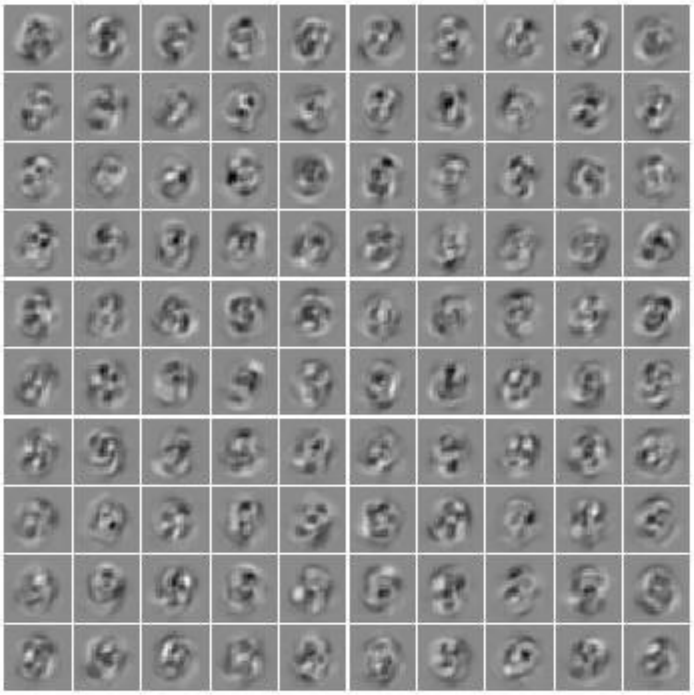
Project Report - Autoencoders for Image Classification

1. How does an auto-encoder detect errors?
   - ➔ The auto-encoder is comprised of an encoder and a decoder. The encoder maps the input to a hidden representation and the decoder tries to reconstruct the original image using the output of encoder. The cost function measures error between the input to encoder and its reconstruction, output of decoder. This could be sum of squared errors, auto-encoders tries to minimize the reconstruction error.

2. The network starts out with 28 × 28 = 784 inputs. Why do subsequent layers have fewer nodes?
   - ➔ Dimensionality reduction facilitates the classification, visualization and storage of high-dimensional data. Each layer helps to identify import feature in the image, such as the first layer has learned to detect edges in the image at different positions and orientations in the image.
   - ➔ So each layer of nodes trains on a distinct set of features based on previous layer's output. The more layers we add, the more complex features the nodes can recognize as they sum up with features learnt from the previous layer.
   - ➔ If the number of nodes matches the size of the inputs then autoencoder will replicate the image and this will result in over fitting. Also, the model will not be able to predict properly on unseen data.

3. Why auto-encoders are trained one hidden layer at a time?
   - ➔ The auto-encoder is comprised of an encoder and a decoder. The encoder maps the input to a hidden representation and the decoder tries to reconstruct the original image using the output of encoder.
   - ➔ Autoencoders is an unsupervised training model. So the features learnt in the first hidden layer are used in the second layer to learn patterns learnt in the first layer.
   - ➔ This way the autoencoders learn important features in each layer and can provide better generalization.

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?
   - ➔ The features in figure 3, attached below, are obtained by encoding the input size of 784 to 100 nodes.



- ➔ In homework 1, the features were extracted using 3 different filter banks namely, Gaussian, Laplacian of Gaussian, and derivate of Gaussian. The filters were used to detect neighborhood pixel values by smoothing or blurring, detecting edges.
- ➔ The detected features were then grouped together using K means to form bag of words and mapping.
- ➔ Autoencoder does not use specific filters as we used in homework 1.
- ➔ Spatial pyramid matching using bag-of-words approach had a less accuracy in comparison with autoencoders. Multiple autoencoders can be implemented to learn different features of the image; this was not done in homework 1.
- ➔ Autoencoders are widely used for reducing the dimension and extracting features from the image.

