# Introduction

An autoencoder is an unsupervised learning algorithm that attempts to reconstruct its input, i.e. set the target values to be equal to the inputs y(i) = x(i). The internal representation of an autoencoder encodes the input features into a feature vector in the hidden layer and learns the weights for these hidden layers by learning the approximation to the identity function hW,b(x) ≈ x.

The identity function places a constraint in the network. By limiting the number of neurons in the hidden layer interesting structure of the data can be learned from the autoencoders. Hence, in the case where the dimension of the hidden layer nh is (< n), the autoencoder obtain a lower dimensional representation of the input signals, that is it performs feature extraction algorithm that learns the important features of the data. This representation can be then feed into other algorithms, such as softmax algorithm for classification problem.

In this experiment, we aim to utilise the Autoencoder in order to use neural network as content addressable memories. The project uses hand-written digit images provided by the MNIST database for the dataset. Each image is a black and white (bilevel) images that is made up of 28x28 pixels (784 pixels). The total size of ~5000 training images and ~1000 testing images have been selected for the project.

## Algorithm

The training process of an autoencoder is based on cost function. The cost function measures the error between the input x and its reconstruction at the output. An autoencoder is composed of an encoder and a decoder. The encoder and decoder can have multiple layers. If the input to an autoencoder is a vector x, then the encoder maps the vector x to another vector z, as follows:



Figure 1. Encoder

where the superscript (1) indicates the first layer, h(1) is a transfer function for the encoder, W is the weight matrix and b is the bias vector. Then, the decoder tries to reconstruct the input by mapping the encoded representation z into an estimate of the original input vector, x, as follows:

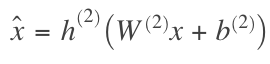


Figure 2. Decoder

where the superscript (2) represents the second layer. h(2) is the transfer function for the decoder.

The reconstructed input and the input values are then compared using a cost function, which uses Mean Squared Error (MSE). This MSE is then minimised during the backpropagation learning, so that the weights for the autoencoder that yield representations close the inputs can be obtained.

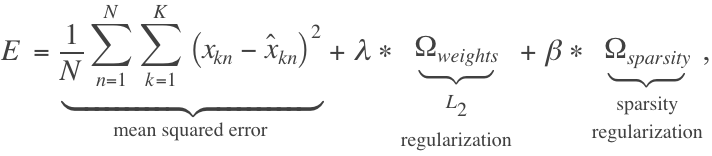


Figure 3. Mean Squared Errors

## Autoencoder Parameters

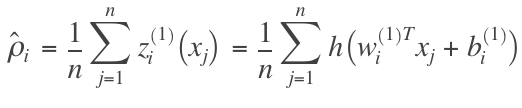
### Transfer Functions

Transfer functions are used to calculate a layer's output from its net input. The transfer functions are used by the neurons to determine neuron activation. For example, with sigmoid function, a neuron is active when the output is close to 1 and inactive when the output is close to 0.

|  |  |  |
| --- | --- | --- |
| **Log-Sigmoid Transfer Function** | **Saturating Linear Transfer Function** | **Linear Transfer Function** |
| Range: [min max] | Range: [0 1] | Range: [0 1] |
| logsig(n) = 1 / (1 + exp(-n)) | a = satlin(n) = 0, if n <= 0  n, if 0 <= n <= 1  1, if 1 <= n | a = purelin(n) = n |

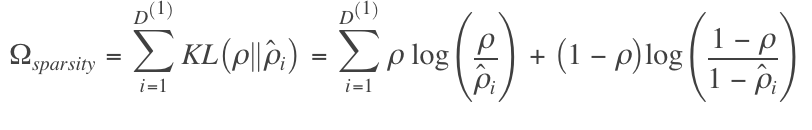
### Sparsity Constraints

When the size of hidden units is large, sparsity constraints can be imposed on the input data to explore interesting structures of the inputs. The activation transfer functions can determine whether a neuron is active or inactive. Sparsity constraint can be established to the AutoEncoder by applying a regularizer, a function of the average output activation value of a neuron. The average activation of a hidden neuron i is defined as:



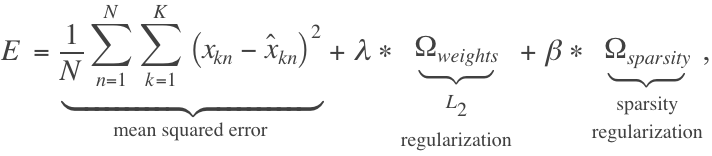
We would like to enforce the constraint: ρi = ρ, where ρ is the sparsity parameter. ρ is usually a small value close to 0. Lower ρ encourages the autoencoder to learn a representation, where each neuron in the hidden layer fires to a small number of training examples so that each neuron specializes by responding to features that is exist in a small subset of the training samples.

Sparsity regularizer is used to enforce a constraint on the sparsity of the output from the hidden layer. Sparsity can be established by adding a regularization term that takes a large value when the average activation value, ρi of a neuron i is not close to its desired value ρ. Here, we used Kullback-Leibler divergence to compute the regularization term.



In matlab, this Ωsparsity is enforced by the *SparsityProportion* parameter.

### Cost Function



In order to enforce sparsity constraints and avoid overwitting, we added sparsity regularization and regularization to the cost function. As seen in the formula above, β is the coefficient to control the sparsity regularization term. In matlab, the values of β can be set using the *SparsityRegularization* parameter.

