CSE 573: Computer Vision and Image Processing

Homework 4

Submitted by
-Ketan Shah
Person #: 50247131

Q 1. How do Autoencoders detect errors?

Ans: Autoencoders take a multi-dimensional input $\mathbf{x}$ such as in our case each image is converted into a 784 dimension vector (values of which are obtained by aligning the 28 by 28 pixel values) and gives a hidden representation $\mathbf{y}$, called the code which can be of a different dimension.

$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}_1)$$

Here $\mathbf{W}$ is the weight and $\mathbf{b}_1$ is the bias. σ is the no linear representation typically a sigmoid function.

This representation y is again mapped back to the input x by the following formula:

$$\mathbf{x}_{recon} = \sigma(\mathbf{W}'\mathbf{y} + \mathbf{b}_2)$$

Here the W' is W^T. Now the error is minimized by taking the sum of squared error between the original x and the reconstructed x and accordingly the weight vector W and biases are adjusted. For our model we use cross entropy error function $-\sum_i p_i \log p_i$ which gives better performance.

Now for building the neural network we stack multiple autoencoders like this to improve the accuracy of the whole model.

For example to obtain an output which is a vector of size 10 representing 10 classes one of each digit we can stack 2 autoencoders, from 784 to 100 dimensional representation, then from 100 to 50 and finally a softmax function to convert the 50 dimensional input to 10 dimensional output.

Now after we stack the autoencoders the error that was minimized earlier while forming each encoder separately still does not give optimal results. Therefore, we reduce the error of the whole model by using back propagation

Q 2. The network starts out with 28 × 28 = 784 inputs. Why do subsequent layers have fewer nodes?

Ans: This is done to actually reduce the dimensionality of the input and effectively produce a compressed representation of the system. If we keep on increasing the number of nodes at each layer, it will increase the computations and the running time of the whole model and it will take longer to train the model.

This is not mandatory and there are cases where there have been interesting observations when the number of neurons in the subsequent layers is more than the preceding layer but for the compressed representation we reduce the size each layer.

CSE 573: Computer Vision and Image Processing

Homework 4

Submitted by
-Ketan Shah
Person #: 50247131

Q 3. Why autoencoders are trained one hidden layer at a time?

Ans: This is actually called the pre training step. What we do is before stacking the auto encoders we train each encoder at a time by simply encoding and reconstructing the input by decoding and minimizing the error and adjusting the weights. The weights selected by this method give a much better result in the whole model and this reduces the overall training time of the model. As mentioned by Geofrey Hinton without pre training the deep auto-encoder always reconstructs the average of the training data even after prolonged fine tuning of the system. Through pre training we actually try and initialize the starting weights of the system to be close to a good solution. This increases the performance of the system greaty compared to randomly initializing weights and training the whole model.

Q 4. How were the features in Figure 3 obtained? Compare the method of identifying features here with the method of HW1. What are a few pros and cons of these methods?

Ans: The features in the Figure 3 are actually the weights decided for the hidden layer representation. It can be obtained by the function `plotWeights(autoenc1)`

These weights are actually obtained when we form the first encoder. Each image in there is 784 pixel image and there 10 x 10 images .So when each weights is multiplied by an input it gives out a 100 dimensional output.

The features here decide which part of the image are more important than other. This method of feature extraction is different from HW1 where we used edge and corner detection methods to extract the features.

The advantage of auto encoder is that we do not have to worry about what features to extract exactly and the model does not change much according to the input. If we selected some other types of images, the model will train itself better to extract the corresponding feature. However, in the method of HW1, we have to decide on many things of what feature to extract and how many features and all and once the type of images change all these values will have to change again.

Now the con of the auto encoder method is that the features are not very meaningful in the sense that we cannot really make out exactly what they are extracting whereas in the HW method we knew exactly what features of shapes we are extracting. The Hw1 method requires a lot of time to train if the images are very complex and have many intricacies to be taken care of. The neural network method takes care of the complexity within the model and we do not have to worry about that.

