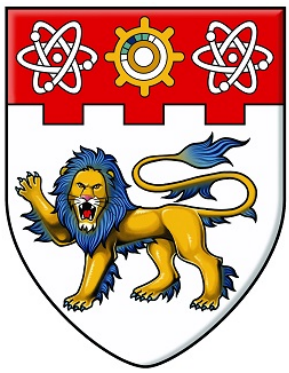

NANYANG TECHNOLOGICAL UNIVERSITY

CZ4042

Neural Network

Project 2 Report

Student:
Tang Cheng
Wei Yumou

Academic Year: 2016-2017 Sem 1

1 Introduction

Convolutional neural network (CNN) is a feed-forward artificial neural network (ANN) inspired by the biological retina and visual cortex structures primarily used for image recognition. The filters and convolution layers simulates the lateral excitation of the photosensitive cells. The pooling layers and fully connected layers simulates the hierarchical connections between different visual cortices.

Autoencoder is an ANN used for unsupervised learning. It explores the intrinsic structures of input data by limiting the hidden layer size and the sparsity constraints. Several autoencoders can be stacked together with fine tuning to form a multi-layer perceptron (MLP) for classification or regression tasks.

Part 2 and 3 will be showing the experimental details on the CNN and autoencoders performed on the MNIST database for recognizing hand-written digits. In both parts, there are 5,005 training samples and 1,000 testing samples.

2 Convolutional neural network

2.1 Algorithm

The typical components of a convolutional neural network consists of the input layer, convolution layer, pooling layer, fully connected layer, softmax layer, and classification layer.

The patterns (edges and borders) of an image can be extracted in the convolution layer, and then the feature dimension is reduced by the pooling layer. Fully connected layer and the rest maps the extracted features to a higher feature dimension for classification purpose.

Back-propagation is used to learn the weights between different layers, and up-sampling is used between the pooling layer and convolution layer when back-propagating the error signal.

2.2 One hidden layer CNN

In this section, we will construct a CNN with one convolution layer and one mean pooling layer.

Following parameters need to be searched (with their default values given):

- `batch size=128`
- `learning rate=0.01`
- `learning rate decay=none`
- `momentum=0.9`

2.2.1 Batch size

Mini-batch size specifies how many samples are learned in each iteration. Mini-batch gradient descent reduces the demand for large memory size compared to batch gradient descent, and also reduces the variation of each update compared to stochastic gradient descent.

The results obtained from 10 runs of random and interleaved division are plotted in Figure 1.

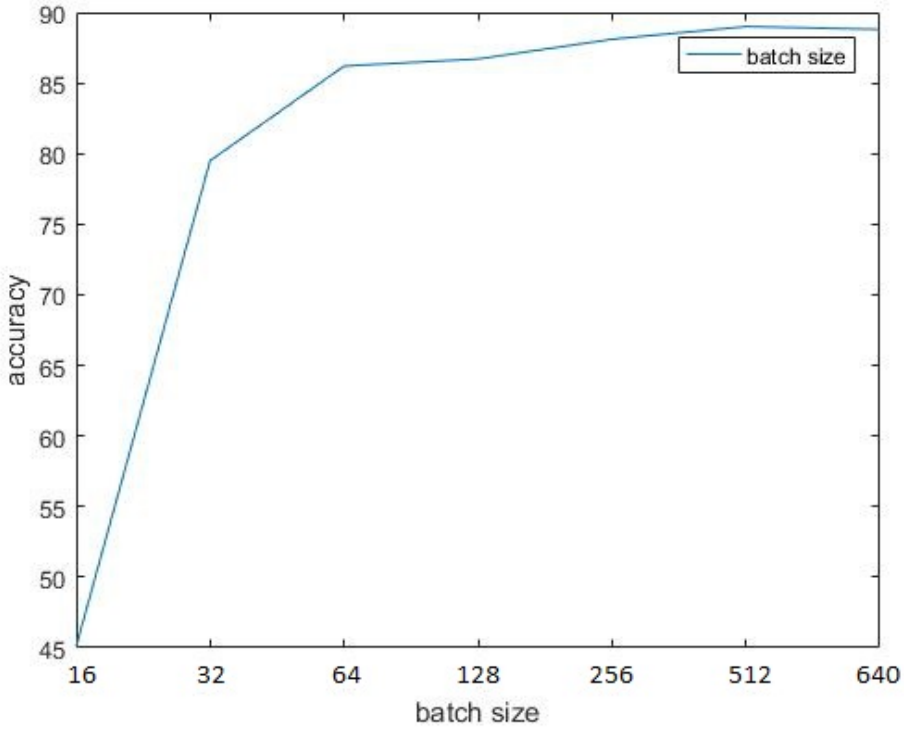

Figure 1: Mini-batch size

It can be observed accuracy is the highest (89%) when the mini-batch size is 512, thus we choose 512 as the optimal mini-batch size.

2.2.2 Learning rate and decay

The initial learning rate of various scales and corresponding results are shown in Figure 2.

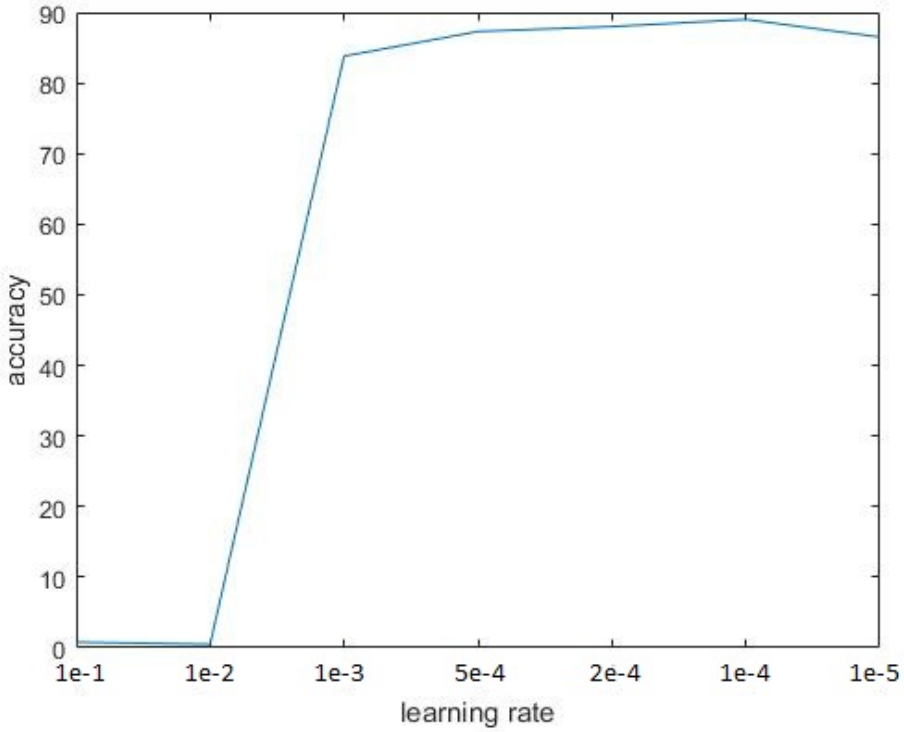

Figure 2: Different sizes of hidden layer

It can be observed that the accuracy is the highest when the initial learning rate is around `1e-4`, so we will use this value to find out the learning rate decay term.