# Methodologies

## Data Pre-processing

Before training and testing, the train and test data was first shuffled using the ‘autoencoderPreprocess.m’, which generates matlab objects for the data images and labels used during the training and testing. After the preprocessing, we will generate the following matlab objects:

* *dataTrain.mat*: 1x5005 cells, with 28x28 matrix in each cell. Every cell in the matrix corresponds to each pixel in the image and has a value within the [0 255] range. This range represents the intensity of the pixel, with 0 as total absence (black) and 255 as total presence (white). This image data will be used as the input to train the network model.
* *labelsTrain.mat*: The target data, 10x5005 matrix of one hot encoded values. Each column in the matrix represent the number of observations, which in this case is 5005. The row is a dummy variable representing a particular class, where all entries in a column are zero except for a single row. This single entry indicates the class label for the sample. There are 10 possible image labels, which is a number between 0-9. This data will be used for training a network model for classification task.
* *dataTest.mat*: 1x1000 cells, with 28x28 matrix on each cell. This data will be used to evaluate the model performance.
* *labelsTest.ma*t: 10x1000 matrix. This data will be used to evaluate the network model for classification task.

## Model Architectures and Learning Algorithms

### 1 Autoencoder

In part A, we implemented ANN model with 1 hidden layer. This layer is an autoencoder which made of 100 neurons. These neurons will then store the features of the input during the encoding process and tries to reconstruct the image back during the decoding process.

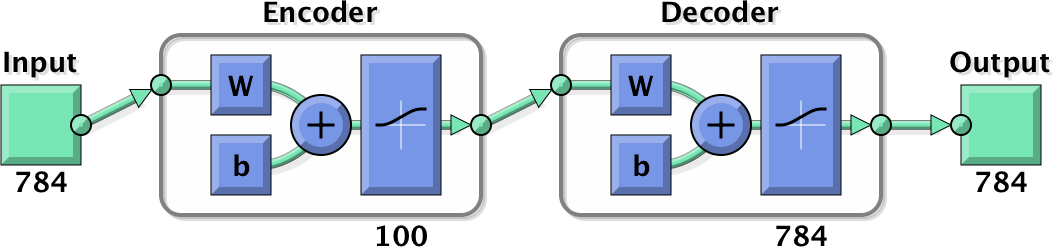


Figure 4. An Autoencoder with 100 hidden neurons

For the 1st experiment, we tested three hyperparameter variations for the AutoEncoder: 1) number of epochs, 2) sparsity regularization coefficient and sparsity proportion, and 3) transfer functions.

### 2 Autoencoders

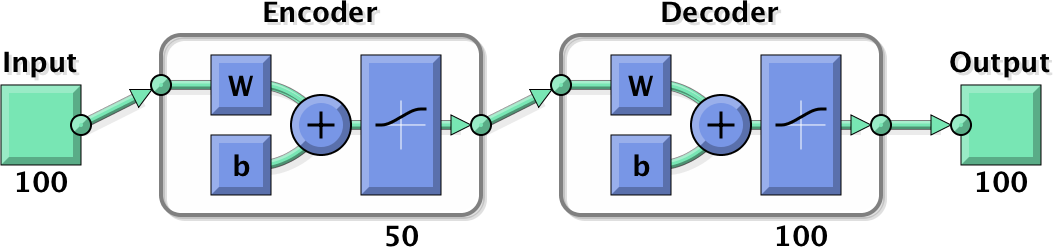


Figure 5. Second Autoencoder with 50 hidden neurons

In part B, we introduced a second autoencoder into the network model. The second autoencoder has 50 neurons and uses the outputs of the first Autoencoder in part 2.2.1 as the input features. As seen in the diagram above, because the first Autoencoder has 100 neurons, the second encoder will receive 100 input features, and reduce the dimensionality of the features into 50 features. Then, at the decoder it will try to reconstruct the input back into 100 features.

### 2 Autoencoders and 1 Softmax layer

In part C, we added a softmax layer as the last layer in the network model. In this architecture, there are 3 hidden layers: 2 hidden layer of autoencoders and the last layer for softmax.