CSE 573: Computer Vision and Image Processing

Homework 4

Submitted by
-Ketan Shah
Person #: 50247131

Q 5. What does the function plotconfusion do?

Ans: This function actually plots the confusion matrix of the actual target output versus the predicted outputs. Each element E_{ij} where j is the column number and i is the row number represent the percentage of output that should have been predicted as j class but were predicted as i class. Therefore, the diagonal values of this matrix give the percentage of the outputs correctly predicted. In addition, the last value of the matrix that is the bottom right hand corner value give the total accuracy and the error percentage of the system.

Q 6. What activation function is used in the hidden layers of the MATLAB tutorial?

Ans: The activation function used in the hidden layers is the sigmoid function. Since we want our hidden representation to have values between 0 and 1 we choose the sigmoid function. There are other activation functions as well like the tanh, RelU that can be used. The sigmoid function is as follows:

$$\boldsymbol{\sigma = \frac{1}{1 + e^{-y}}}$$

Q 7. In training deep networks, the ReLU activation function is generally preferred to the sigmoid activation function. Why might this be the case?

Ans: The RelU function is generally preferred over the sigmoid function. The first problem with the sigmoid function is that it causes the gradients to vanish. If the neurons activation output saturates to either 1 or 0 the gradients of these values are really small. This local gradient is multiplied to the previous output during back propagation. Therefore, if the local gradient is small it slowly vanishes over the previous layers until it reaches the input. This makes it difficult to adjust the weights effectively. In addition, the outputs of sigmoid are not zero centered which makes the gradient output either all positive or negative. This makes the gradient step too big in either direction making it difficult to reach the local minima easily. It can be used on the final layer though since we want our output to be either 1 or 0 to be able to predict the class.

To solve these problems the RelU (rectified linear unit) activation function is used. It has 2 advantages. Firstly, it is simpler to compute compared to sigmoid. It is simply max(0, y). This makes it faster. Secondly, it avoids the vanishing gradient problem.

CSE 573: Computer Vision and Image Processing

Homework 4

Submitted by
-Ketan Shah
Person #: 50247131

Q 8. The MATLAB demo uses a random number generator to initialize the network. Why is it not a good idea to initialize a network with all zeros? How about all ones, or some other constant value? (Hint: Consider what the gradients from backpropagation will look like.)

Ans: If we initialize all the weights with all zeros or all ones or some constant value what it means that the output at each neuron will be equal. This in turn means that the partial derivatives will be equal too and then the derivative of the cost function will be symmetrical like this. Now this means that after each step of the gradient descent all the weights will be updated but in a symmetrical manner. That means again each weight will end up being similar. This way reaching the local minima is very difficult. Hence we initialize the network with all zeros or ones or any constant values.

Q 9. Stochastic gradient descent vs. batch gradient descent.

Ans: The advantage of stochastic gradient descent is that it goes through each single input and only updated the weights when a new minimum is reached. This way the local minima is reached closely. In batch gradient descent, we average out all the values in the batch so now the values, which were not minimizing the weights, are also being considered in the calculation. Therefore, the answer from batch method might not be the minima.

The batch gradient descent is however faster in calculation because it has lesser number of iterations. To tackle the problem with batch gradient descent mentioned above what we do is take smaller sizes of batches and take random samples from the total number of samples and then increase the number of epochs. This method is faster than stochastic in terms of epochs.

