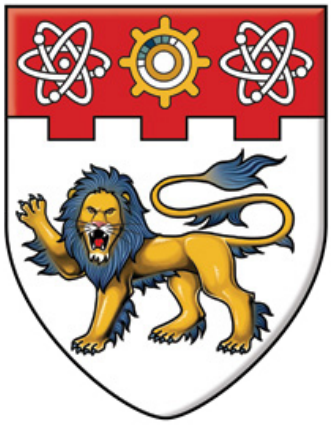

NANYANG TECHNOLOGICAL UNIVERSITY

CZ4042 NEURAL NETWORKS
ASSIGNMENT 1 REPORT

EDWIN CANDINEGARA
U1320135K

AY 2016/17

School of Computer Science and Engineering

1. Introduction

The main objective of the first assignment of the CZ4042 Neural Networks course is to give a hands-on experience to students on how to write programs for building a neural network with different configurations. Moreover, this assignment allows students to understand how different configuration variables such as learning rate, number of hidden layers, number of neurons in each hidden layer, and number of epoch affect the performance of the neural network model.

There are two problems to be solved in this assignment. The first one is a classification problem in which given a feature vector of an email, the neural network should be able to identify whether it is a spam or not. The second problem is a regression problem in which given a feature of a house in California, the neural network should be able to predict the price of the house.

2. Libraries and Tools

For this assignment, the programming language Python is used in conjunction with multiple well-known libraries for data manipulation as well as building a neural network model. The libraries used are as follows:

a. Pandas

Pandas is one of the best data manipulation and analysis tools for Python and widely used in data analytics and machine learning area. Using Pandas, we could load data in the form of a DataFrame object and perform calculations and/or manipulations on each row or column easily.

b. NumPy

NumPy is a well-known library in Python which adds supports for multi-dimensional arrays and matrices as well as some other data analytics related functions. In this assignment, this library is only used for shuffling the rows of the raw data.

c. Keras

Keras is a minimalist neural networks library for Python. Its backend engine can run on top of both Theano and TensorFlow. This library is quite easy to use and to tweak, enabling fast experimentation.

d. Scikit-learn

Scikit-learn is one of the most popular machine learning toolkit for Python. For this assignment, this library is used only for splitting training and testing data.

3. Problem 1 – Spam Base

a. Dataset Overview

The first problem dataset contains 4601 rows of features extracted from emails. Out of those 4601 rows, 1813 are classified as spams. There are 57 distinct attributes most of which are real values and the rest are integer values, and 1 class label. This is clearly a classification problem.

b. Neural Networks Architecture

1) Multi-Layer Perceptron (MLP)

The architecture used in the neural networks is based on the well-known Multi-Layer Perceptron (MLP). A neural network based on MLP architecture has at least 1 hidden layer in between the input and output layer. For the experiments, the number of hidden layers and the number of neurons per hidden layer may be changed in each experiment in order to find the best network configuration. In addition, each neuron in a layer will be fully connected to the neurons in the next layer.

Since there are 57 distinct attributes for each data row, there are 57 input neurons as well. Moreover, as this is a classification problem with only 2 possible classes, the neural network only needs 1 output neuron which will give the probability of being a spam email or not. Figure 1 illustrates how the network looks like.

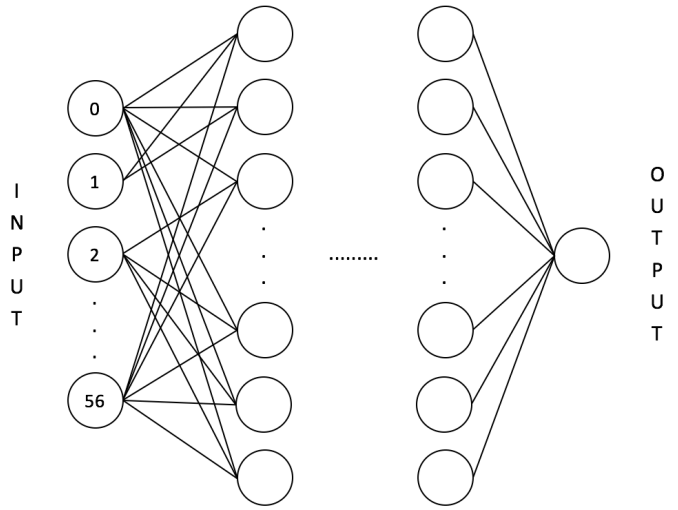

Figure 1: The neural network with 57 input neurons, variable number of hidden layers, number of neurons per hidden layer, and 1 output neuron.

2) Learning Algorithms

The learning algorithm in this neural network model is based on the Mini-batch Gradient Descent. In the traditional Stochastic Gradient Descent, the update on the network's weights is performed for each training example. On the other hand, in the traditional Batch Gradient Descent, the update on the network's weights is performed for the entire training examples. The Mini-batch Gradient Descent method takes the best of both traditional methods. It performs an update on the network's weights for every mini batch of N training examples.

Traditional Stochastic Gradient Descent is too slow as it updates the weights for every single example in each epoch. Although traditional Batch Gradient Descent algorithm is much faster than the Stochastic Gradient Descent, the update on the network's weights can be too big if the learning rate is not set properly and hence may result in not getting the optimal weights. By using the Mini-batch Gradient Descent algorithm, the computations take less time while each weights update will not be too big.

Regardless of which variants of Gradient Descent algorithms, the basic equation for updating the weights is:

$$\boldsymbol{w}_{new} = \boldsymbol{w}_{old} - \alpha \frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}}$$

The only difference between the 3 variants of Gradient Descent algorithms is simply how frequent it updates the network's weights. For this problem, the value of N is 32, meaning that the network's weights will be updated after processing 32 examples of the training data.

3) Loss Function

The loss function used for training the neural network model is the Mean Squared Error. The equation for Mean Squared Error is as follow:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - d_i)^2$$

where d_i is the desired output and y_i is the predicted output.

4) Activation Function

As this is a classification problem, the targeted value range for the output node is from 0 to 1. If the output is less than or equal to 0.5, it is considered as not a spam email. If the output is bigger than 0.5, it is considered as a spam email. The output node is using a unipolar sigmoid activation function. Furthermore, each neuron in each hidden layer is also using a unipolar sigmoid activation function.

c. Experimentation Method

There are 4 variables whose behaviours are going to be observed in this assignment. The 4 variables are the learning rate, the maximum epoch, the number of hidden layers, and the number of neurons per hidden layer. In order to understand how increasing or decreasing the value of one variable may affect the neural network performance, when exploring one variable, all other variables will be set to one fixed value.

Before training the model, the data is split into 70% training data and 30% testing data. The 30% testing data will only be used for getting the final error using a neural network model trained with the 70% training data using the best configuration. Additionally, the best value for one variable is found by using 3-Fold Cross Validation method, meaning there will be 3 pairs of training (66.67% out of the 70% training data) and validation (the remaining 33.33% out of the 70% training data) sets, and 3 evaluation results obtained for each value. The best value for that variable is chosen based on the average accuracy and loss.