5. What does the function *plotconfusion* do?
   - ➔ It is used for plotting classification confusion matrix. On the confusion matrix plot, the rows correspond to the predicted output and the columns show the expected output (label). The diagonal cells in the confusion matrix display the accuracy (in percentage) that the trained network correctly predicts the estimated output.
   - ➔ The other diagonal shows the incorrect classification done by the network.

Activation Functions:

1. What activation function is used in the hidden layers of the MATLAB tutorial?
   - ➔ The tutorial uses sigmoid activation function.

2. In training deep networks, the *ReLU* activation function is generally preferred to the sigmoid activation function. Why might this be the case?
   - ➔ Rectified linear Unit (ReLU) function is given by, **A** = max(0, **y**) where **y** = xW + b
   - ➔ The benefit of using ReLU is that of reduced likelihood of the gradient to vanish that is when y > 0. In this region the gradient remains constant. Whereas logistic sigmoid has the problem of vanishing gradient as the input increases or decreases. Due to constant gradient, the learning becomes faster, i.e it converges to the minima quickly.
   - ➔ One more important benefit of ReLU is sparsity, this arises when **y** < 0. The more such outputs occur the layer becomes sparser in representation. Logistic sigmoid always generates some non-zero output which results in dense representation.
   - ➔ Also, ReLU does not saturate In the positive region, it is computationally efficient and converges faster than sigmoid.

3. 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 back propagation will look like.)
   - ➔ According to back propagation algorithm, partial derivate of weights in a particular layer is computed by using the formula = activation * delta difference
   - ➔ So if the weights are set to zero then it remains zero for all iterations. This prevents the activation from increasing or decreasing.
   - ➔ Zero weights basically block the error from propagating and ultimately don't allow learning from error.

Training Loop

1. Give pros and cons for both stochastic and batch gradient descent. In general, which one is faster to train in terms of number of epochs? Which one is faster in terms of number of iterations?
   - ➔ Gradient descent computes the gradient using the whole dataset. This is great for convex or smooth error manifolds.
   - ➔ Stochastic gradient descent (SGD) computes the gradient using a single sample. One benefit of SGD is that it is computationally faster than gradient descent. This is because an entire dataset cannot be stored in RAM which makes vectorized operation much slow and inefficient. Mini-batch SGD is intentionally small and it is computationally quicker than batch gradient descent.
   - ➔ In gradient descent, the weights are updated after entire pass of the data, thus larger the training set the slower our algorithm will update the weight and it will take more time to

converge. Whereas, in case of stochastic gradient descent, the weights are updated after each training sample. Hence it is faster than batch gradient descent.

- In SGD, because it uses only one sample of data at a time, its path to converge to minima is more random in nature than that of the batch gradient descent. But, SGD will surely converge to minima if the cost function is convex in nature.
- So to conclude, faster in terms of number of epochs to converge is batch gradient descent as it has more training samples. While SGD will be faster in terms of number of iterations.

Exploration

1. Try playing around with some of the parameters specified in the tutorial. Perhaps the sparsity parameters or the number of nodes; or number of layers. Report the impact of slightly modifying the parameters. Is the tutorial presentation robust or fragile with respect to parameter settings?
    - Reducing the number of neurons reduces the accuracy of classification. This is expected.
    - Added one more layer of autoencoder and the accuracy reduced drastically. This is because the data was trained perfectly and resulting to over fitting.
    - Also, allowing the data to scale did not have much effect on the accuracy.
    - Certain parameters were modified and their observation were noted as mentioned below

Hidden Layers	2	2	2	3
Number of Neurons	100, 50	50, 25	50, 25	100, 50, 25
Epochs	100	100	400	100
Accuracy	**99.2**	**98.8**	**98.1**	**49.9**

- From the above we can see that best accuracy obtained was 99.2, keeping all parameters same and changing the sparsity proportion to 0.5 gave an accuracy of 99.5%. So earlier the value was 0.1 and due to which each neuron in the hidden layer was trained to give high output for small number of training examples.
- The default transfer function for encoding and decoding is logistic sigmoid.
    - First changed Encoder function to '**satlin**', accuracy = 98.8 %
    - Now, both encoder & decoder function were changed to '**satlin**', accuracy = 99 %
- Finally modified the loss function to be Mean squared error instead of the default, cross entropy. The accuracy of the change dropped to 97.1%