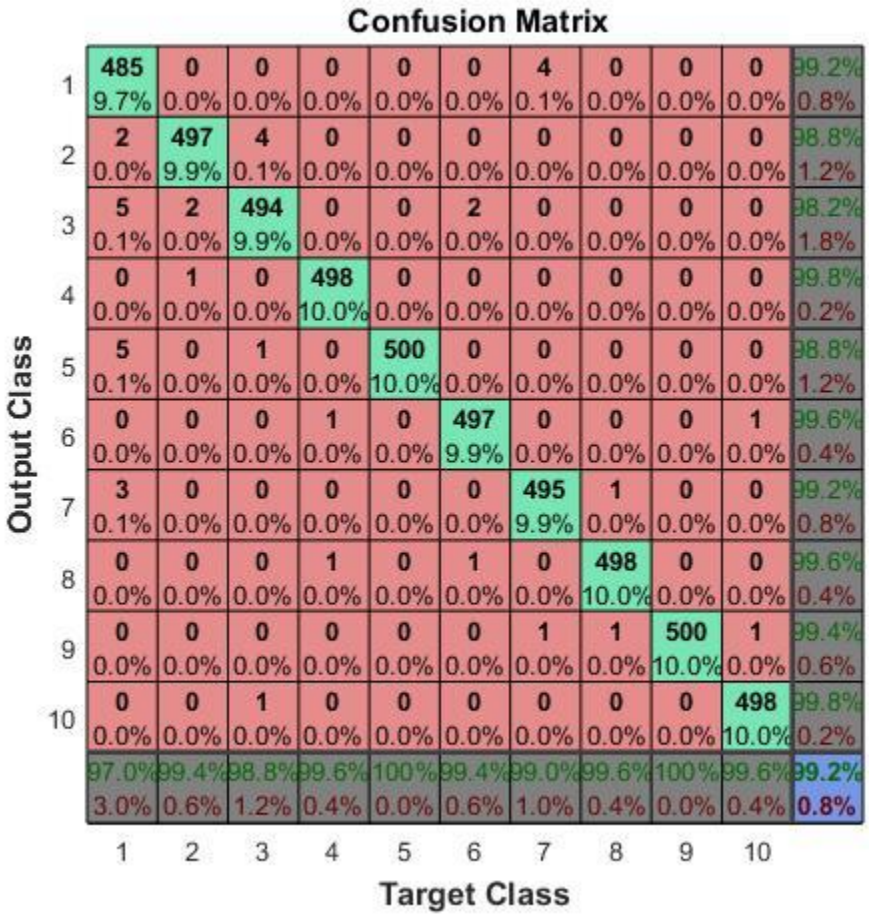
CSE 573: Computer Vision and Image Processing

Homework 4

Submitted by
-Ketan Shah
Person #: 50247131

Q 10. Exploration

- **Changing no. of nodes**
  I tried increasing the no. of nodes in the first hidden layer to 1000 to check the results what happens when we keep the numbers in the hidden layers more than the previous layer. The model gives a 99.2 % accuracy. However, this is not much improvement since I was able to achieve that with initial 784-100-50-10 setting too. Therefore, for our dataset it is of not much use to increase the no of nodes. It only increases the run time of the system. Following is the confusion matrix for it.