After the data has been split, the predictor variables are normalized so that the resulting values corresponds to zero mean and unit standard deviation. The mean and standard deviation used in the normalization is calculated from the training data set (70%). This mean and standard deviation value is also used for normalizing the testing data set (30%).

d. Results

1) Learning Rate

There are 6 learning rate values tested in this experiment. The 6 learning values are 0.0001, 0.001, 0.01, 0.1, 0.5, and 1. This experiment is using 1 hidden layer with 50 neurons and the maximum epoch is 3000.

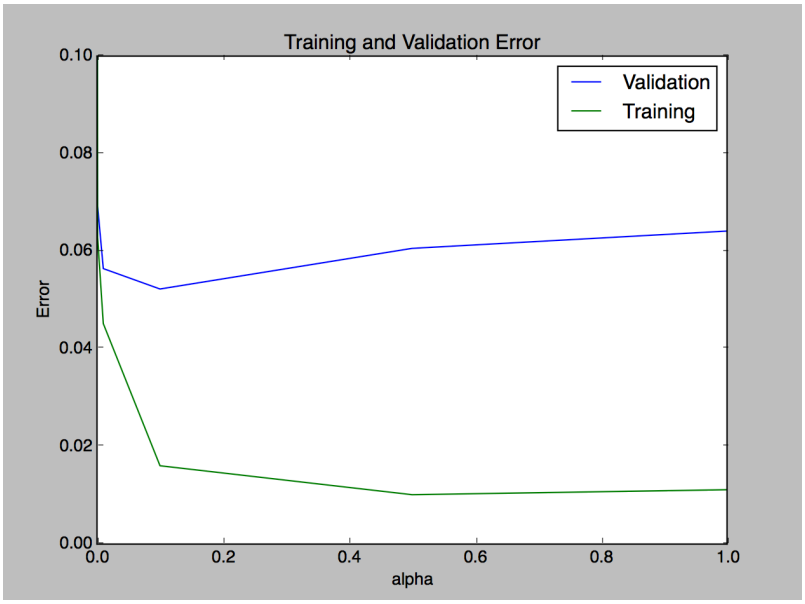

Figure 2: The training and validation set loss for different learning rate values.

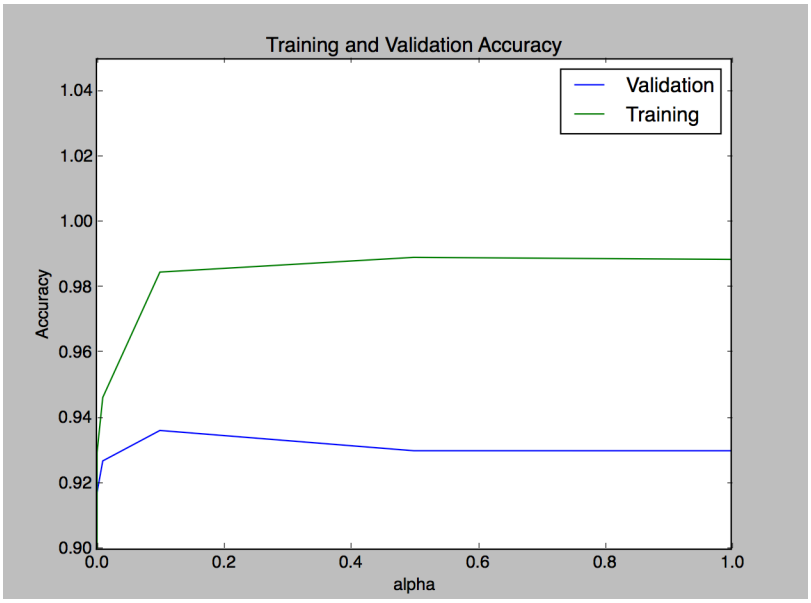

Figure 3: The training and validation set accuracy for different learning rate values.

From Figure 2 and 3, we can observe that when the learning rate is very small, as small as 0.0001, the loss and accuracy are not good. This is expected since the network updates its weights very slowly and the loss may not have converged fully by the time the training algorithm reaches the maximum epoch. When the learning rate is too big, the network's weights are updated more quickly. This would sometimes result in missing the minima of the loss function. Thus, the accuracy and loss are worse as well. The value 0.1 is the best learning rate value. We can see that when the learning rate is 0.1, the validation set has the highest accuracy and the lowest loss.

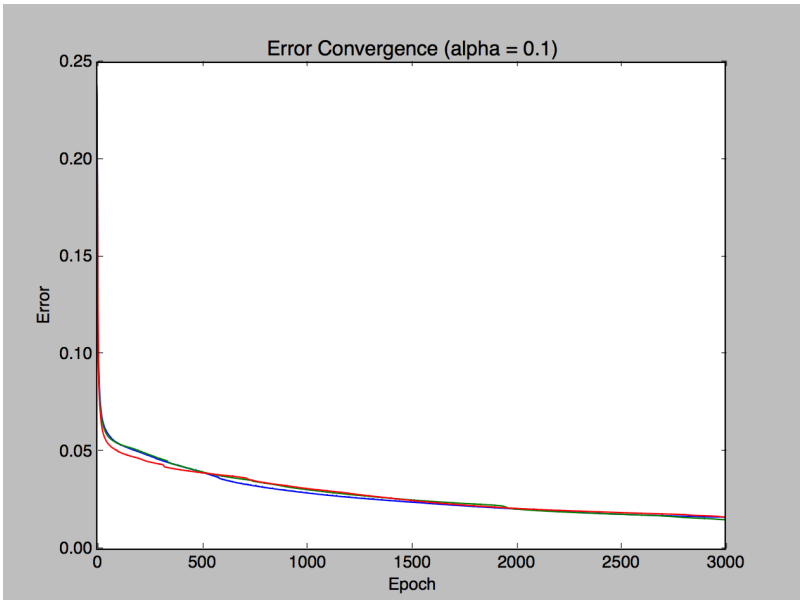

Figure 4: Loss convergence graph for 3-fold cross validation when the learning rate is 0.1.

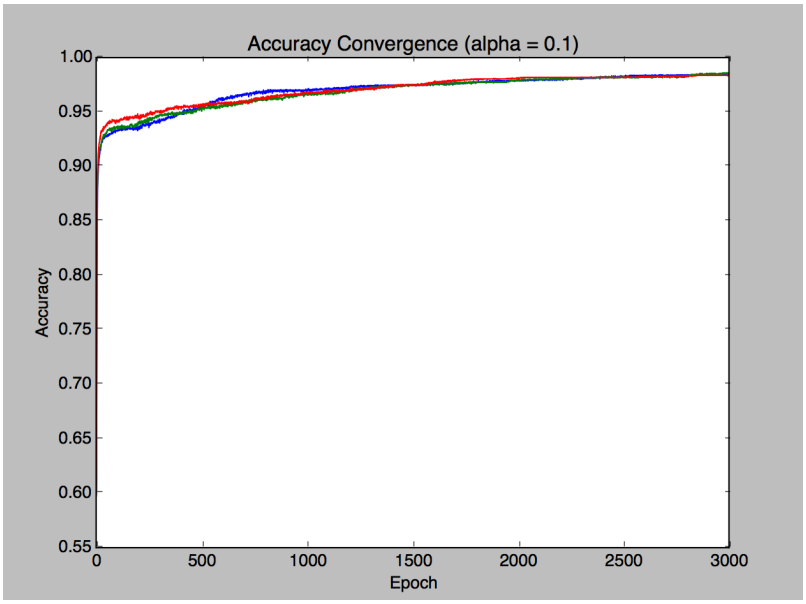

Figure 5: Accuracy convergence graph for 3-fold cross validation when the learning rate is 0.1.