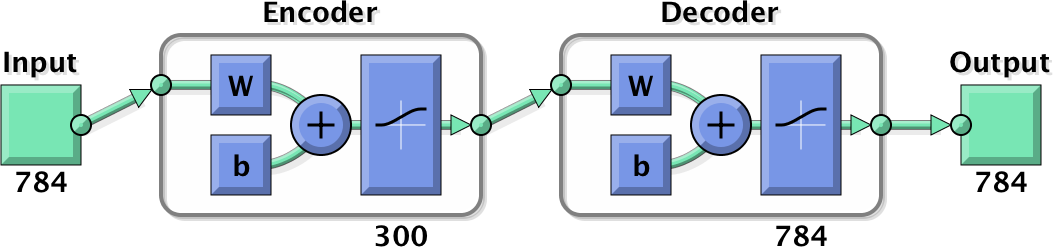


Figure 6. 1st Layer: AutoEncoder with 300 hidden neurons

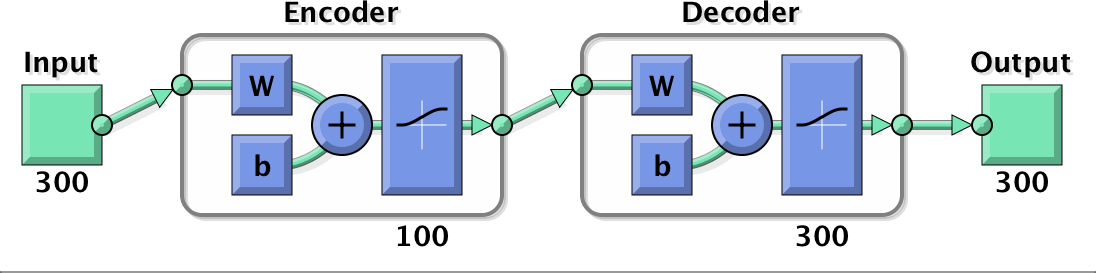


Figure 7. 2nd Layer: AutoEncoder with 100 hidden neurons

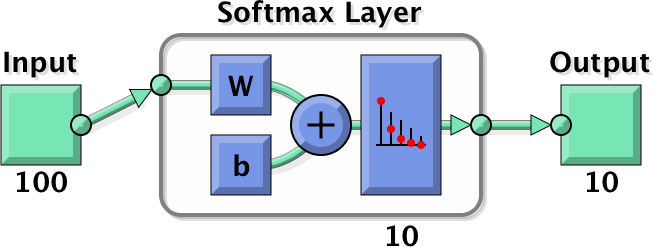


Figure 8. 3rd Layer: Softmax with 10 possible classes.

First, the image data will be encoded using the first autoencoder and the output is fed into the second autoencoder. The second autoencoder will further encode the encoded image and generate some outputs. Next, the softmax layer will receive the second AutoEncoder outputs. The softmax layer is used to solve multi-class labels classification problem, which in this experiment is given image data, classify which numeric label (0-9) is appropriate for that image data. Because there are 10 possible classes, the softmax layer will have 10 neurons at the output. Unlike autoencoders, a softmax layer is trained in a supervised using the labels from the training data.

For the third experiment, we tested different hyper parameters of the network model, such as: 1) number of neurons in the hidden layers, and 2) sparsity proportion and sparsity regularization coefficient of the autoencoders. In this part, we also did some experiment with deep network by stacking several autoencoders together with the softmax layer and compare its performance.

# Implementation

## Model with 1 Autoencoder Layer

The following parameters for training the Autoencoder are first set to the following values at initial.

* **EncoderTransferFunction and DecoderTransferFunction: logsig**
* **MaxEpochs: 1000**
* **SparsityProportion: 0.05**
* **SparsityRegularization: 1**

After each experiment, we observe the results and change the parameter of the next experiment to use the best parameters from previous observations. The best parameter in this case is the hyper parameters that generate the lowest MSE

1. ***Number of Epochs***

First, we performed the first experiment to find the best model among different number of epochs. We used the following numbers for the maximum epoch size:

*Different Epochs: [100 200 300 400 500 600 700 800 900 1000].*

1. ***Sparsity Proportions and Sparsity Regularization Coefficient***

**Best Epoch: 1000** (from previous experiment)

We used 1000 as the max epoch size to experiment with different sparsity proportion and regularization coefficient.

*Different Sparsity Proportions: [0.01 0.05 0.1 0.2 0.3].*

*Different Sparsity Regularization Coefficients: [1 4 7 10]*

1. ***Transfer Functions***

**Best Epoch: 1000**

**Best Sparsity Regularization Coefficient: 10**

**Best Sparsity Proportion: 0.1**

Based on the best parameters found in the previous experiment, we used them to test with different transfer functions.

*Different Encoding Transfer Functions: [logsig satlin]*

*Different Decoding Transfer Functions: [logsig satlin purelin]*

## Model with 2 Autoencoder Layers

The learning parameters used to create the Autoencoders in the second experiment used the best parameters found in experiment 2.2.1. The two autoencoders in this experiment will have the same hyper parameters except for the number of neurons. These controlled variables of the Autoencoders include:

* **EncoderTransferFunction and DecoderTransferFunction: logsig**
* **MaxEpochs: 1000**
* **SparsityProportion: 0.1**
* **SparsityRegularization: 10**

Here, the first autoencoder receives input features and encode it into an intermediate representation. This output is fed as an input to the encoder of the second autoencoder to generate an encoded representation of the encoded image. Next, this output is decoded using the second autoencoder’s decoder, and reconstructed back to the input features using the decoder of the first autoencoder. The reconstruction MSEs of using two autoencoders are then compared to the results obtained from using 1 autoencoder.

## Model with 2 Autoencoder Layers and 1 Softmax Layer

The following parameters for the Autoencoder are first set to the following values at initial.

* **EncoderTransferFunction and DecoderTransferFunction: logsig**
* **MaxEpochs: 200**
* **SparsityProportion: 0.1**
* **SparsityRegularization: 10**