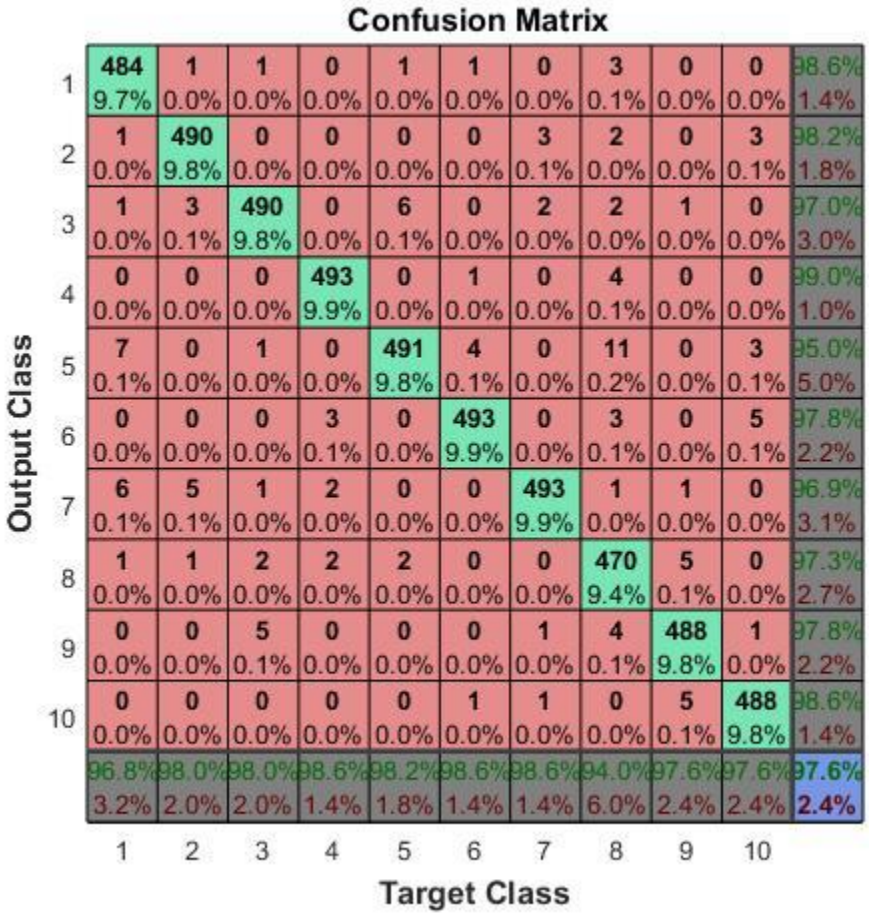


Next, I tried decreasing the number of nodes. I tried the following setup 784-50-25-10.
This does affect the performance and the accuracy comes down to about 97 %. See the confusion matrix below.

CSE 573: Computer Vision and Image Processing

Homework 4

Submitted by
-Ketan Shah
Person #: 50247131



- **Changing SparsityProportion**

  Lets try changing the **SparsityProportion** parameter. The tutorial says that this should ideally be between 0 and 1. I tried different values, some closer to 1 and some to 0. Here are the accuracies I got with some of values.

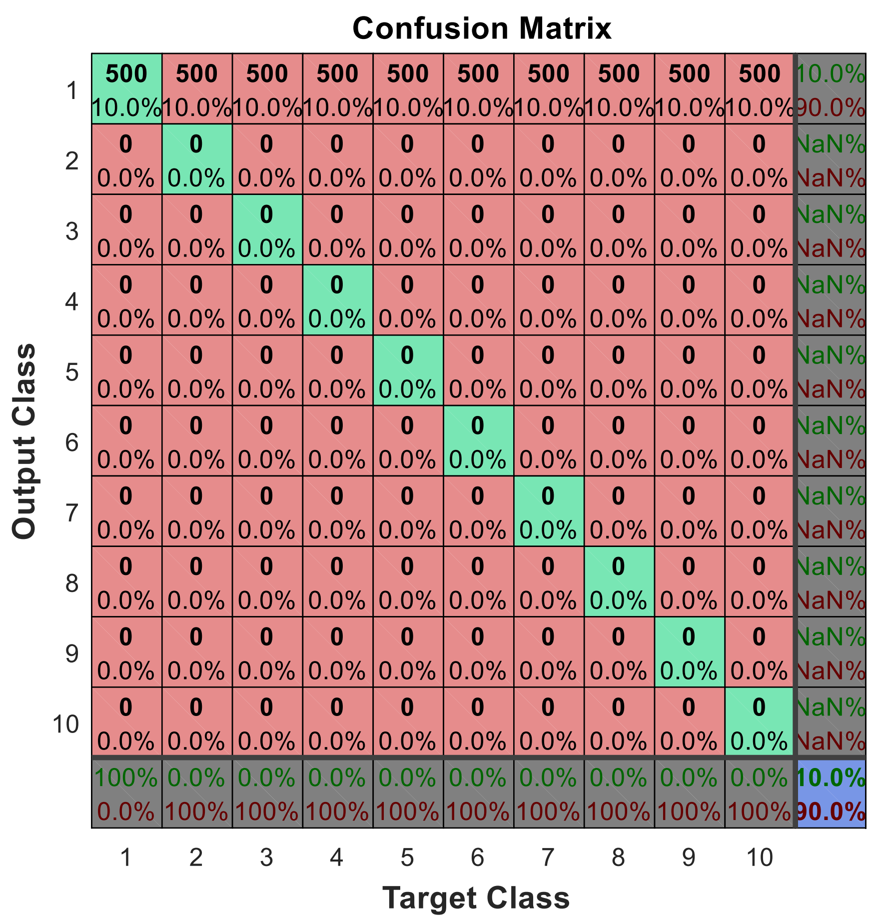
Value	Accuracy(%)
0	99.2
0.05	98.3
0.5	97.72
0.9	87.3

