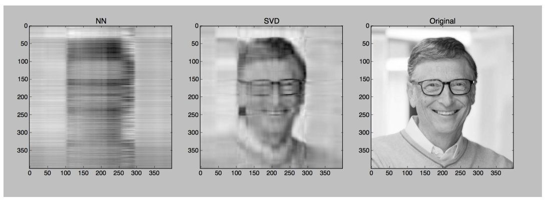

Figure 15: Bill Gates rank 10 approximation.

We were, however, able to very accurately reproduce the rank 1 approximation of the image. Figure 16 shows visually how accurate the rank 1 NN approximation is compared to the SVD. The difference in the errors was 0.0013. This, interestingly, is orders of magnitude better than the rank 10 approximation. This shows that our neural net is incredibly good at learning rank 1 versions of matrices, but still has work to do to better learn higher rank versions.

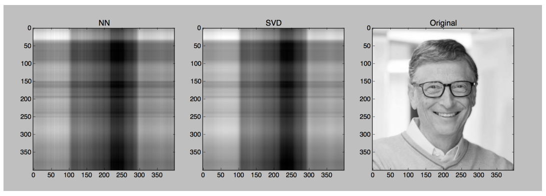

Figure 16: Bill Gates rank 1 approximation.

When approximating larger matrices like images, we learned that it was important to closely monitor the neural network's progress. As mentioned before, we had to up the work that the net was doing in order to get anything nearly accurate. In order to track the progress, we plotted the "loss" from epoch to epoch and plotted it on a graph. The results were extremely insightful. We saw how the neural net would make breakthroughs, and then stagnate for a while before making more breakthroughs. The more we allowed the net to run and learn through these breakthroughs, the better results we saw. Unfortunately we didn't have the time or the processing power to let the net run as long as it might have taken to see if it could actually get the Gates image perfectly. It is likely, though, that it would eventually get it with enough time to make these "breakthroughs." Figure 17 shows the loss that the neural net went through from epoch to epoch and shows how the network learned when we ran this on the Bill Gates image. Unfortunately, the neural net did not make any major breakthroughs and thus did not accurately approximate the image. This is in fact evident from the plot. Figure 20 on the other hand, shows how the neural network can in fact make progress and learn. (Note: a lower loss indicates a higher accuracy to the original matrix.)

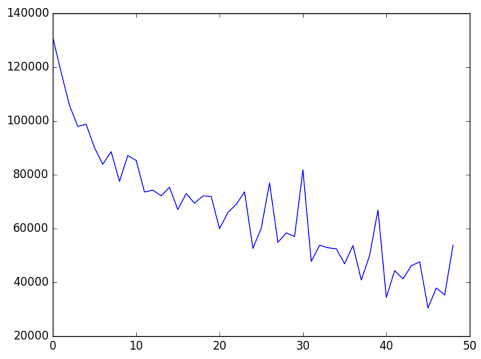

Figure 17: Bill Gates loss plot. Y-axis: loss. X-axis: epochs.

5.4.2 APPLE LOGO

Seeing as we didn't have much luck approximating Bill Gates, we decided to try a smaller image. Using a 256 × 256 pixel image of the Apple logo (see Figure 18), we put our neural network to work. This yielded much better results than it did with Bill Gates. The main reasons for this is that the image is much smaller than the Bill Gates image and far less complex. Figure 19 shows the results from our test on approximating a rank 10 version of the original matrix. It is evident that the neural net is not as accurate as the SVD, but this is to be expected for such a large matrix. The visual accuracy of the NN approximation is interesting to say the least. The loss plot for this run is shown in Figure 20 and shows how the neural network makes advances and learns as it goes. The breakthroughs around epoch 10 and 30 are highly significant to the final result. It is important to note as well that the neural net's error in it's approximation of the Apple logo was 0.02859 while the SVD's error was 0.01585. The difference between these two is 0.01274. In the grand scheme of this project, that result is very good.

Figure 18: Apple logo: 256 × 256 pixel image.

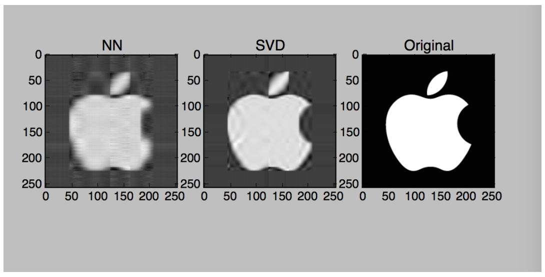

Figure 19: Apple logo results.

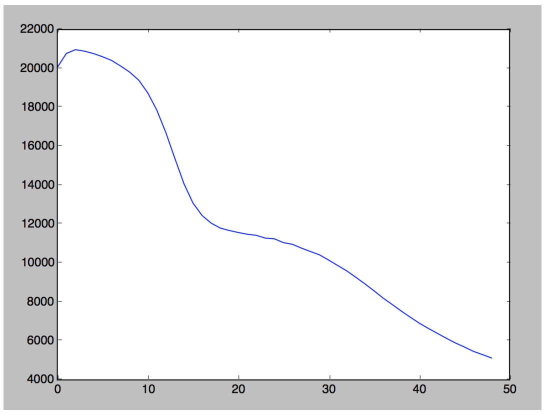
Figure 20: Apple logo loss.