1. ***Number of Neurons in the Hidden Layers***

*Hidden Neurons of First Autoencoder = [400, 300, 200, 100]*

*Hidden Neurons of the Second Autoencoder = [200, 100, 50]*

1. ***Sparsity Proprotion and Sparsity Regularization Coefficient***

**Best Hidden Neurons for the First Hidden Layer: 300**

**Best Hidden Neurons for the Second Hidden Layer: 100**

*sparsity\_regs = [1, 4, 7, 10]*

*sparsity\_props = [0.1, 0.2, 0.3]*

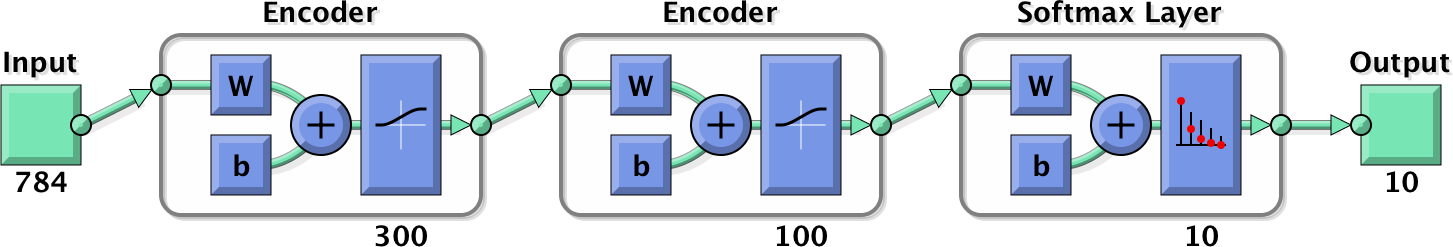


Figure 10. Network Model for the Third Experiment

In this experiment, the softmax layer receives the intermediate representation from the second autoencoder as the input features. This layer is trained using the training image labels and tested using the testing image data and testing image labels to evaluate the performance. The accuracy of each model is compared using confusion matrix.

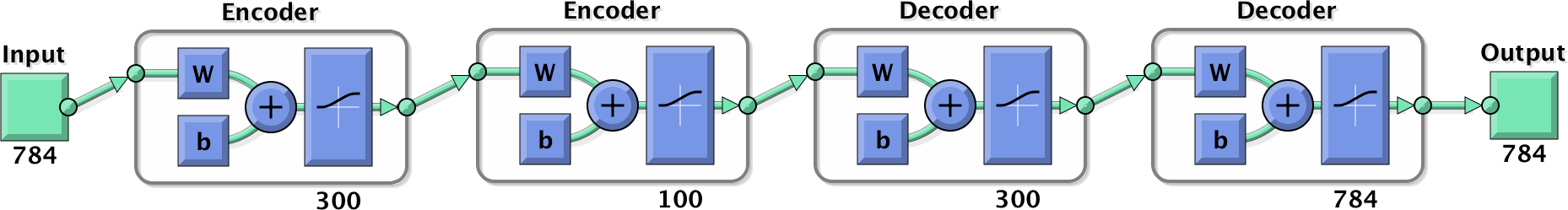


Figure 11. Autoencoders in the Third Experiment

Similar to part 3.2, in order to compare between the reconstruction errors of each models, the image needs to be decoded back to the original data by using decoders.

For the deep network experiment, the size of the hidden representation of the layer must match the input size of the next layer or network in the stack. The first input argument of the stacked network is the input argument of the first autoencoder. The output from the first encoder is the input of the second autoencoder in the stacked network. The output from the second encoder is the input argument to the final layer of in the stacked network, the softmax layer. We used the best parameters obtained in the previous experiment which were 300-100 hidden neurons in the first and second autoencoder.

# Results and Analysis

In this section, the results of different hyper parameters will be discussed. The comparison method will Mean Squared Errors (MSE) between inputs and its reconstructed output. This MSE obtained from experimenting with various hyper parameters will be displayed as a diagram and a table in each experiment. For reconstruction results, we will have a table. The first row (or the first 2 rows for multiple autoencoders) will show the weight visualizations while the last row displays the reconstructed image. Note that for the reconstruction results, we will only compare and discuss the best and worst hyper parameters observed during the experiment. The best and worst hyper parameters of an experiment is defined as those parameters that generate the lowest and highest MSE errors respectively.

Below is the original test image data used to evaluate the models.