CSE 573: Computer Vision and Image Processing

Homework 4

Submitted by
-Ketan Shah
Person #: 50247131

- **Changing L2WeightRegularization**
  The tutorial mentions that this parameter should ideally be very small. I tried increasing it to a value of 0.1 and found that the model performs very badly. It predicts every class as class 1.
  It gives 10% accuracy which means the model doesn't do anything.



- **Changing SparsityRegularization**
  Changing this parameter didn't really change anything. I tried 4 difference values like 0, 9, 50, 100.
  The accuracy remained the same.

CSE 573: Computer Vision and Image Processing

Homework 4

Submitted by
-Ketan Shah
Person #: 50247131

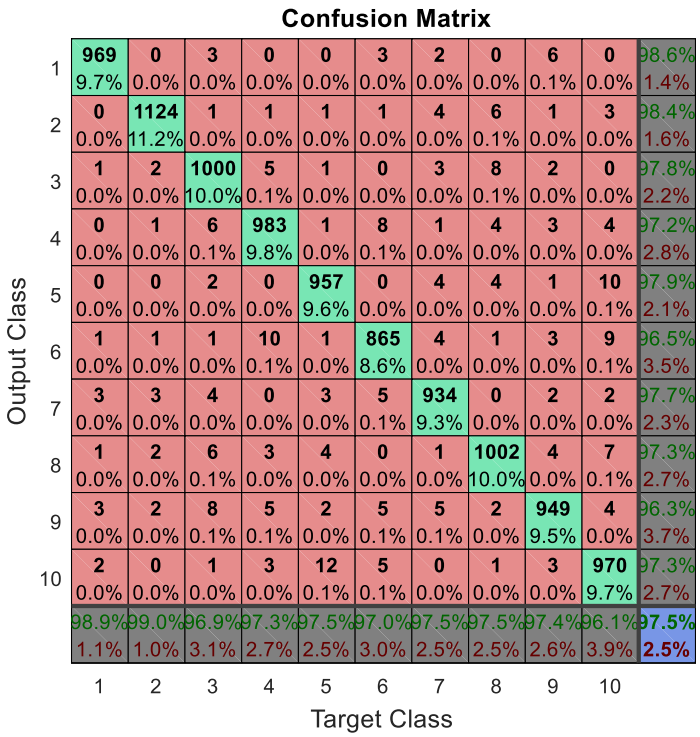
- **Changing MaxEpochs**
  I reduced this parameter to see if anything changes. Turns out if I reduce the max epochs in the single autoencoder tuning levels the model till works fine but if I reduce the max epoch for training the deep autoencoder then accuracy gets affected. Ofcourse if the max epochs are reduced tremendously the model doesn't really get trained much.

The tutorial presentation is pretty robust in my opinion. The model was easily tranferrable to mnist dataset with minimal changes. Also the parameters selected give good results in both models.

Extra Credit

I performed the same tutorial on the MNIST dataset. I've attached the file autoencoder_mnist.m in the zip folder which contains my code related to this. I tried 2 variations.