In Matlab, the only available learning rate drop scheme is specified by `piecewise`, under which the learning rate will be multiplied by a factor after a certain amount of epochs.

The multiplication factor is specified by `LearnRateDropFactor`, which will be searched for, and the number of epochs for each rate update is specified by `LearnRateDropPeriod`, which will be set to 10 as in default.

Figure 3 shows the results of different dropping factors.

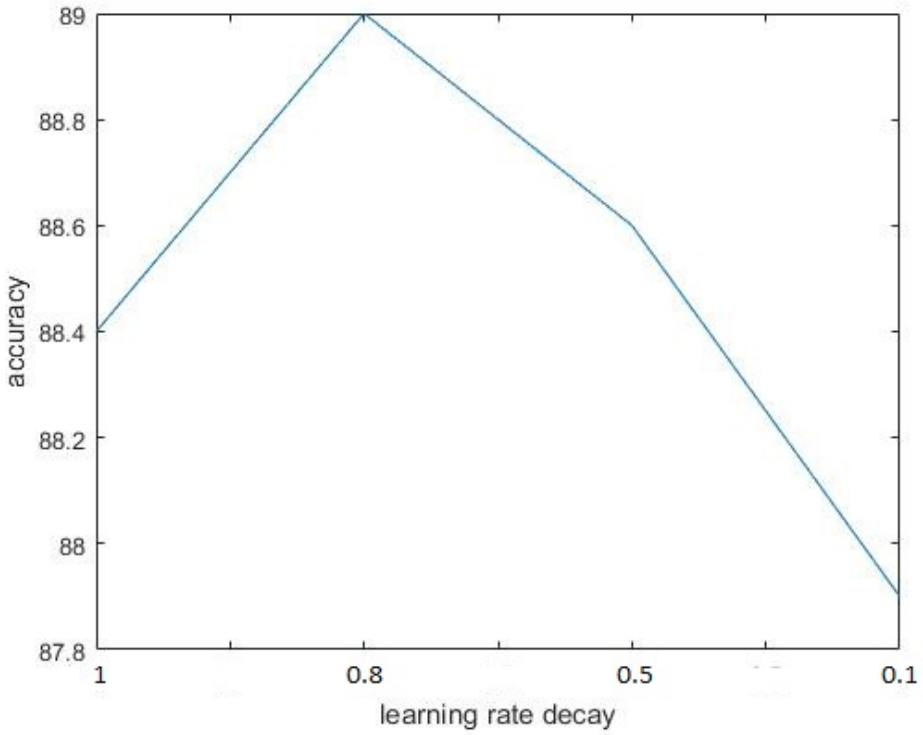

Figure 3: learning rate decay

According to the graph, we choose the decay rate to be 0.8.

2.2.3 Momentum

`Momentum` specifies the amount of contribution the previous iteration makes to the current iteration. It is helpful in accelerating convergence when the error function has the shape of a shallow ravine.

Figure 4 shows the results of different momentums.

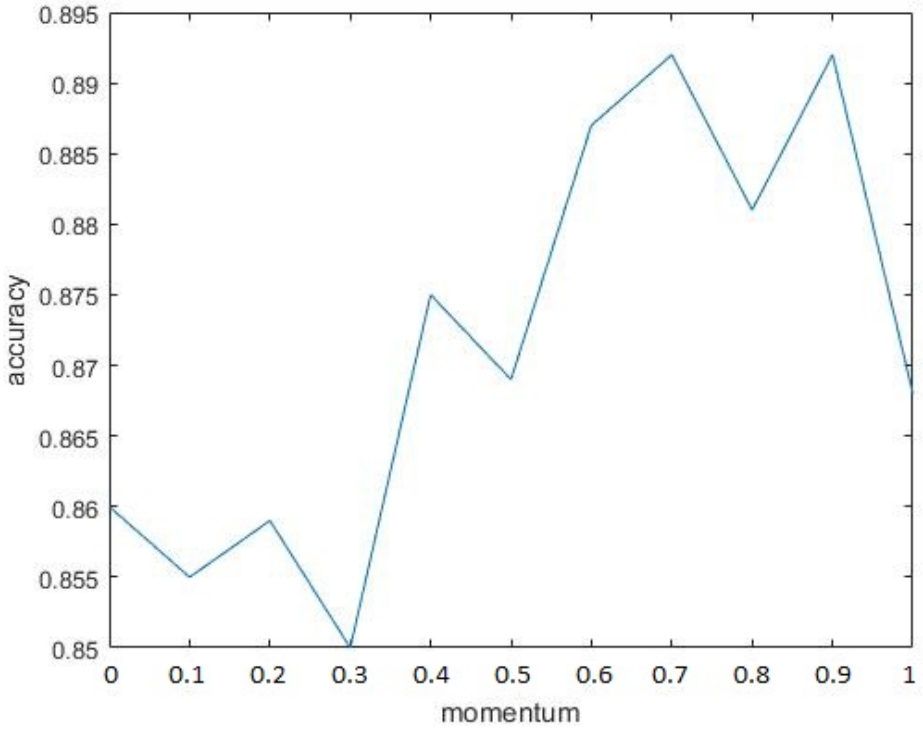

Figure 4: Momentum

The best result is achieved when the momentum is set to 0.7 and 0.9. We make 0.9 our final choice because it is also the default value.

2.2.4 Summary on one hidden layer CNN

According to the experiments conducted in Section 2.2.1 to Section 2.2.3, our final choice of parameter is listed below.

- `batch size=512`
- `learning rate=0.0001`
- `learning rate decay=0.8`
- `momentum=0.9`

The accuracy obtained on these parameters should be around 89.2%.(Run `P2_part1a.m`)

2.3 Deep CNN

In this section, we expand the model into a CNN with two sets of convolution and pooling layers.

2.3.1 Hidden Layer Size

Different sizes for both hidden layers are to be explored in this section. The results are shown in Figure 5.