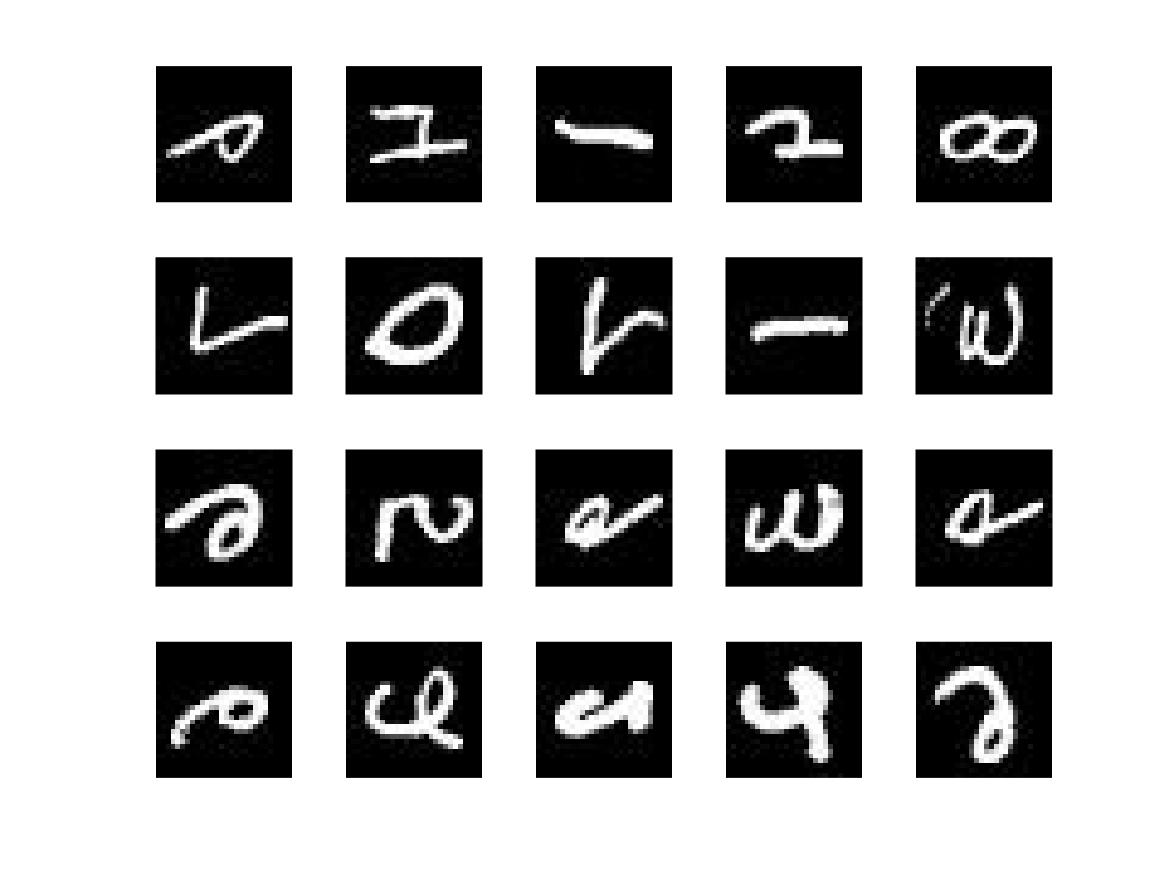


Figure 12. Original Test Image

## Model with 1 Autoencoder layer

### Number of Epochs

#### MSE

Figure 13. MSE Plot of Various Max Epoch Size

As seen in the diagram and the table above, we can see the inverse relation between MSE and epoch size. As we increase the number of epochs, the algorithm will continue to optimize the network by minimising the MSE, hence the MSE will decrease as the number of epochs in increased. The highest epoch size, 1000 epochs, give the best MSE of 244.36 while the lowest epochs, 100 gives the worst MSE of 812.437.

|  |  |
| --- | --- |
| **Best Epoch** | **Lowest MSE** |
| **1000** | **244.36** |

#### Reconstruction

|  |  |
| --- | --- |
| **Worst MSE Error (100 epochs) ≈ 812** | **Best MSE Error (1000 epochs) ≈ 244.36** |
| Part2-autoencoders/Results/part1/weights/weights_epoch100.jpg | Part2-autoencoders/Results/part1/weights/weights_epoch1000.jpg |
| Part2-autoencoders/Results/part1/reconstructed/autoenc1_reconstructed_epoch100.jpg | Part2-autoencoders/Results/part1/reconstructed/autoenc1_reconstructed_epoch1000.jpg |

As visualized above, model with 1000 epochs gives a reconstructed image that looks closer to the original image compared to the 100 epochs. The reason is because the model with 1000 epochs can learn more information about the input features because of the cost function minimization in backpropagation algorithm. Therefore, this model stores better weights/features that best represent the image features than other models with lower epochs.

### Sparsity Proportions and Sparsity Regularization Coefficients

#### MSE

Figure 15. MSE Plot of Various Sparsity Proportion and Sparsity Regularization Pairs

From the plot and the table, we can see that there seems to be no correlation between sparsity and MSE, nor sparsity proportion and sparsity regularization, this is shown by the random and imbalanced fluctuation of the MSE when the sparsity parameters are varied. However, it can be seen that even we vary these parameters, the range between the lowest MSE = 226.9904 and highest MSE = 260.4396 is very small, only ~33.4 . This means that for the network model with 50 hidden neurons, varying sparsity proportion into 0.01-0.3 and sparsity regularization into 1-10 does not really matter because these models generate MSE that fall within similar range. We suspect this is because the size of the autoencoder is small (50 neurons only). It is not a huge layer, so sparsity in this case is not significant.

Observing from the MSE results, the sparsity proportion of 0.1 and sparsity regularization of 10 generates the lowest MSE.

|  |  |  |
| --- | --- | --- |
| **Best Sparsity Proportion** | **Best Sparsity Regularization** | **Lowest MSE** |
| **0.1** | **10** | **226.9904** |

#### Reconstruction

|  |  |
| --- | --- |
| **Worst MSE Error (prop=0.2, reg=4) ≈ 260.4** | **Best MSE Error (prop=0.1, reg=10) ≈ 226.99** |
| Part2-autoencoders/Results/part1/weights/weights__ep1000_sr4_sp0.2.jpg | Part2-autoencoders/Results/part1/weights/weights__ep1000_sr10_sp0.1.jpg |
| Part2-autoencoders/Results/part1/reconstructed/reconstructed__ep1000_sr4_sp0.2.jpg | Part2-autoencoders/Results/part1/reconstructed/reconstructed__ep1000_sr10_sp0.1.jpg |

From the visual observations, it can be seen that both reconstructed images look similar because their MSE difference is relatively small.

### Transfer Functions

#### MSE

Based on the observations, both logsig-logsig and satlin-satlin produces better results compared to other transfer function combinations. The reason probably because, logsig and satlin always produce output that falls within [0 1] range, and have a smooth transition therefore it is able to reconstruct the image better than purelin. From the table, we can see how selecting transfer functions really affect the Autoencoder performance. This can be observed because the gap between the maximum MSE (~1756.9) and minimum MSE (230) = 1756.9-230 = 1526.9, is quite large.