Extra Credit

- Loaded the images and labels. One difference with digit example and MNIST data is that, digit example has the data loaded in a different way and also the labels are in a one hot vector format. Need to convert MNIST data in the format as used by the example.
- Number of training images = 60,000
  Number of Testing images = 10,000

- Below is the comparison for Classification of synthetic images and MNIST images

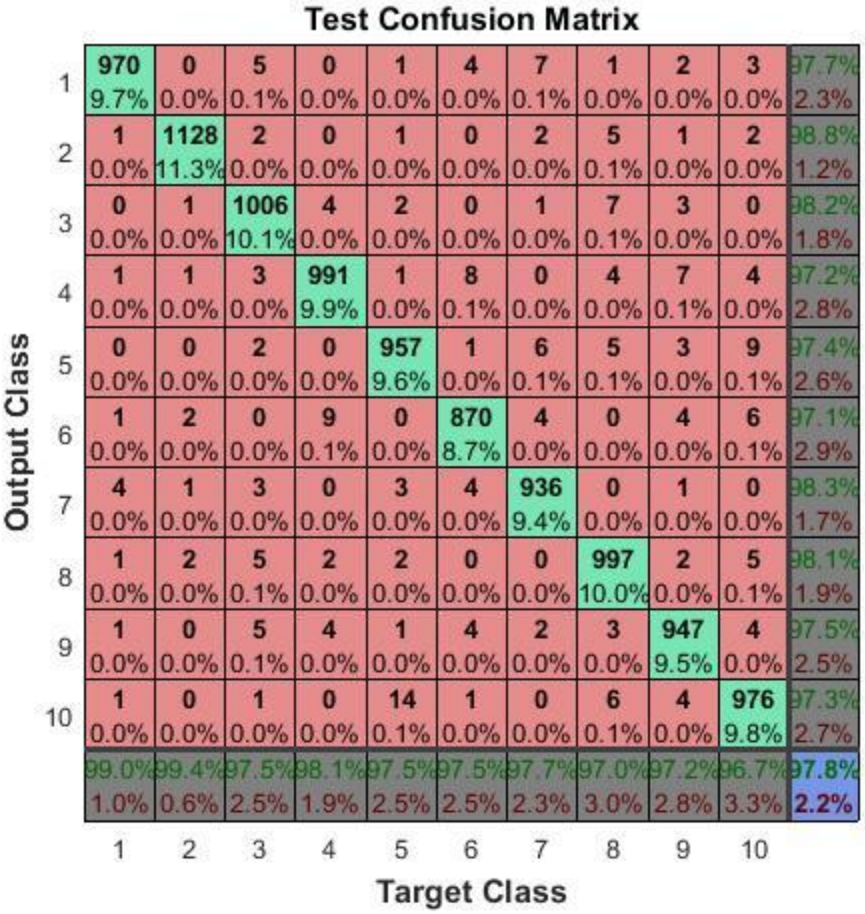
Data	Synthetic	MNIST
Hidden Layers	2	2
Number of Neurons	100, 50	100, 50
Epochs	100	100
Accuracy	**99.2**	**97.7**

- Parameters were modified for MNIST data also, the observations are given below

Hidden Layers	2	2	2	3
Number of Neurons	100, 50	50, 25	50, 25	100, 50, 25
Epochs	100	100	400	100
Accuracy	**97.7**	**96.9**	**96.8**	**82.7**

- From the above we can see that best accuracy obtained was 97.7, keeping all parameters same and changing the sparsity proportion to 0.5 gave an accuracy of 97.8%.
- As you increase the number of neurons, model will be able to capture more features, but if you capture too many features, then you end up over fitting your model to the training data and it won't do well with unseen data. This is what happened when I increased Number of neurons to 300.
- As done for Synthetic images modified the encoding & decoding transfer function and their observations were noted,
  - First changed Encoder function to ‘**satlin**’, accuracy = 97.1 %
  - Now, both encoder & decoder function were changed to ‘**satlin**’, accuracy = 97.5 %