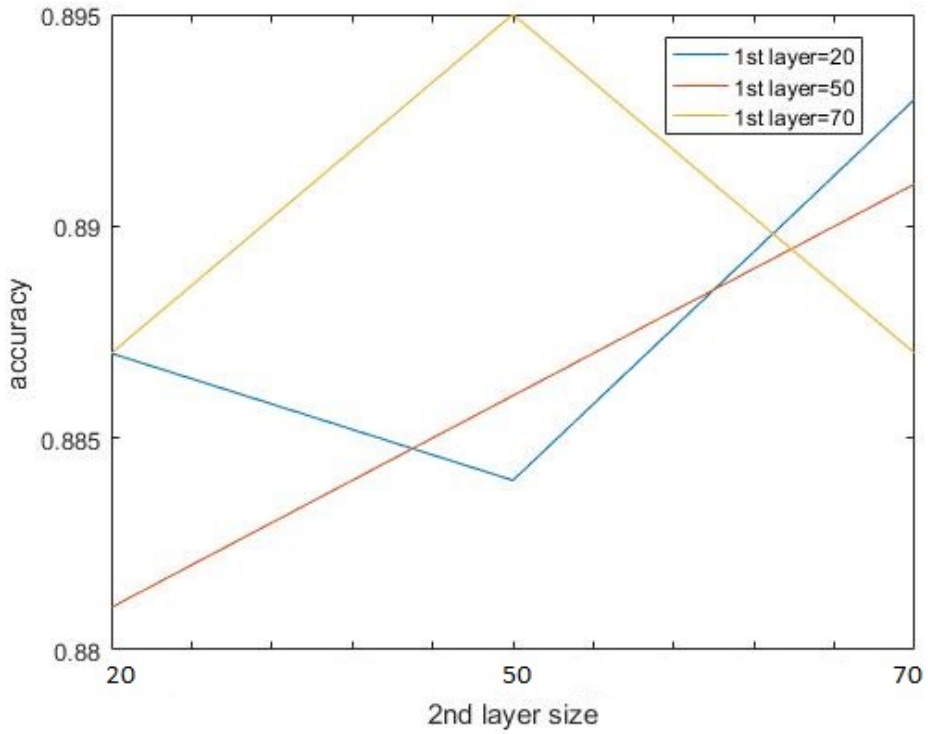

Figure 5: Different sizes of hidden layer

The architecture with 70 filters in the first layer and 50 filters in the second layer outperforms the rest, and the accuracy is around 89.5%. Note that the first layer size is larger than that of the second layer, which is in accordance with the fact that lower visual cortices has more neuron connections than the higher ones. (Run `P2_part1b.m`)

2.4 Rectified Linear Unit (ReLU) layer

Rectified Linear Unit (ReLU) layer uses activation function `f(x)=Max(0,x)` to perform a threshold function on the input data. It adds non-linearity to the whole network without significantly increasing the computational cost.

By adding one ReLU layer between the convolution layer and average pooling layer, the accuracy can be improved to 94.2%. (Run `P2_part1c.m`)

2.5 Summary on CNN

The parameters for one hidden layer CNN is listed below.

- `batch size=512`
- `learning rate=0.0001`
- `learning rate decay=0.8`
- `momentum=0.9`

The final accuracy should be around 89.2%.
The layer size for deep CNN is listed below.

- `layer one=70 filters`
- `layer two=50 filters`

The final accuracy should be around 89.5%.

With one more ReLU layer in the one hidden layer architecture, the accuracy can be boosted to 94.2%.

3 Autoencoder

3.1 Algorithm

An autoencoder is an ANN that tries to learn a compressed representation of the input signal in an unsupervised fashion, by having a hidden layer of smaller dimension than that of the input. It usually consists of an encoder and a decoder. As the names suggest, the encoder transforms (encodes) the input signal into a more concise representation based on which the decoder then reconstructs (decodes) the original input. In this experiment, we use Mean Squared Error (MSE) calculated from the reconstructed results and the original inputs to measure the quality of an autoencoder.

3.2 Single autoencoder

In this section, we construct a single autoencoder with 100 neurons in the hidden layer and inspect the MSEs for different configurations of the parameters shown below (with the default values given):

- `Maximum epochs=1000`
- `Sparsity proportion=0.05`
- `Sparsity regularization=1`

Once an autoencoder with the smallest MSE is determined, we will also observe the differences in MSEs and the reconstructed results when we apply different transfer functions.

3.2.1 Maximum epochs

`Maximum epochs` parameter specifies the maximum number of training epochs an autoencoder can run before it stops. This is to ensure the training will not last indefinitely in case that the training algorithm does not converge within a reasonable period of time. Figure 6 shows the resulting MSEs for maximum epochs from 100 to 1,000 in an increment of 100.

It can be observed that the MSE monotonically decreases with the maximum epochs, possibly because more training epochs allow the autoencoder to develop a deeper understanding of the input patterns. Therefore, we stick to the default 1,000 for maximum epochs.

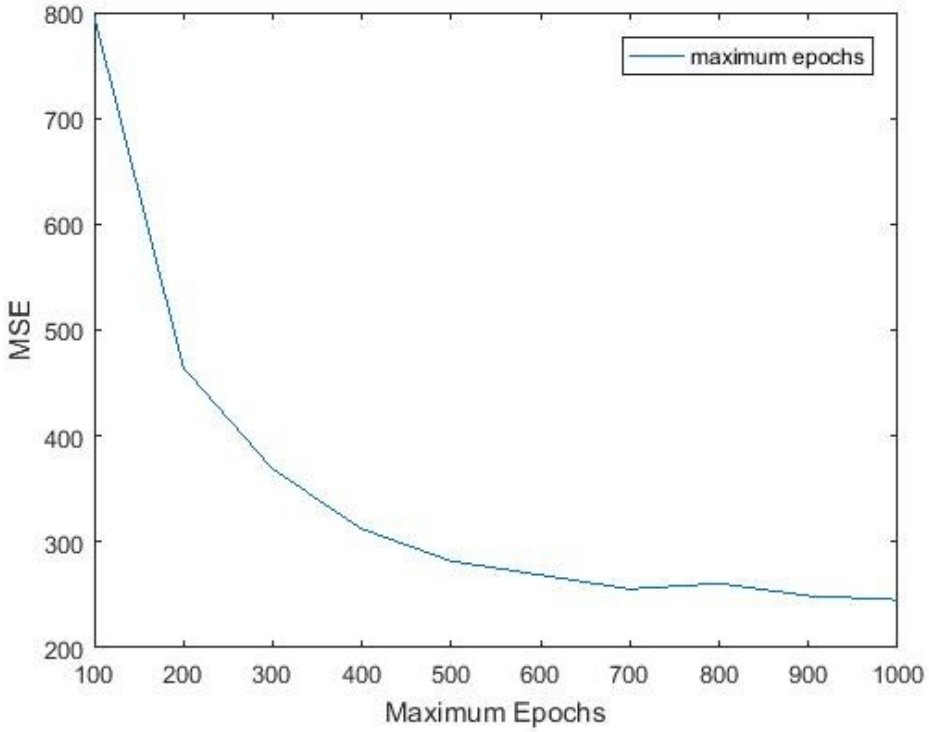

Figure 6: Maximum Epochs

3.2.2 Sparsity proportion

`Sparsity proportion` specifies the proportion of training example to which a neuron fires (produces a high output activation value). The lower the sparsity proportion is, the greater the degree to which a neuron is specialising in a small number of input examples is. Figure 7 below shows the MSEs of different sparsity proportions ranging from 0 to 0.19 in an increment of 0.01. Note the maximum epoch is set to 100 in this case to allow fast convergence.

It can be concluded that the optimal sparsity proportion is 0.11 at which the MSE is the smallest.