2) Maximum Epoch

There are 8 maximum epoch values tested in this experiment: 100, 500, 1000, 2000, 3000, 5000, 7500, and 10000. This experiment is using 1 hidden layer with 50 neurons and the learning rate is 0.1.

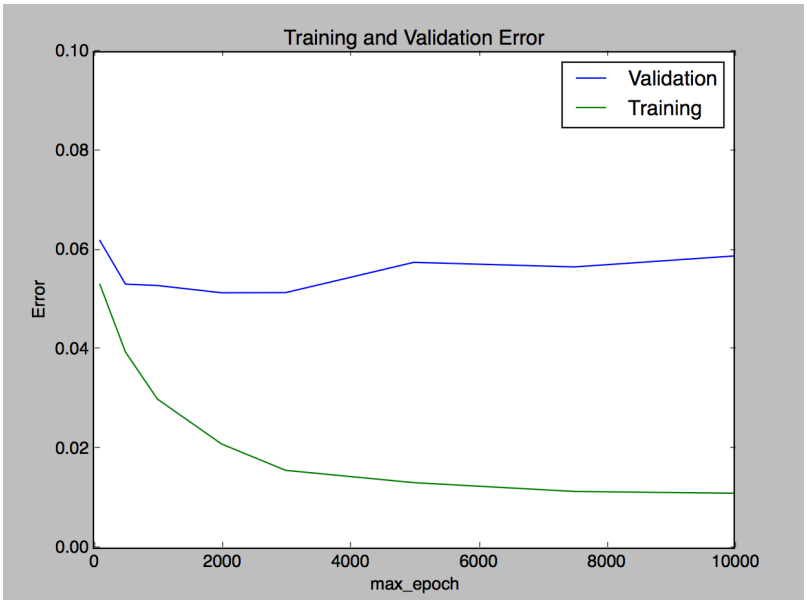

Figure 6: The training and validation set loss for different maximum epoch values.

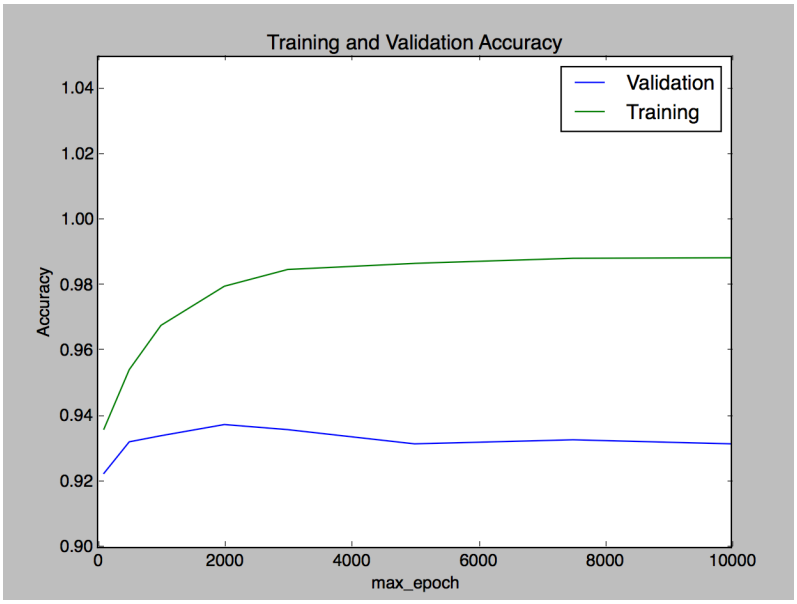

Figure 7: The training and validation set accuracy for different maximum epoch values.

When the learning rate is small enough and the value of maximum epoch is big, the network may overfit the training data. This is observable from Figure 6 and 7. After around 3000 epoch, the training accuracy and loss are getting better while the testing accuracy and loss are getting worse. The value 3000 should be good enough for the neural network.

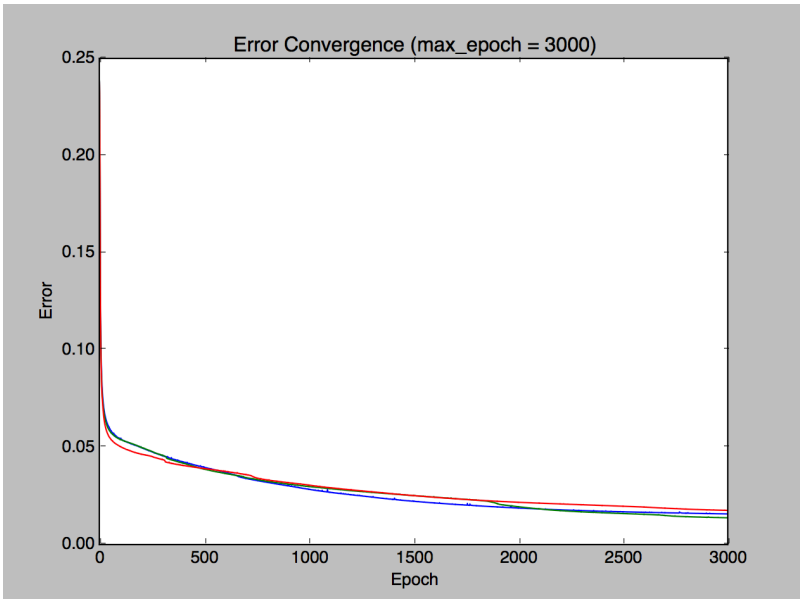

Figure 8: Loss convergence graph for 3-fold cross validation when the maximum epoch is 3000.