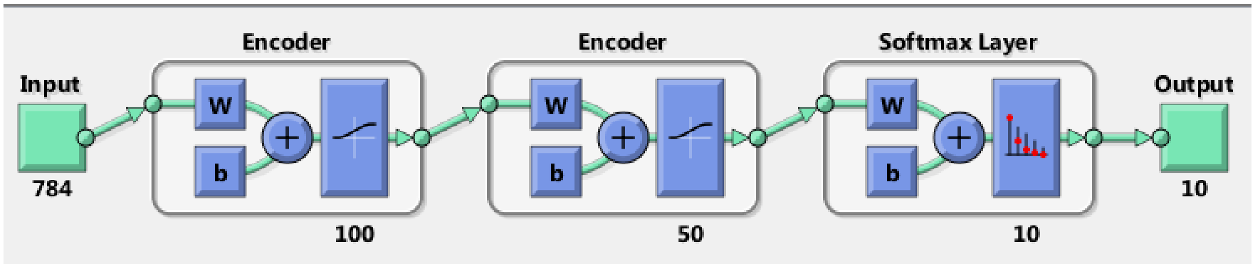
- The confusion matrix for best accuracy on MNIST data is attached below, 4)



Test Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	6	7	8	9	10	
1	970 9.7%	0 0.0%	5 0.1%	0 0.0%	1 0.0%	4 0.0%	7 0.1%	1 0.0%	2 0.0%	3 0.0%	97.7% 2.3%
2	1 0.0%	1128 11.3%	2 0.0%	0 0.0%	1 0.0%	0 0.0%	2 0.0%	5 0.1%	1 0.0%	2 0.0%	98.8% 1.2%
3	0 0.0%	1 0.0%	1006 10.1%	4 0.0%	2 0.0%	0 0.0%	1 0.0%	7 0.1%	3 0.0%	0 0.0%	98.2% 1.8%
4	1 0.0%	1 0.0%	3 0.0%	991 9.9%	1 0.0%	8 0.1%	0 0.0%	4 0.0%	7 0.1%	4 0.0%	97.2% 2.8%
5	0 0.0%	0 0.0%	2 0.0%	0 0.0%	957 9.6%	1 0.0%	6 0.1%	5 0.1%	3 0.0%	9 0.1%	97.4% 2.6%
6	1 0.0%	2 0.0%	0 0.0%	9 0.1%	0 0.0%	870 8.7%	4 0.0%	0 0.0%	4 0.0%	6 0.1%	97.1% 2.9%
7	4 0.0%	1 0.0%	3 0.0%	0 0.0%	3 0.0%	4 0.0%	936 9.4%	0 0.0%	1 0.0%	0 0.0%	98.3% 1.7%
8	1 0.0%	2 0.0%	5 0.1%	2 0.0%	2 0.0%	0 0.0%	0 0.0%	997 10.0%	2 0.0%	5 0.1%	98.1% 1.9%
9	1 0.0%	0 0.0%	5 0.1%	4 0.0%	1 0.0%	4 0.0%	2 0.0%	3 0.0%	947 9.5%	4 0.0%	97.5% 2.5%
10	1 0.0%	0 0.0%	1 0.0%	0 0.0%	14 0.1%	1 0.0%	0 0.0%	6 0.1%	4 0.0%	976 9.8%	97.3% 2.7%
	99.0% 1.0%	99.4% 0.6%	97.5% 2.5%	98.1% 1.9%	97.5% 2.5%	97.5% 2.5%	97.7% 2.3%	97.0% 3.0%	97.2% 2.8%	96.7% 3.3%	97.8% 2.2%

➔ The network architecture is shown below,



References:

1) https://www.mathworks.com/help/nnet/ref/trainautoencoder.html
2) http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/
3) http://ufldl.stanford.edu/tutorial/supervised/Pooling/
4)