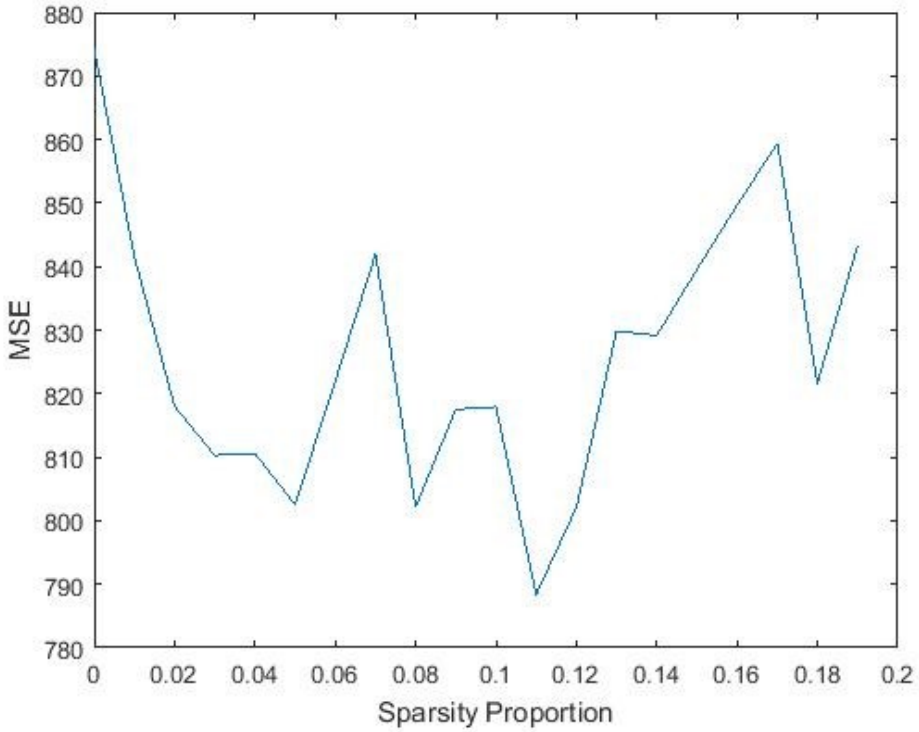

Figure 7: Sparsity Proportion

3.2.3 Sparsity regularization

Sparsity regularisation determines the impact of the sparsity regulariser. It encourages sparsity when set to a large value. Below shows the MSEs for different sparsity regularisation ranging from 1 to 30 in an increment of 1.

The sparsity regularisation that leads to the smallest MSE is 22. To summarise, we have obtained the following optimal values for the parameters of interest:

- `Maximum epochs=1000`
- `Sparsity proportion=0.15`
- `Sparsity regularization=22`

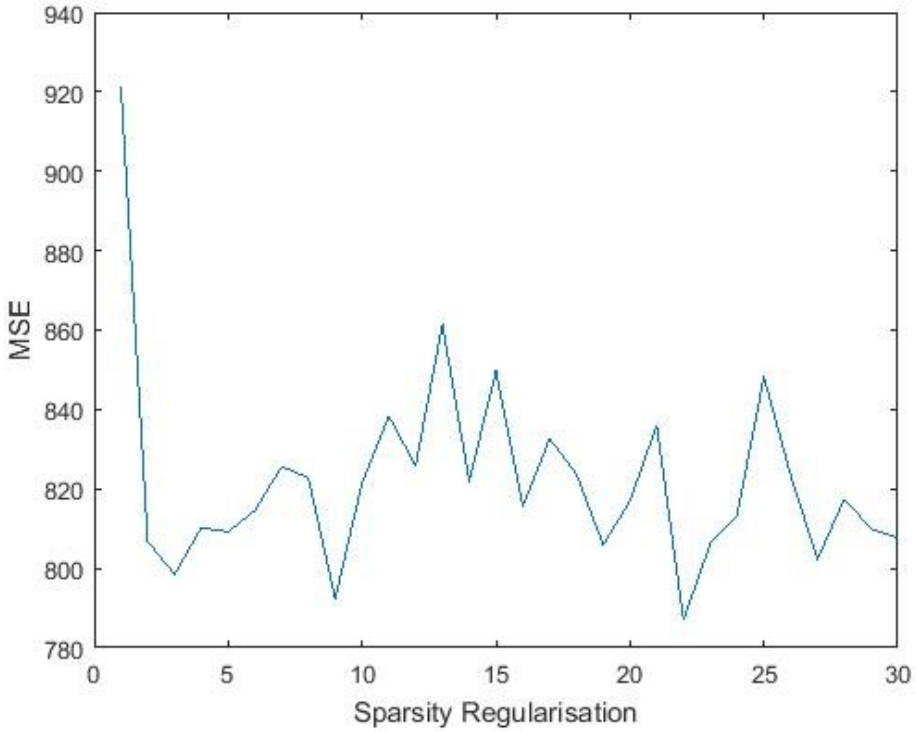

Figure 8: Sparsity Proportion

3.2.4 Transfer functions

Having obtained an optimal set of the parameter configuration, in this section we investigate the impact of the encoders and the decoders transfer function on the MSEs. Table 1 summarizes the MSEs for all possible combinations of the transfer functions.

Encoder	Decoder	MSE
logsig	logsig	224.66
logsig	satlin	697.23
logsig	purelin	440.85
satlin	logsig	1.12e+03
satlin	satlin	241.01
satlin	purelin	1.37e+03

Table 1: MSE for All Transfer Function Combinations

It can be observed that the transfer function pairs logsig-logsig and satlin-satlin are candidate optimal transfer functions, since they have the least MSEs amongst others. However, an inspection on their visual representations reveals some peculiarities. Below shows the original test images (on the left) and the reconstructed images (on the right) for the pair logsig-logsig,

Figure 9: logsig-logsig Original & Reconstructed

and the original test images (on the left) and the reconstructed images (on the right) for the pair satlin-satlin:

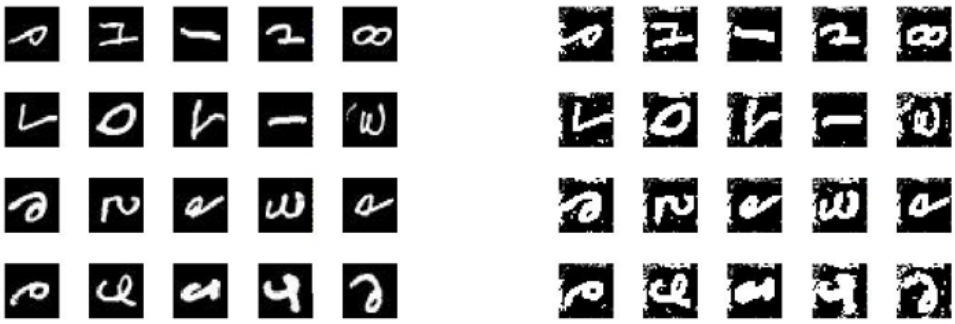

Figure 10: satlin-satlin Original & Reconstructed

For most humans, the reconstructed images of the `logsig-logsig` pair are so illegible that we can hardly tell they are replications of the original inputs. But still, interestingly, this pair has a MSE less than that of the `satlin-satlin` pair whose reconstructed images are almost identical to the original images, at least as far as humans can read. We repeated this experiment on these two pairs of transfer functions for several times using different training and testing sets but the results remained unchanged: the `logsig-logsig` pair always outperformed the `satlin-satlin` pair in terms of MSEs, despite the visually unconvincing results they produced. We shall investigate the reasons behind for the discrepancy between MSE and human visual perception. First of all, the training and test images are given as intensity images that are represented by a single matrix containing as elements intensities for each pixel in the images. The colouring of an *intensity image* is determined by intensity range and colour map. The default colour map for intensity images is greyscale; however, the default intensity range depends on the data type of the representing matrix. If the representing matrix is of 8-bit unsigned integer (`uint8`), which is the case for the training and test images, the intensity range is from 0 to 255 where 0 represents black and 255 represents while. If the representing matrix is of double precision (`double`), the intensity range is by default from 0.0 to 1.0 where 0.0 represents black and 1.0 represents while.