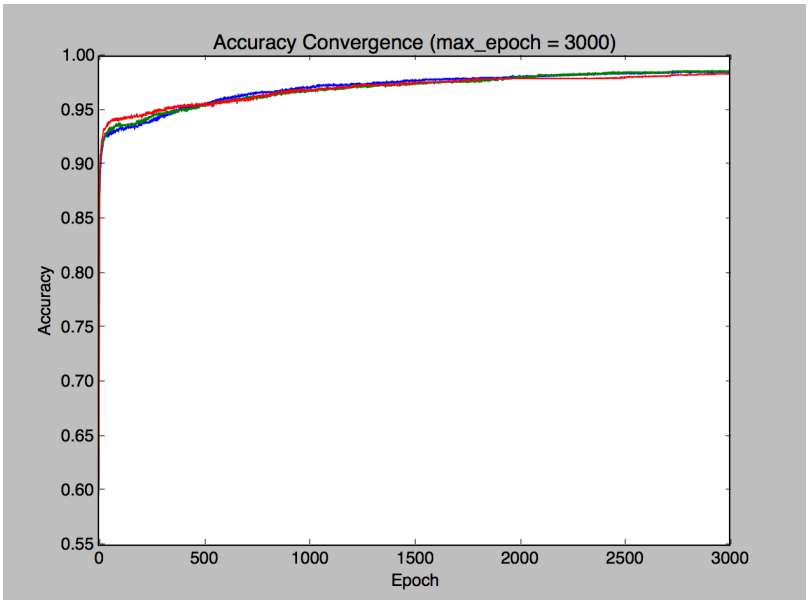

Figure 9: Accuracy convergence graph for 3-fold cross validation when the maximum epoch is 3000.

3) Number of Hidden layers

There are 4 number of hidden layer values tested in this experiment: 1, 2, 3, and 4. This experiment is using 50 neurons per hidden layer, the learning rate is 0.1, and the maximum epoch is 3000.

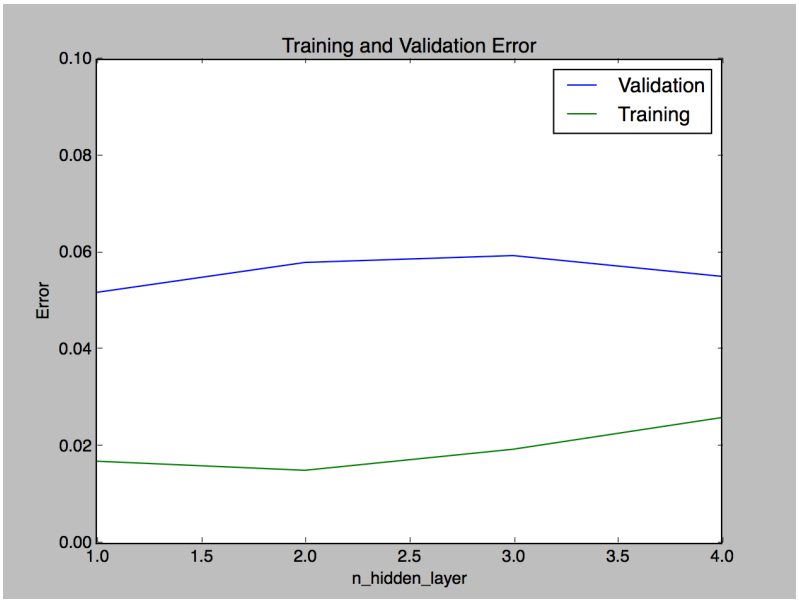

Figure 10: The training and validation set loss for different number of hidden layers.

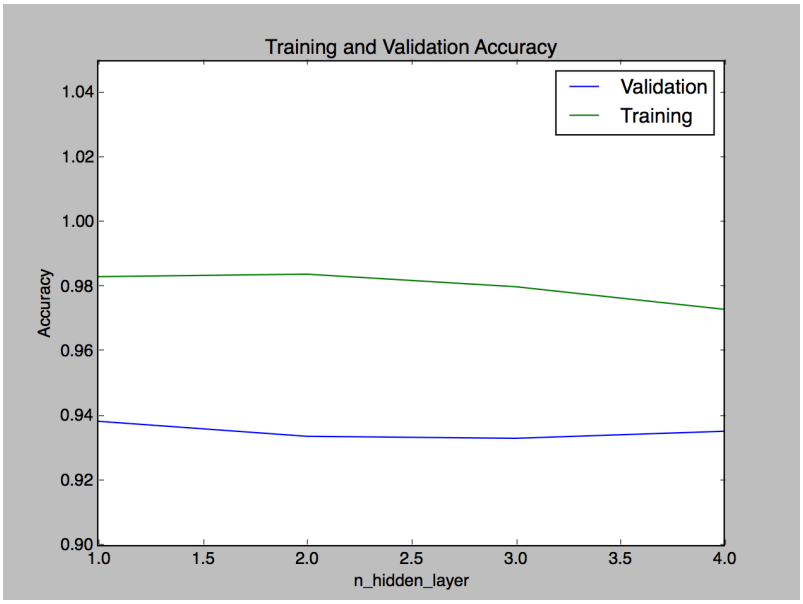

Figure 11: The training and validation set accuracy for different number of hidden layers.

Neural networks with higher number of hidden layers are usually used for solving a function with higher degree of non-linearity. In this case, the problem's function may not have a high degree of non-linearity. Hence, the neural network may be able to solve the function of the problem using only one hidden layer as can be seen from Figure 10 and 11. When the network is using one hidden layer, it has the highest accuracy and lowest loss value.

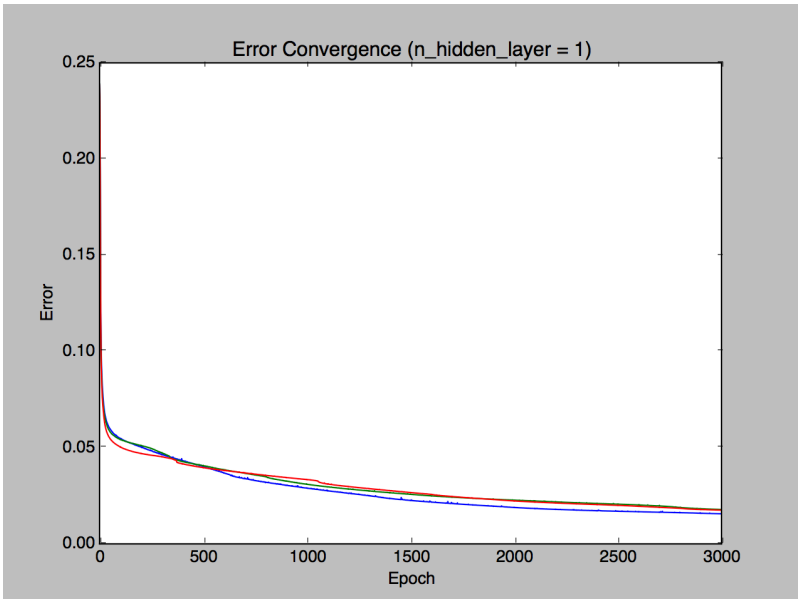

Figure 12: Loss convergence graph for 3-fold cross validation when the number of hidden layer is 1.