5.4.3 COLOR WHEEL

As we went down in size of images, the quality increased rapidly. Figure 21 shows a color wheel image. This image was 10 pixels by 10 pixels (my apologies for the low quality). We tried to get a rank 5 approximation of the image.

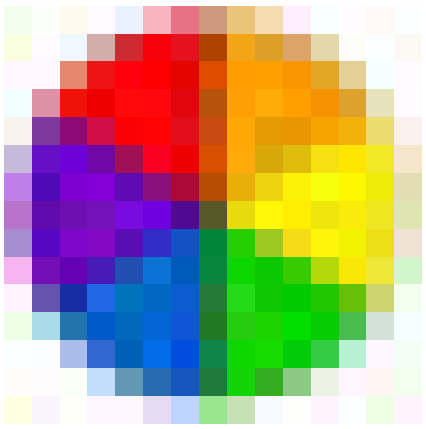
Figure 21: 10 × 10 pixel image of a color wheel.

Figure 22 shows the results of the neural network's approximation. While not perfect, it is clear that it is getting the right idea. The difference in errors for this test was 0.0324 and Figure 23 shows the loss for the neural net over 30 epochs.

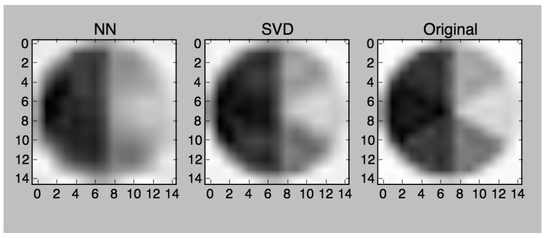

Figure 22: Rank 5 color wheel approximation.

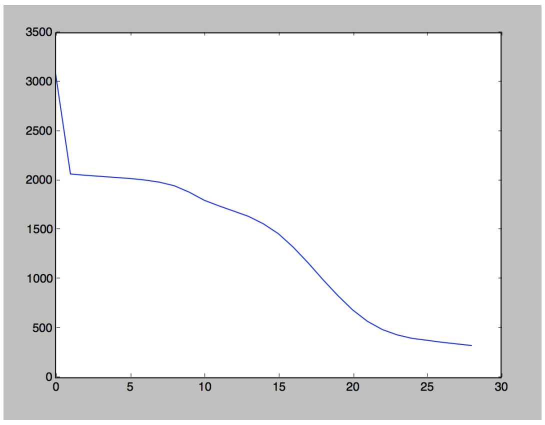
Figure 23: Color wheel loss from rank 5 over 30 epochs.

Here, there is a more gradual learning that isn't broken up and as steep as it was with the Apple logo.

5.5 NEURAL NETWORK LEARNING

As mentioned in the previous section, increasing the number of "epochs" was crucial to pushing the learning of the neural net past a certain level. The question remains, why? We have set up our neural network to improve based on the "mean-squared-error," or MSE, between runs. This means that as the network is learning, it is changing values within it's black box of "learning" to decrease the MSE from run to run. If you look carefully at the figures that show loss, you'll notice that the larger the matrix the higher the loss. This is because the loss is proportional to the size of the matrix. This also means that a "good" approximation might not necessarily correspond to a low loss. It will, however, correspond to a *lower* loss than the original loss from the first run.

So as we increase the number of epochs we see that the network begins to learn more and more. Another thing that affects loss from run to run is the activation method of the layers, as well as the initializing functions for the layers. Knowing and purposefully manipulating this was outside the scope of this project, but could prove useful in future iterations of this work.

6 CONCLUSION

Overall, the overwhelming sentiment from this project that arose is that it is not worth it to use a neural network to approximate a matrix. That said, our results were very exciting. The fact that the network can almost perfectly approximate a rank 1 approximation of a matrix of any size or complexity is incredible as shown in Figure 16. And for smaller matrices, we can accurately approximate low rank versions of matrices. We also learned that it was no easier for the neural net to approximate symmetric matrices, but it was easier to approximate some diagonal matrices but not all.

We also found that the design of the neural net, with no further insights than we currently have, doesn't improve the approximations the more complex it gets. In fact it mostly got worse. That is in part due to our lack of knowledge and experience designing neural networks. One thing to take away from this project is that is almost certainly possible to design a neural net that will accurately mimic the SVD approximation of any matrix. It just so happened that the one that we worked with was adequate for most random matrices, and very good for getting rank 1 approximations.

I'm confident that given the time to get to really know the neural network construction and time to experiment, we could build a neural network that could accurately represent the SVD approximation of any possible matrix.