For the same colour map (for example, greyscale), different intensity ranges result in different colouring of the images, which explains the strange-looking reconstructed images by the `logsig-logsig` pair. The original training and test images are represented in data matrixes of uint8 type. Af-

ter the `logsig-logsig` transfer functions are applied, the resulting reconstructed images are actually represented in data matrixes of `double` type. The change in data type also modifies the intensity range and as a result, the reconstructed images shown previously are not the authentic ones. To get the correct reconstructed images, we applied a `uint8` conversion before we used `imshow` to display images. The results are shown below.

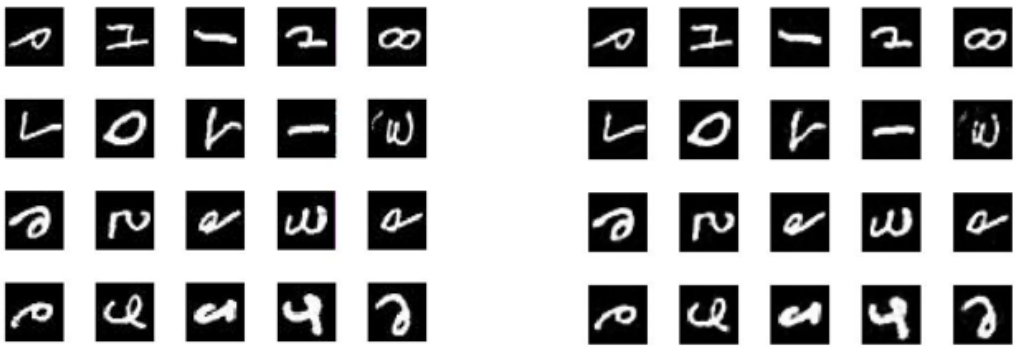

Figure 11: Correct `logsig-logsig` Original & Reconstructed

Fortunately, our human visual perception triumphed. It was the computer that played the trick with us. From the reconstructed images above we can observe that the `logsig-logsig` pair produces nearly perfect results, at least visually. Although the `satlin-satlin` pair produced a slightly worse result, we nevertheless decided to use both pairs for subsequent explorations because they do not differ substantially from each other.

3.2.5 Summary on single autoencoder

We have obtained the optimal values of the following parameters for a single autoencoder.

Parameter	Optimal Value
Maximum Epochs	1,000
Sparsity Proportion	0.11
Sparsity Regularization	22
Encoder transfer function	logsig/satlin
Decider transfer function	logsig/satlin

Table 2: Optimal Parameters for Single Autoencoder

The plots of convergence with the above optimal parameters are shown below (`logsig-logsig` pair on the left and `satlin-satlin` pair on the right) (Run `part2_a_final.m`).

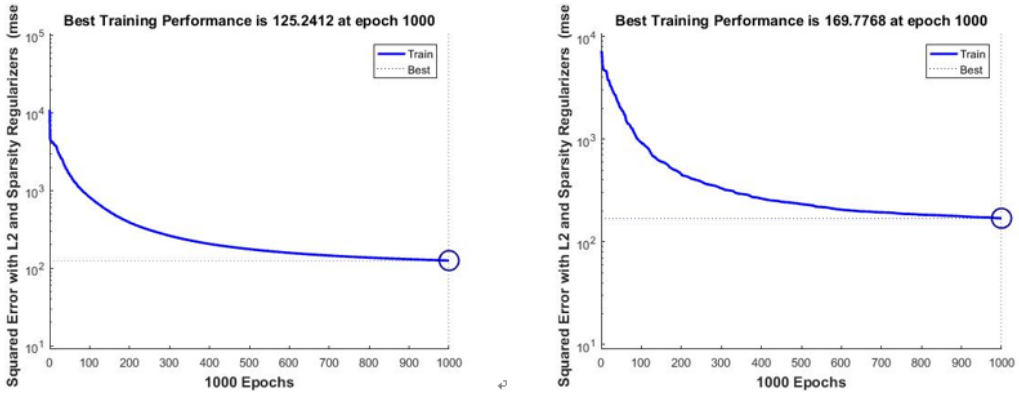

Figure 12: Plots of Convergence

3.3 Stacked autoencoders

Using the optimal parameter configuration obtained in the last section, we train an additional autoencoder that has a parameter configuration identical to the first ones but contains in its hidden layer only half the number of neurons in the first one. The second autoencoder is trained using the features generated from the first one and the decrease in dimension allows the second autoencoder to learn an even smaller representation of the input signal. We will compute the MSE for this stacked autoencoder which is then compared with that from the single autoencoder case. We use both logsig-logsig and satlin-satlin pairs in our experiments.

3.3.1 `logsig-logsig` pair

In this case, the encoder and decoder both use the logsig transfer function for both autoencoders. Below show the features learned by the first (on the left) and the second autoencoder (on the right), respectively.

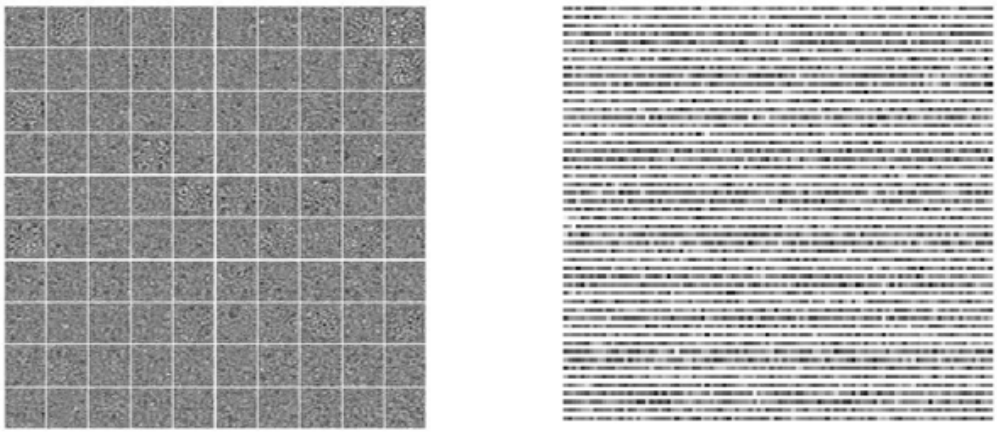

Figure 13: logsig-logsig Features Learned

The MSE calculated from the reconstructed images and the original test images turns out to be 1.6196e+03, which is demonstrated by the visually dramatic differences between the two sets of images as shown below.