|  |  |  |
| --- | --- | --- |
| **Enc = Logsig – Dec = Logsig** | **Enc = Logsig - Dec = Purelin** | **Enc = Logsig - Dec = Satlin** |
| **Part2-autoencoders/Results/part1/reconstructed/reconstructed_ep1000_sr10_sp0.1_enclogsig_declogsig.jpg** | **Part2-autoencoders/Results/part1/reconstructed/reconstructed_ep1000_sr10_sp0.1_enclogsig_decpurelin.jpg** | **Part2-autoencoders/Results/part1/reconstructed/reconstructed_ep1000_sr10_sp0.1_enclogsig_decsatlin.jpg** |
| **Enc = Satlin - Dec = Logsig** | **Enc = Satlin - Dec = Purelin** | **Enc = Satlin - Dec = Satlin** |
| **Part2-autoencoders/Results/part1/reconstructed/reconstructed_ep1000_sr10_sp0.1_encsatlin_declogsig.jpg** | **Part2-autoencoders/Results/part1/reconstructed/reconstructed_ep1000_sr10_sp0.1_encsatlin_decpurelin.jpg** | **Part2-autoencoders/Results/part1/reconstructed/reconstructed_ep1000_sr10_sp0.1_encsatlin_decsatlin.jpg** |

Comparing the reconstructed images and its mse from the table, it can be seen that generally, the Autoencoder learns the input features better if we use logsig transfer function as the encoder, espescially using logsig-logsig at both the encoder and decoder layer. From the table, using purelin in the decoder will generate worst results, because this transfer function is too simple to learn from the complex nature of the image data. Therefore, using non simple linear transfer function, such as logsig and satlin generalise better as a hidden representation function approximator than linear transfer functions. Aside from using logsigs, using satlin in both encoder and decoder also generates a good reconstructed results (mse error of ~253), this is because satlin has similar plot as sigmoid function.

|  |  |  |
| --- | --- | --- |
| **Best Encoder Transfer Function** | **Best Decoder Transfer Function** | **Lowest MSE** |
| **Logsig** | **Logsig** | **230** |

If we look closely to the reconstructed images produced by satlin, we can see that there are some noises in the reconstructed image (the little dots). This is because, satlin will maps output to 0 if x is less than 0 and to 1 if x is > 1, therefore the output signals will change very fast to either black or white, unlike the smooth transition with logsig signals. With the same reason, using satlin for the encoder and logsig for the decoder will lower the model performance because there will be information loss when decoding the result from logsig to satlin as some outputs is set to 0 and 1 by the satlin function. This case does not happen if we are using logsig as the encoder and satlin as the decoder. This is because the input signals from logsig will always be decimal values between [0 1] which is within satlin allowable range, hence the satlin function is able to produce a comparable result.

Although logsig-logsig transfer functions win in this experiment, using satlin-satlin can beneficial to sparse network. Sparsity arises when the features ≤ 0. The more these units exist in a layer, the sparser the resulting representation become. On the other hand, logsig function always generates non-zero values resulting in dense representations. Therefore, in the case of sparse representations, using satlin-satlin will be beneficial.

#### Reconstruction Results

|  |  |
| --- | --- |
| **Worst MSE Errors (Enc=Satlin, Dec=Purelin) ≈ 1756.9** | **Best MSE Error (Enc=Logsig, Dec=Logsig) ≈ 230** |
| Part2-autoencoders/Results/part1/weights/weights__ep1000_sr10_sp0.1_encsatlin_decpurelin.jpgPart2-autoencoders/Results/part1/reconstructed/reconstructed_ep1000_sr10_sp0.1_encsatlin_decpurelin.jpg | Part2-autoencoders/Results/part1/weights/weights__ep1000_sr10_sp0.1_enclogsig_declogsig.jpgPart2-autoencoders/Results/part1/reconstructed/reconstructed_ep1000_sr10_sp0.1_enclogsig_declogsig.jpg |

Based on the observation above, it can be seen that there is information loss in using satlin-purelin functions, shown clearly by the very low quality of the reconstructed image if compared to the high quality of the reconstructed image from using logsig-logsig transfer functions.

## Model with 2 Autoencoder Layers

For the second part, we use the best parameters from part 4.1 to train 2 autoencoder layers and to create a network with 2 autoencoders. The MSE that we received from this network is **1723.5982**, which is very high compared to the lowest MSE found in part 4.1 (~227). This is because, with the addition of the second Autoencoder, the encoded image is further encoded again, hence there will be more dimensionality reduction. This will allow the network model to learn more about the important features of the input images, however it will make it harder to reconstruct the input image back because of the double compressions. So there will be trade-offs between the saved features in the autoencoders and the reconstructed image quality.

### Sample Features and Reconstructed Image

|  |  |
| --- | --- |
| **1 Autoencoder (MSE ≈ 227)** | **2 Autoencoders (MSE ≈ 1723.6)** |
| Part2-autoencoders/Results/part1/weights/weights__ep1000_sr10_sp0.1_enclogsig_declogsig.jpg | Part2-autoencoders/Results/part2/weights/weights1__second_ep1000_sr10_sp0.1_enclogsig_declogsig_autoenc1.jpg |
| None | Part2-autoencoders/Results/part2/weights/weights2__second_ep1000_sr10_sp0.1_enclogsig_declogsig_autoenc2.jpg |
| Part2-autoencoders/Results/part1/reconstructed/reconstructed__ep1000_sr10_sp0.1.jpg | Part2-autoencoders/Results/part2/decoded/reconstructed_second__second_ep1000_sr10_sp0.1_enclogsig_declogsig.jpg |

The features of network model with 2 Autoencoders (the second column of the table) is observed. From the table, the first row represents the extracted features of the first hidden layer, while the second row represents the extracted features of the second hidden layer. For the first layer, the dimension of the features is 10x10 which denotes the number of the hidden neurons in the first hidden layer, 100 neurons. For the second features image, the dimension of the image is 100x50 pixels which represent the 100 neurons from the first hidden layer and the 50 neurons in the second hidden layer. The black and white color of the weights represent the magnitude of the weights connected to the input features.