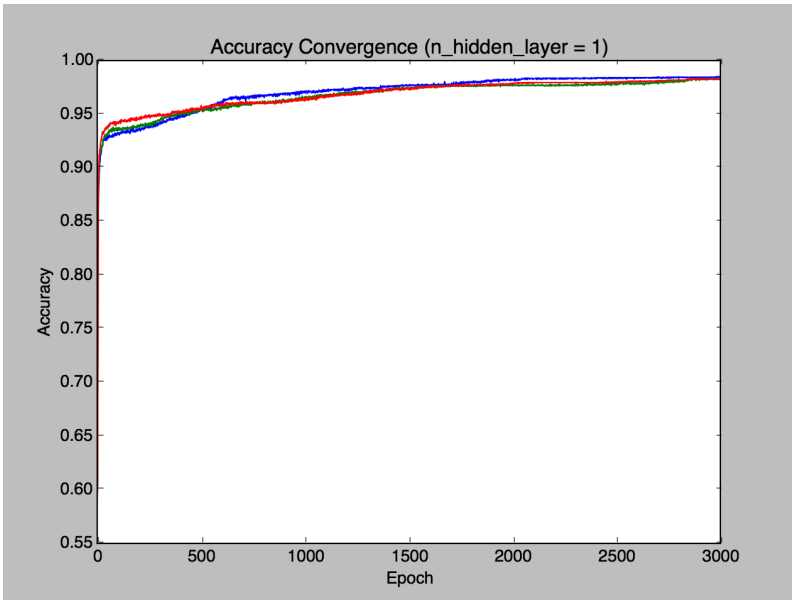

Figure 13: Accuracy convergence graph for 3-fold cross validation when the number of hidden layer is 1.

4) Number of Neurons per Hidden Layer

There are 8 number of neurons per hidden layer values tested in this experiment: 1, 5, 10, 30, 50, 100, 200, and 300. This experiment is using 1 hidden layer, the learning rate is 0.1, and the maximum epoch is 3000.

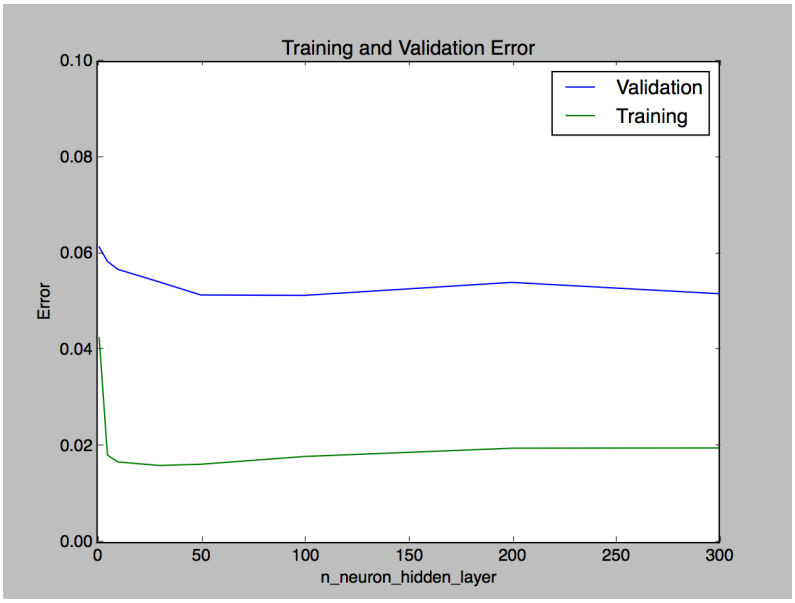

Figure 14: The training and validation set loss for different number of neurons per hidden layer.

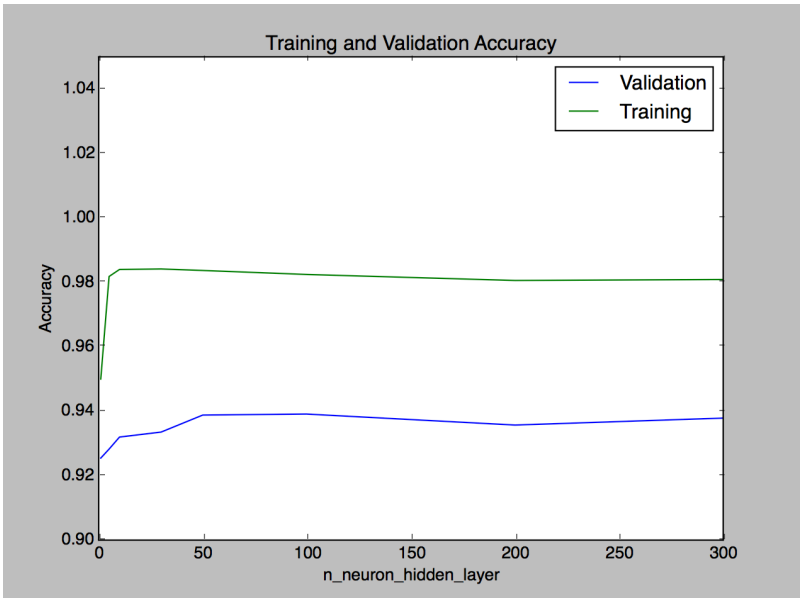

Figure 15: The training and validation set accuracy for different number of neurons per hidden layer.

Neural networks with higher number of neurons per hidden layer can be useful for solving complex functions. Generally, when the number of neurons per hidden layer is higher, it may overfit the training data. However, it may not be the case in this experiment and this data set. Based on Figure 14 and 15, there is no huge difference in terms of the accuracy and loss between having 50 and 300 neurons in the hidden layer. In order to reduce computations needed, 50 neurons per hidden layer should be good enough.

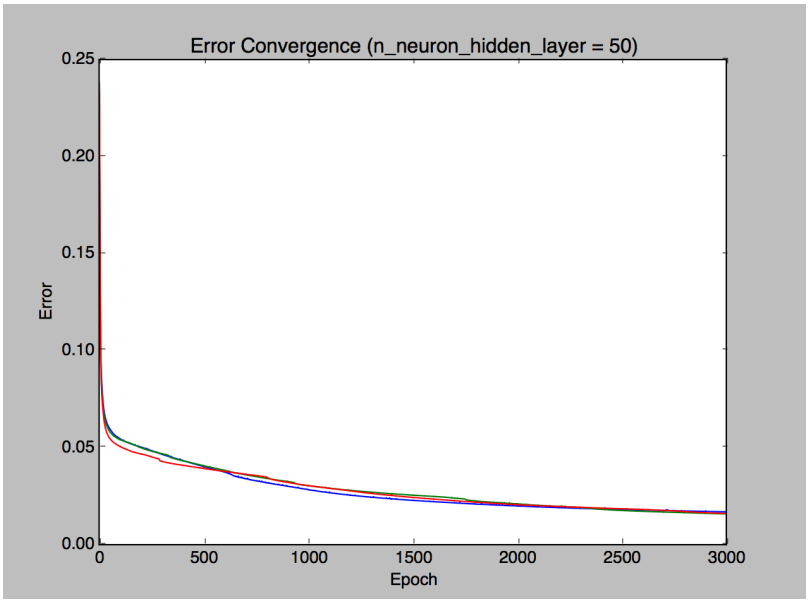

Figure 16: Loss convergence graph for 3-fold cross validation when the number of neurons per hidden layer is 50.