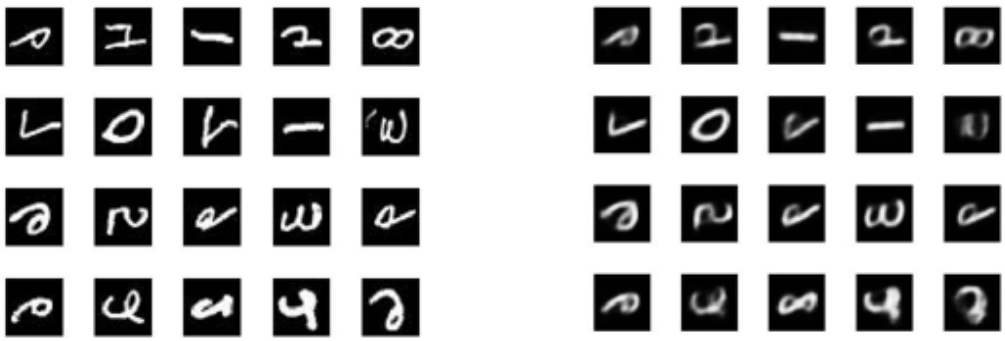

Figure 14: logsig-logsig Original & Reconstructed

3.3.2 `satlin-satlin` pair

In this case, the encoder and decoder both use the satlin transfer function for both autoencoders. Below show the features learned by the first (on the left) and the second autoencoder (on the right), respectively.

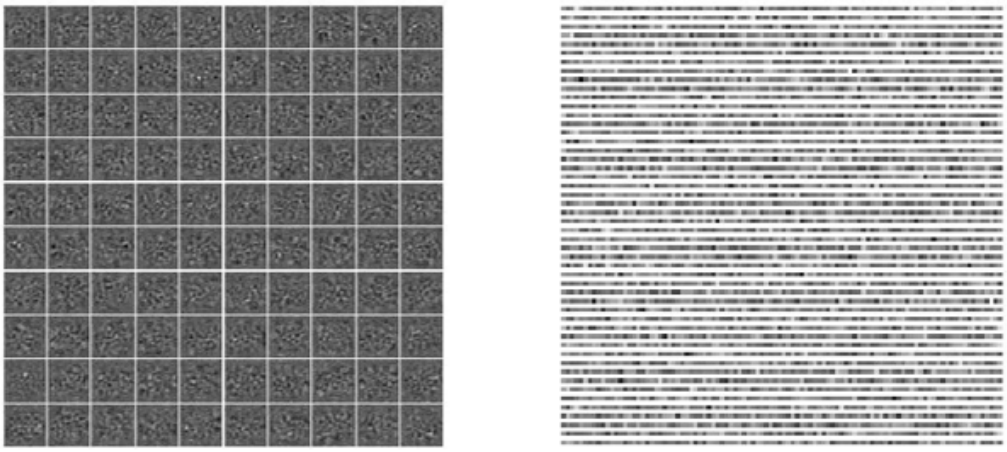

Figure 15: satlin-satlin Features Learned

The new stacked autoencoder using satlin transfer function has an apparently higher MSE which amounts to 805.0558. The increase in MSE is almost self-explanatory if we compare the legibility of the reconstructed images (on the right) below with the ones in the single autoencoder: the former is apparently less legible than the latter.

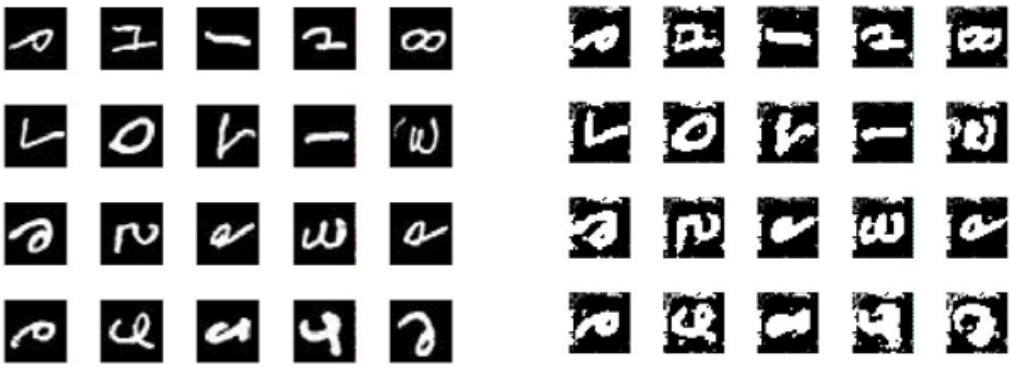

Figure 16: satlin-satlin Original & Reconstructed

(Run `part2_b.m`)

3.4 Deep ANN

In this section we build a deep ANN on top of the stacked autoencoder by appending an additional softmax layer to the second autoencoder. In this deep network, the two autoencoders are responsible for extracting from the input images feature representations which are then passed to the softmax classifier to learn in a supervised fashion.

Our goal is to evaluate the impact of the number of neurons in both hidden layers on the classification accuracy (measured in percentage) and the reconstruction MSE. To begin with, we need to find an optimal pair of sparsity regularization and sparsity proportion to use in our deep ANN.

3.4.1 Sparsity regularization

We look for an optimal value of sparsity regularisation that (ideally) gives both the largest classification accuracy and the smallest MSE. Below show the results.