Because when using two autoencoders the image is compressed twice and decoded twice, the image will have some loss hence the quality of the image is much worse than using single autoencoder. However, note that quality of the image (mse magnitude) only reflects the autoencoder’s ability to reconstruct images similar to the input image. We will see in part 4.3 how MSE / reconstructed image quality does not reflect the autoencoder’s ability in predicting classification labels.

## Model with 2 Autoencoder layers and 1 Softmax Layer

### Different Hidden Layers

#### MSE

Similar to the previous experiment, because we are using 2 second Autoencoders, the image will be encoded and decoded twice, hence the MSE is quite high ranging from ~1200-3100.

From the plot, it can be seen that there seems to be no correlation between the number of hidden neurons and the MSE, hence we must experiment with the number of hidden neurons manually in both autoencoders to get the best network model that best fits the input features. In this experiment we found that using 300 neurons in the first Autoencoder and 100 neurons in the second Autoencoder produce the lowest MSE.

|  |  |  |
| --- | --- | --- |
| **Best Hidden Neuron Size (hiddenSize1)** | **Best Hidden Neuron Size (hiddenSize2)** | **Lowest MSE** |
| **300** | **100** | **1200.83** |

#### Reconstructed Image and Confusion Matrix

|  |  |
| --- | --- |
| **Worst MSE Error (100-200)**  **MSE ≈ 3116.21, Accuracy = 90.4%** | **Best MSE Error (300-100).**  **MSE ≈ 1200.83, Accuracy = 86.8%** |
| Part2-autoencoders/Results/part3/weights/weights1__hidden_ep200_sr10_sp0.1_enclogsig_declogsig_h1100_autoenc1.jpg | Part2-autoencoders/Results/part3/weights/weights1__hidden_ep200_sr10_sp0.1_enclogsig_declogsig_h1300_autoenc1.jpg |
| Part2-autoencoders/Results/part3/weights/weights2__hidden_ep200_sr10_sp0.1_enclogsig_declogsig_h1100h2200_autoenc2.jpg | Part2-autoencoders/Results/part3/weights/weights2__hidden_ep200_sr10_sp0.1_enclogsig_declogsig_h1300h2100_autoenc2.jpg |
| Part2-autoencoders/Results/part3/decoded/reconstructed_hidden_ep200_sr10_sp0.1_enclogsig_declogsig_h1100h2200_autoenc2.mat.jpg  Part2-autoencoders/Results/part3/confusion/confusion__hidden_ep200_sr10_sp0.1_enclogsig_declogsig_h1100h2200_autoenc2.jpg | Part2-autoencoders/Results/part3/decoded/reconstructed_hidden_ep200_sr10_sp0.1_enclogsig_declogsig_h1300h2100_autoenc2.mat.jpg  ../part3/confusion/confusion__hidden_ep200_sr10_sp0.1_enclogsig_declogsig_h1300h2100_autoenc2.jpg |

From this comparison, it can be seen that lower MSE generates reconstructed image that is blurry and quite different from the original image. However, surprisingly, the softmax layer connected to these autoencoders are able to predict the class label with a higher accuracy (90.4%) compared to the autoencoders that has the best MSE (86.8%).

Because of this finding, we tried to plot the relationship between MSE and accuracy of the network with various number of hidden neurons.

From the plot, it is shown that there seems to be no correlation between MSE and accuracy. Therefore, the MSE of the Autoencoders cannot be used to reflect the model accuracy in predicting multiclassification labels. So again, we must experiment with the number of neurons manually in order to find the best network with the best predicting capabilities.

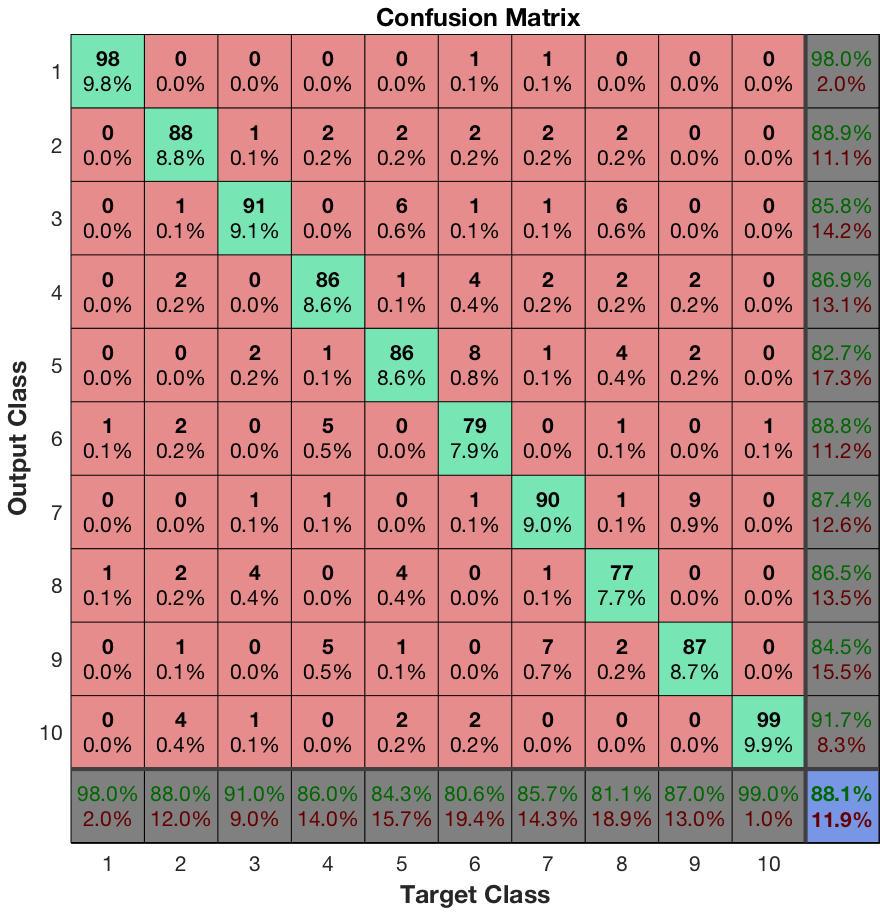
### Different Sparsity Regularization and Proportion