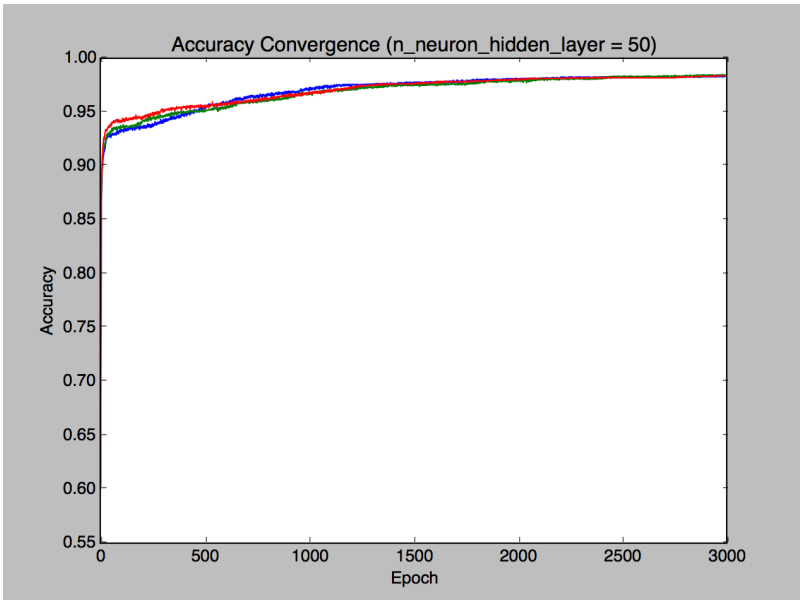

Figure 17: Accuracy convergence graph for 3-fold cross validation when the number of neurons per hidden layer is 50.

e. Best Configuration

Based on the experiments conducted in the previous section, the best neural network configuration for the spam classification problem is:

1) **Learning rate:** 0.1
2) **Maximum epoch:** 3000
3) **Number of hidden layers:** 1
4) **Number of neurons per hidden layer:** 50

Using this configuration, a final neural network model is trained using the full training data set (70% of the whole data set). This final model is then evaluated using the testing data (the remaining 30% of the whole data set) that has been put aside from

the beginning of the experiments. The final loss value is 0.050789 and accuracy value is 0.939899.

4. Problem 2 – California Housing Price

a. Dataset Overview

The second problem dataset contains 20640 rows of house related features in California. There are 8 distinct attributes and also the house price. The target of the neural model is to predict the house price given a number of features related to a house. Thus, this is a regression problem.

b. Neural Networks Architecture

1) Multi-Layer Perceptron (MLP)

The architecture used in the neural networks is based on the well-known Multi-Layer Perceptron (MLP). Similar to the first problem, the number of hidden layers and the number of neurons per hidden layer may be adjusted in each experiment in order to find the best network configuration. Furthermore, each neuron in a layer will be fully connected to the neurons in the next layer.

Since there are 8 distinct attributes for each data row, there are 8 input neurons as well. As for the number of output neurons, only 1 output neuron is needed as this is a regression problem. Figure 18 illustrates how the network looks like.

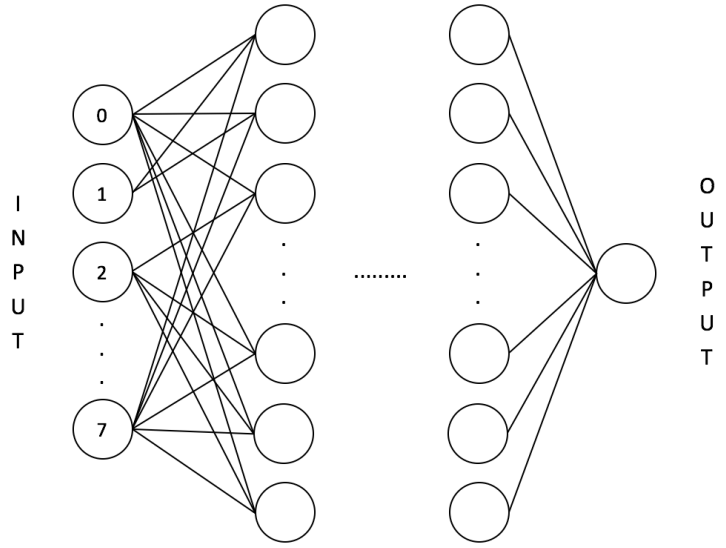

Figure 18: The neural network with 8 input neurons, variable number of hidden layers, number of neurons per hidden layer, and 1 output neuron.

2) Learning Algorithms

The learning algorithm used in problem 2 is called RMSprop. It is a gradient-based optimization algorithm which is capable of adapting the learning rate to the weights value. The adaptation depends on the previous gradients computed for a weight. In fact, RMSprop is a method proposed by Geoff Hinton in Lecture 6e of his Coursera class and it is unpublished [1].

The main idea of RMSprop is to scale the gradient by keeping a running average of recent gradient values squared and dividing each of the current gradient's

components by the square root of the running average [2]. The formula for updating the network's weights using RMSprop is:

$$\boldsymbol{r_t} = (1-\gamma)\left(\frac{\partial E(\boldsymbol{w_t})}{\partial \boldsymbol{w_t}}\right)^2 + \gamma\, \boldsymbol{r_{t-1}}$$

$$\boldsymbol{w}_{new} = \boldsymbol{w}_{old} - \frac{\alpha}{\sqrt{\boldsymbol{r}_t} + \epsilon}\frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}}$$

where $\gamma \in (0, 1)$, $\boldsymbol{r_t}$ is the mean square vector, α is the learning rate, and ϵ is a small term to prevent numerical overflow [2].

The updating of the network's weights is also done in mini-batches. For this problem, the mini-batch size is 256. This reduces the number of computations needed while keeping the size of one weight update small enough.

3) Loss Function

The loss function used for training the neural network model is the Mean Squared Error. The equation for Mean Squared Error is as follow:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - d_i)^2$$

where d_i is the desired output and y_i is the predicted output.

4) Activation Function

As this is a regression problem, there is no limit for the predicted house price. Hence, all neurons, including those in the hidden layers and output layer, are using linear activation function.

c. Experimentation Method

The experiments are conducted in a similar way to those in the first problem. There are 4 variables to explore which are the learning rate, the maximum epoch, the number of hidden layers, and the number of neurons per hidden layer. The experiments will be done by only tweaking a variable while fixing all other variables.

The data is also split into 70% training and 30% testing data. The 70% training data will be used for building models with different configurations using 3-Fold Cross Validation. Once the best configuration is found, a model will be built using the best configuration and trained on the 70% training data. Finally, the final testing error will be obtained by evaluating the final model on the 30% testing data.

After the data has been split, the predictor variables are normalized so that the resulting values corresponds to zero mean and unit standard deviation. The mean and standard deviation used in the normalization is calculated from the training data set (70%). This mean and standard deviation value is also used for normalizing the testing data set (30%). Hence, please note that the prediction result and the loss value are both based on the normalized housing price value.