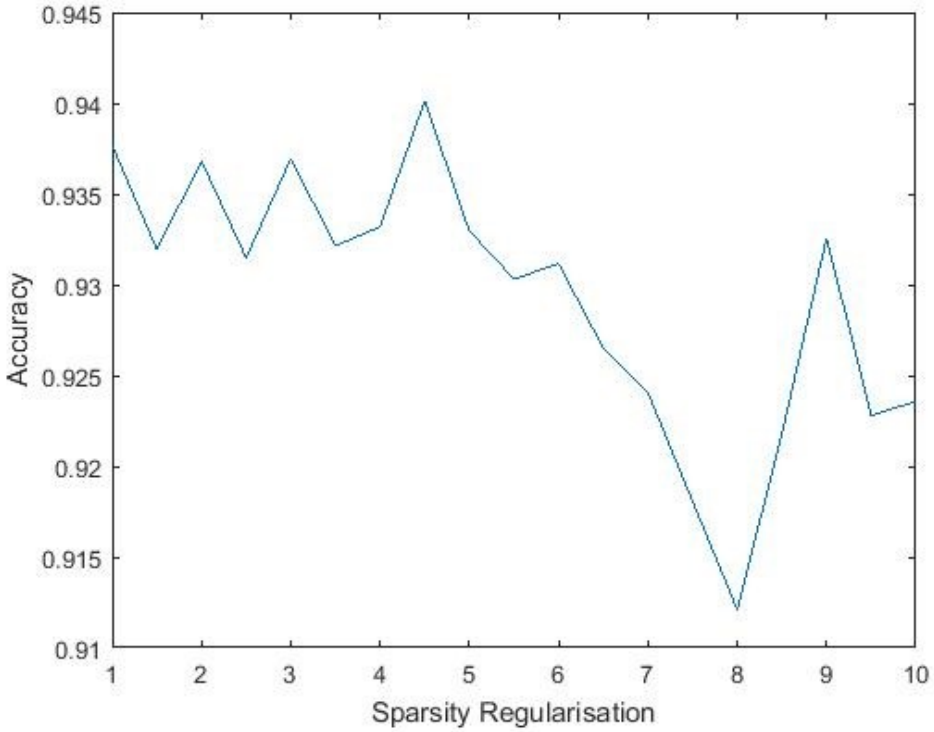

Figure 17: Sparsity Regularisation Against Accuracy

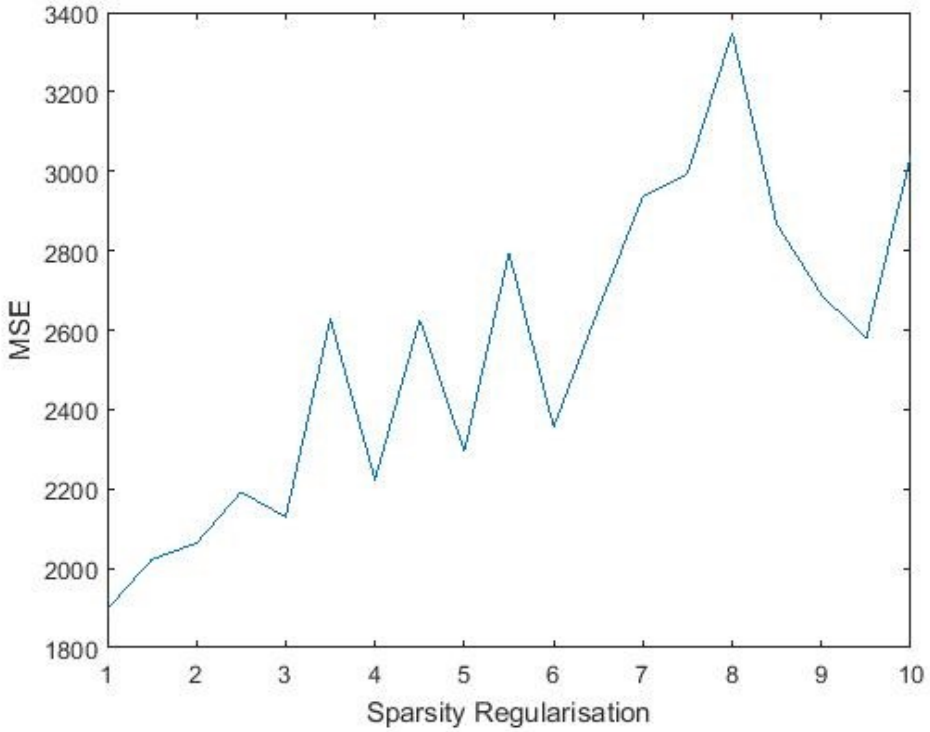

Figure 18: Sparsity Regularisation Against MSE

It is clear that the best accuracy and the best MSE are not achieved at the same sparsity regularisation. However, it is rather interesting that the worst accuracy and the worse MSE both appear at sparsity regularisation of 8. We choose 1 as our optimal sparsity regularisation because it has the smallest MSE and the second largest accuracy.

3.4.2 Sparsity proportion

Likewise, we look for a particular value of sparsity proportion that (ideally) gives both the largest classification accuracy and the smallest MSE. The results are shown below.