1st is to run the same model as we did for synthetic dataset and see the results. I got a 97.5 % accuracy for this model on the mnist dataset. So the same model performs worse on the mnist dataser Here is the confusion matrix for that :



CSE 573: Computer Vision and Image Processing

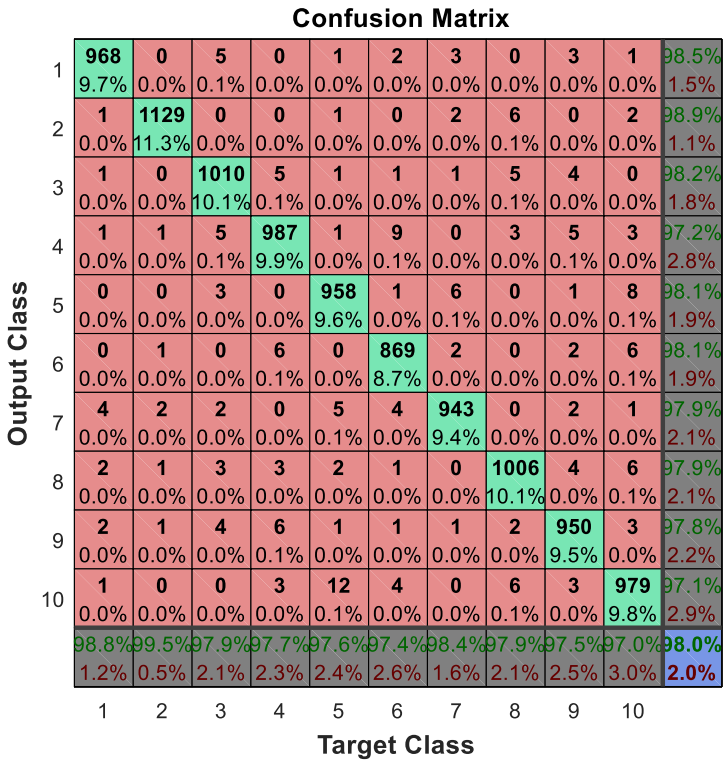
Homework 4

Submitted by
-Ketan Shah
Person #: 50247131

Now to improve the accuracy I used a 3 hidden layer model. The sizes were 784-500-200-50-10 from input to output.

With this model I get a 98.0 % accuracy.