Please note that there is no accuracy graph shown for problem 2 as by definition, the accuracy will most likely be 0.

d. Results

1) Learning Rate

There are 6 learning rate values tested in this experiment. The 6 learning values are 0.00001, 0.0001, 0.001, 0.01, 0.1, and 0.5. This experiment is using 2 hidden layers with 15 neurons in each layer and the maximum epoch is 3000.

Figure 19: The training and validation set loss for different learning rate values.

From Figure 19, we can observe that when the learning rate is quite big, the loss value will be very high. This is expected as the batch size is pretty big and the network updates its weights quickly. As a result, the network misses the optimal weights easily and the loss increases a lot. This behaviour can be seen from Figure 20. In an epoch, when the learning rate is 0.5, the loss may fluctuate aggressively.

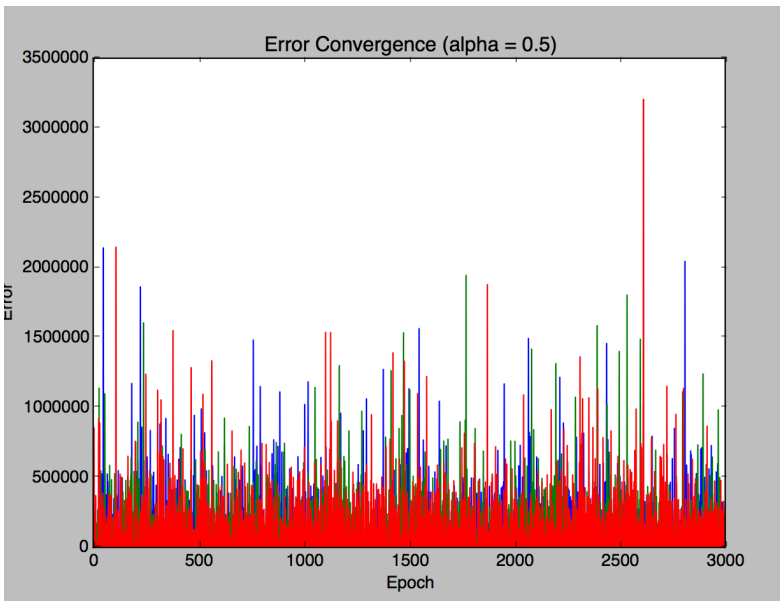

Figure 20: Loss convergence graph for learning rate = 0.5.

Based on the experiment, the value 0.0001 is chosen to be the best learning rate value for this problem.

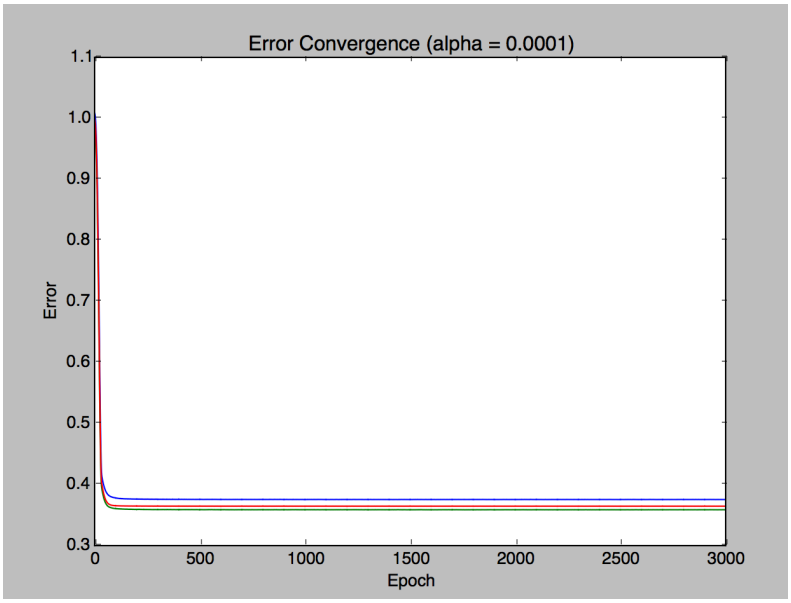

Figure 21: Loss convergence graph for 3-fold cross validation when the learning rate is 0.0001.

2) Maximum Epoch

There are 8 maximum epoch values tested in this experiment. The tested 8 maximum epoch values are 100, 500, 1000, 2000, 3000, 5000, 7500, and 10000. This experiment is using 2 hidden layers with 15 neurons and the learning rate is 0.0001.

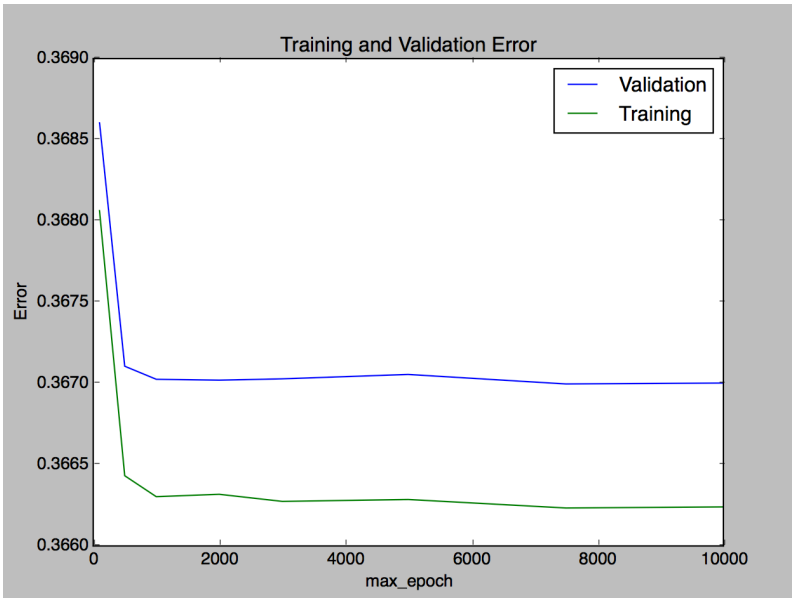

Figure 22: The training and validation set loss for different maximum epoch values.

From Figure 19, we can observe that when the maximum epoch value is too small, the loss is higher. This is expected as the network has not learnt all the examples well by the time it hits the maximum epoch limit. When the maximum epoch value is increased beyond 2000 epoch, there is no big difference in terms of the loss value. Based on the experiment, the value 3000 epoch should be good enough.