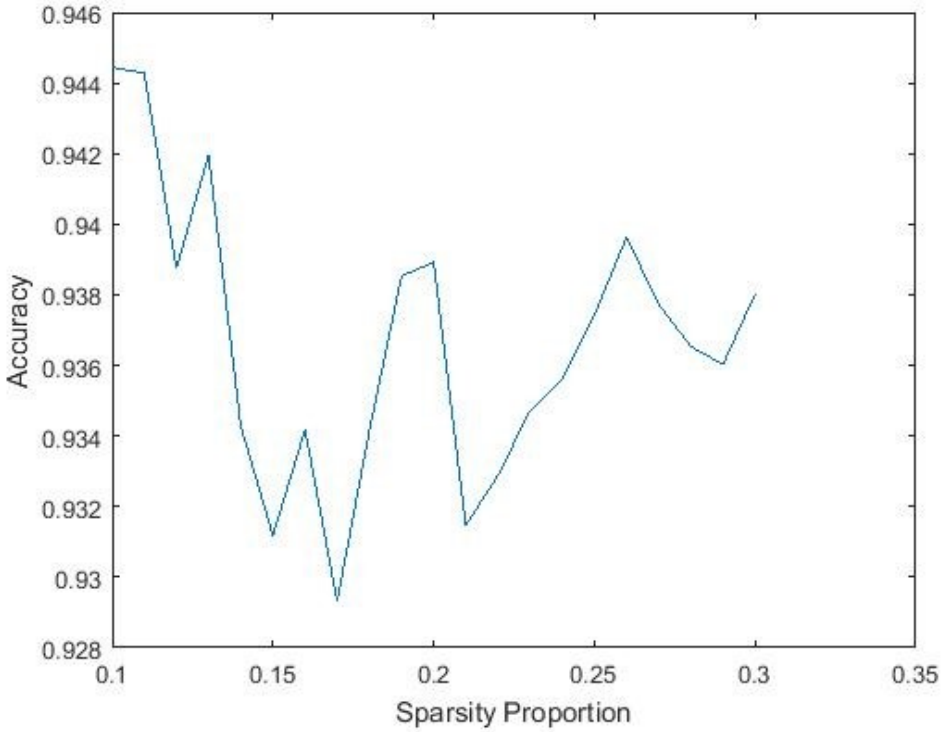

Figure 19: Sparsity Proportion Against Accuracy

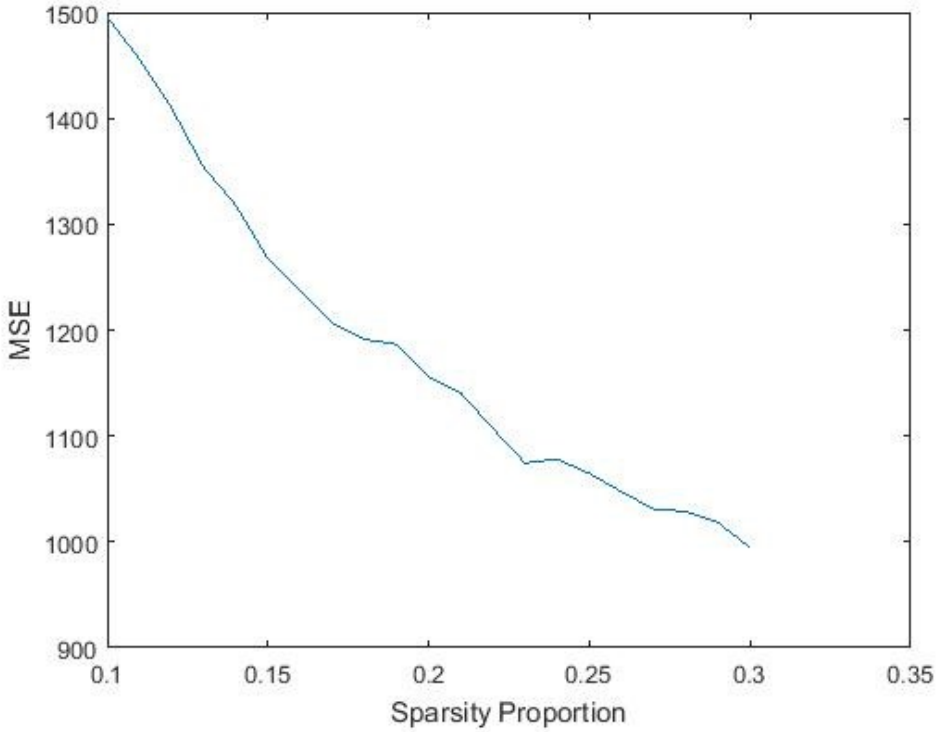

Figure 20: Sparsity Proportion Against Accuracy

Similar to the case of sparsity regularisation, the best classification accuracy and the best MSE are not achieved at the same sparsity proportion value. We decide to choose 0.3 as our sparsity regularisation because it gives the best MSE and a reasonably good classification accuracy.

3.4.3 Hidden layer size

To summarise, we have had so far the following parameter configurations:

- `Maximum epochs=200`
- `Sparsity proportion=0.3`
- `Sparsity regularization=1`

Our deep ANN has two hidden layers. We allow the size of the first hidden layer to vary from 80 to 120 and the size of the second hidden layer to vary

from 30 to 70. We plot the classification accuracy and MSE against the two hidden layer sizes as following:

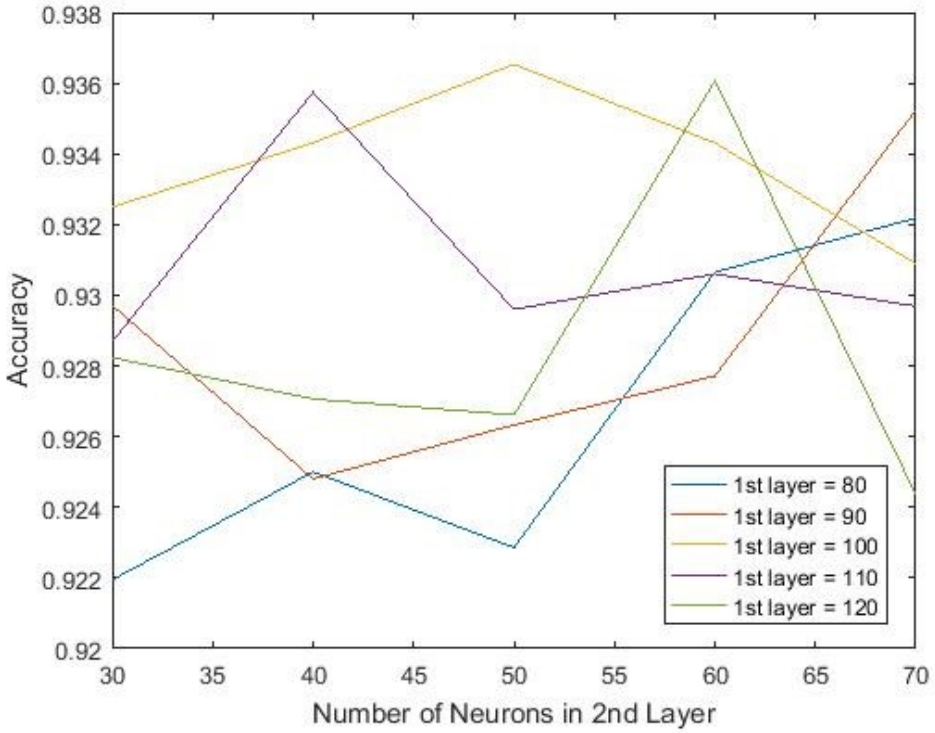

Figure 21: Hidden Layer Sizes Against Accuracy

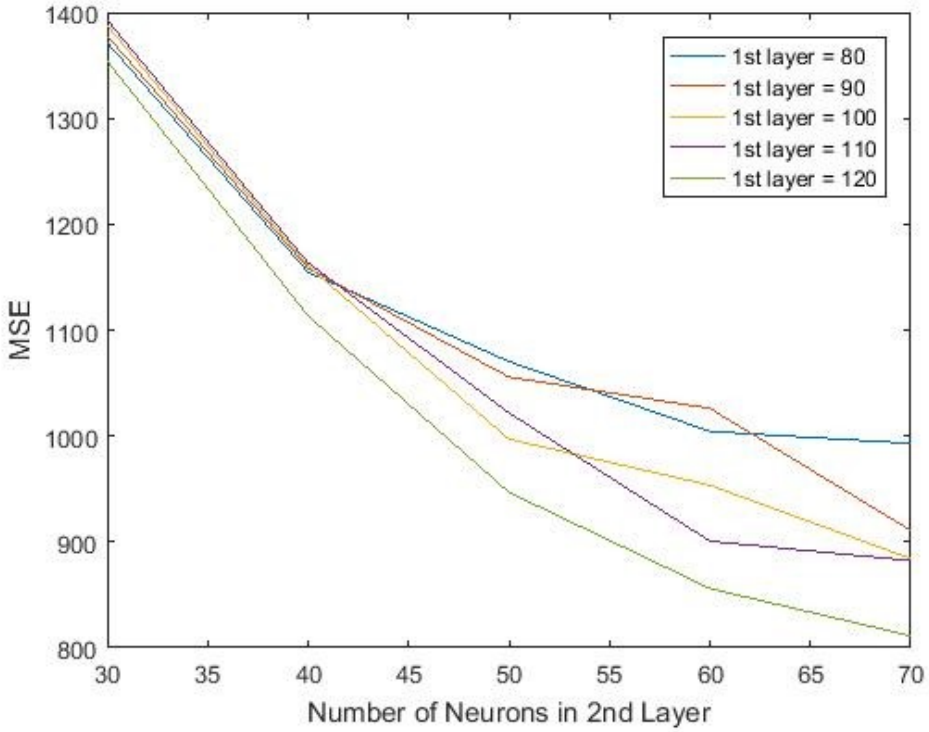

Figure 22: Hidden Layer Sizes Against MSE

It can be observed that the default hidden layer size configuration (first hidden layer of 100 and second hidden layer of 50) does leads to the best classification accuracy (around 93.7%) but a high MSE (around 890) at the same time. Instead, we choose the first hidden layer size to be 120 and the second to be 60, because this configuration gives the second best classification accuracy (around 93.6%) and the best MSE (around 810).

3.4.4 Summary on Deep ANN

To summarise, we have the following parameter configuration for the deep ANN:

Parameter	Optimal Value
Maximum Epochs	200
Sparsity Proportion	0.3
Sparsity Regularization	1
First hidden layer size	120
Second hidden layer size	60
Encoder transfer function	logsig
Decider transfer function	logsig

Table 3: Optimal Parameters for Single Autoencoder

The best accuracy we obtained is 93.7%, and the best MSE is 810. (Run part2_c_final.m)