Following is a confusion matrix for the 3 hidden layered model

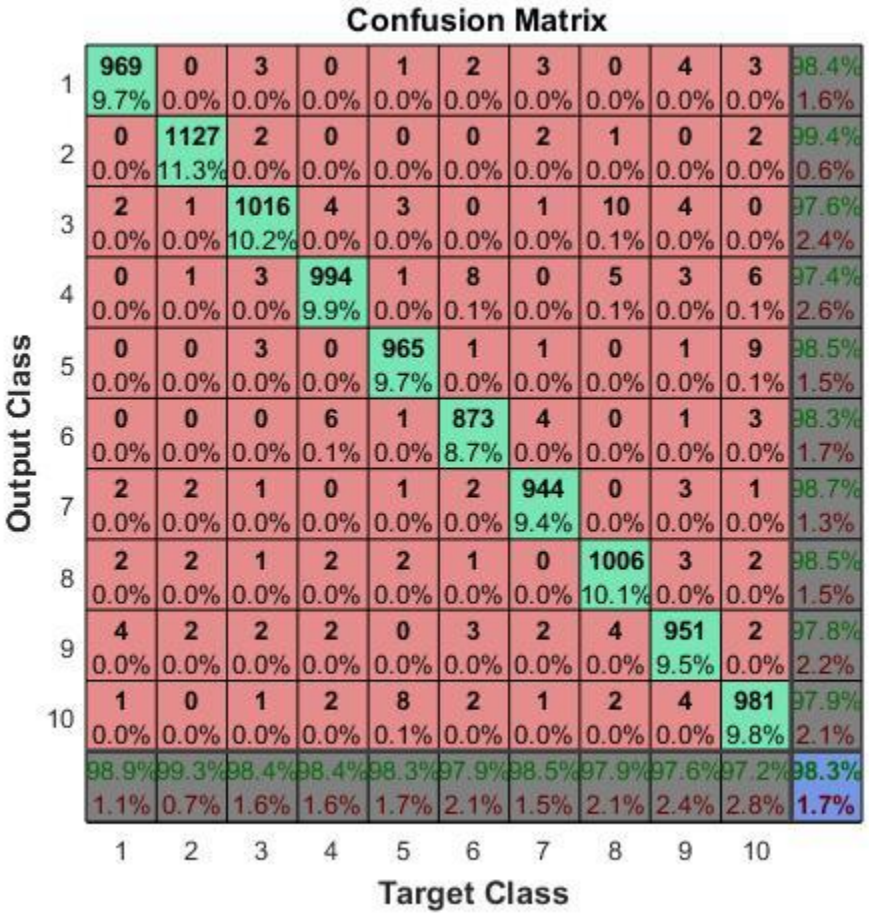


CSE 573: Computer Vision and Image Processing

Homework 4

Submitted by
-Ketan Shah
Person #: 50247131

I also used a model with the following layers 784-200-100-10 and I got an accuracy of 98.3 %. Below is the confusion matrix for it



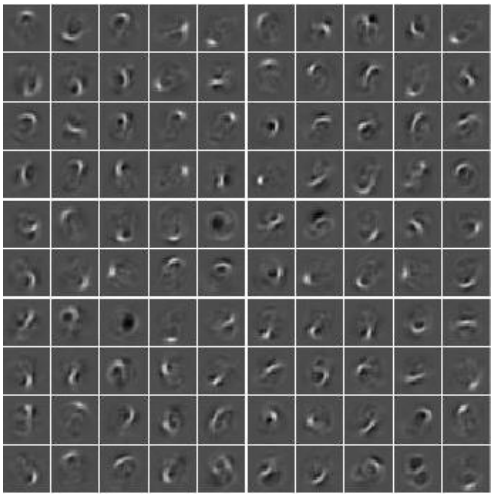
CSE 573: Computer Vision and Image Processing

Homework 4

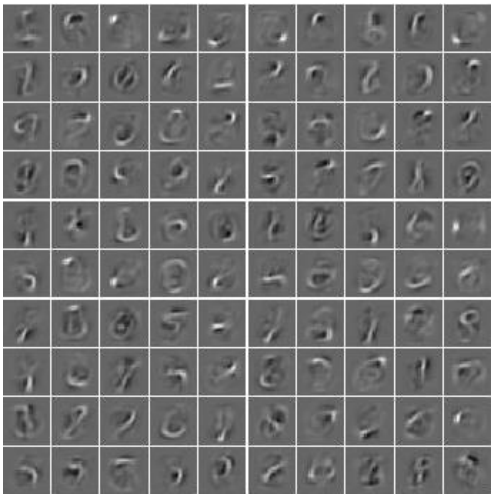
Submitted by
-Ketan Shah
Person #: 50247131

Below are the features extracted from the first hidden layer for synthetic data and the mnist data

The comparison is when I applied the same model to both datasets.



Synthetic data features



Mnist data features

They are certainly not similar in shape but in structure they seem to be doing the same thing. You'll also notice similar shapes at different positions in both the images. So their net effect seem to be similar.

Submitted work:

1. stacked_autoencoder.m – code related to the tutorial for synthetic images
2. autoencoder_mnist.m – code relate to making a autoencoder neural network for mnist dataset.

References:

- Artificial Intelligence and Statistics, pages 249–256, 201
- ai.stanford.edu/~quocle/tutorial2.pdf
- https://iksinc.wordpress.com/2016/07/07/autoencoders/
- https://en.wikipedia.org/wiki/Backpropagation
- https://www.youtube.com/watch?v=qCWWDyOS9Xo

CSE 573: Computer Vision and Image Processing

Homework 4

Submitted by
-Ketan Shah
Person #: 50247131