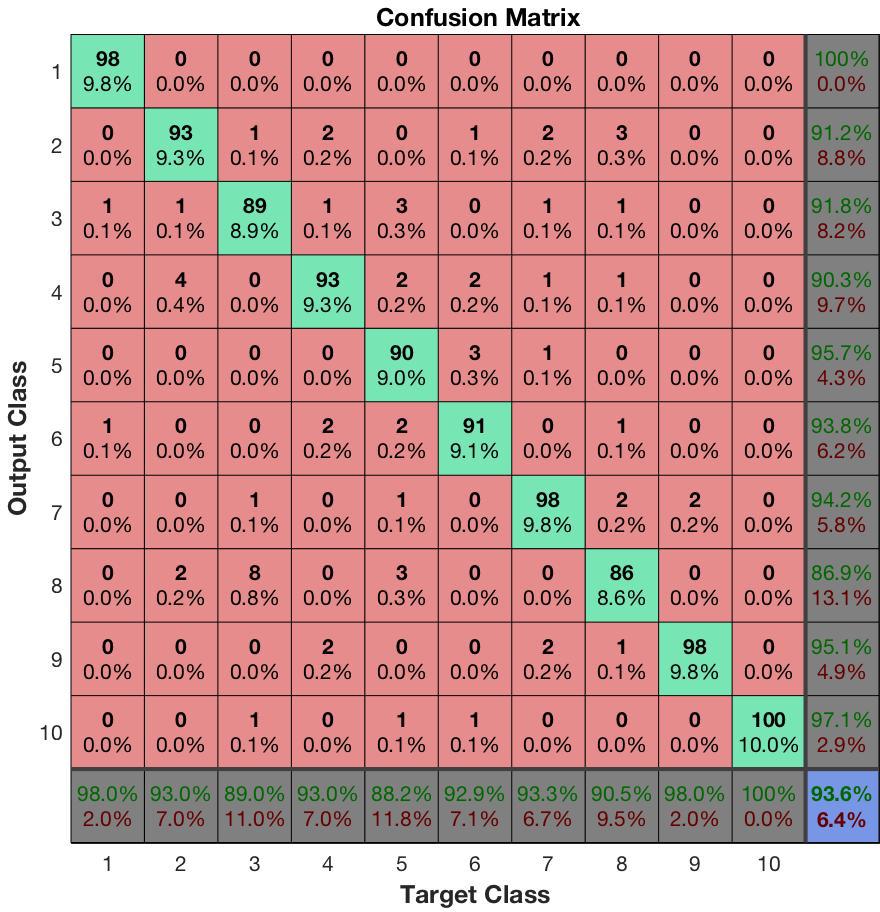
#### MSE

From these observations, it seems like the MSE errors generally increase when we increased the sparsity regularization and lowering the sparsity proportion. I suspect this is because our input features is not huge enough, so making the network sparse does not yield any benefits, so increasing the sparsity of the network by lowering the sparsity proportion, will instead reduce the number of important features learned by the Autoencoders.

#### Reconstructed Image and Confusion Matrix

|  |  |
| --- | --- |
| **Worst MSE Error (prop=0.1, reg=7)**  **MSE ≈ 1925.14, Accuracy ≈ 86.50** | **Best MSE Error (prop=0.1, reg=1)**  **MSE ≈ 1006.97, Accuracy ≈ 1006.97** |
| Part2-autoencoders/Results/part3/weights/weights1__sparsity_ep200_h1300h2100sr7_sp0.1_enclogsig_declogsig_autoenc1.jpg | Part2-autoencoders/Results/part3/weights/weights1__sparsity_ep200_h1300h2100sr1_sp0.3_enclogsig_declogsig_autoenc1.jpg |
| Part2-autoencoders/Results/part3/weights/weights2__sparsity_ep200_h1300h2100sr7_sp0.1_enclogsig_declogsig_autoenc2.jpg | Part2-autoencoders/Results/part3/weights/weights2__sparsity_ep200_h1300h2100sr1_sp0.3_enclogsig_declogsig_autoenc2.jpg |
| Part2-autoencoders/Results/part3/decoded/reconstructed__sparsity_ep200_h1300h2100sr7_sp0.1_enclogsig_declogsig.mat.jpg | Part2-autoencoders/Results/part3/decoded/reconstructed__sparsity_ep200_h1300h2100sr1_sp0.3_enclogsig_declogsig.mat.jpg |

1-0.3

10-0.3

# Discussions and Challenges

* Again, we must experiment with the hyper parameters of an autoencoder to get the best results,
* Time constraint.
* Increase epochs to gain higher accuracy