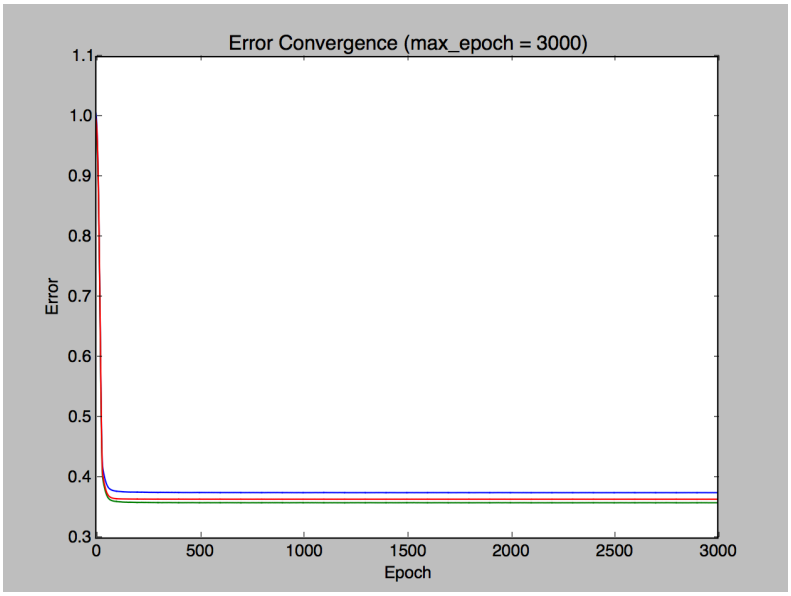

Figure 23: Loss convergence graph for 3-fold cross validation when the maximum epoch is 3000.

3) Number of Hidden Layers

There are 4 number of hidden layers tested in this experiment. Those are 1, 2, 3, and 4. This experiment is using 15 neurons per hidden layer, the maximum epoch is 3000, and the learning rate is 0.0001.

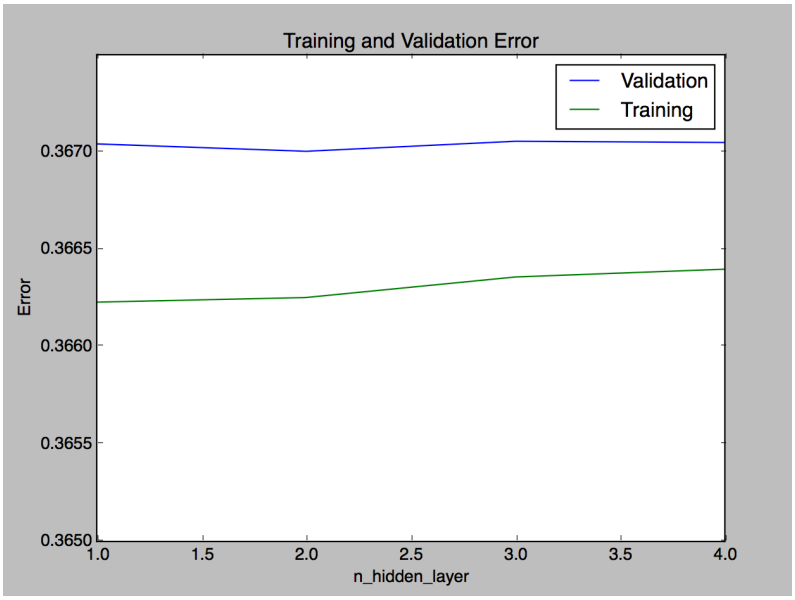

Figure 24: The training and validation set loss for different number of hidden layers.

From Figure 24, we can observe that there is no significant difference in the loss value when the number of hidden layers is increased. It seems this problem does not require high degree of non-linearity to predict the house price. Thus, with 2 hidden layers, the performance of the network is pretty good.

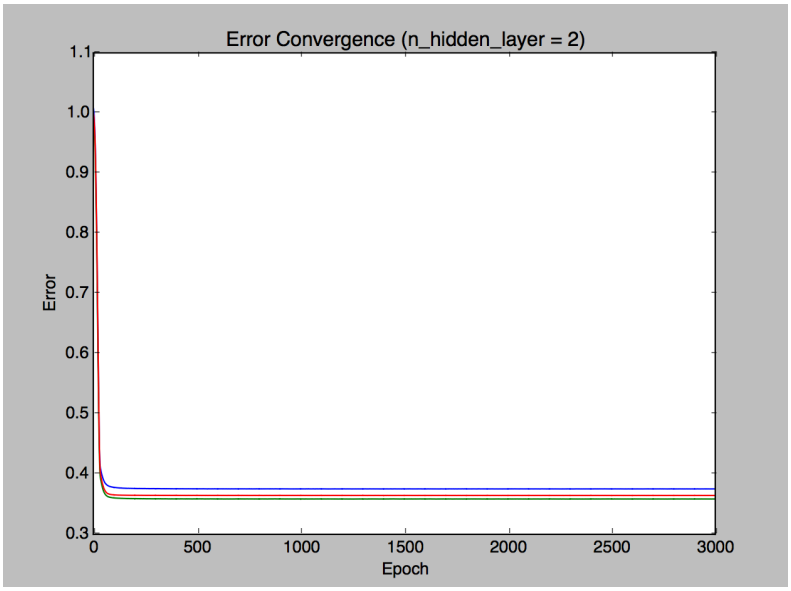

Figure 25: Loss convergence graph for 3-fold cross validation when the number of hidden layers is 2.

4) Number of Neurons per Hidden Layer

There are 7 number of neurons per hidden layer tested in this experiment. Those values are 1, 3, 5, 10, 15, 30, and 50. This experiment is using 2 hidden layers, the maximum epoch is 3000, and the learning rate is 0.0001.

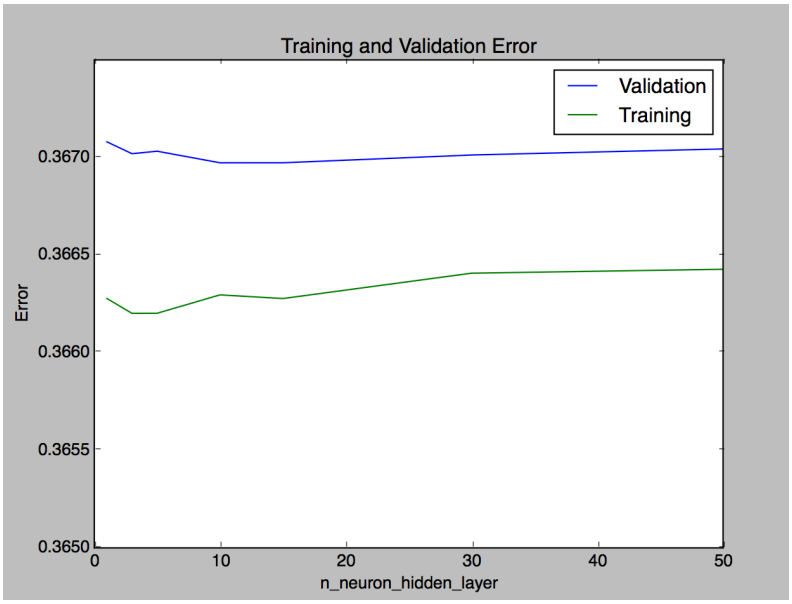

Figure 26: The training and validation set loss for different number of neurons per hidden layer.

From Figure 26, we can observe that when the number of neurons per hidden layer is increased, the loss is increasing as well. 15 neurons per hidden layer is reasonably good and it has the lowest validation loss value as well.

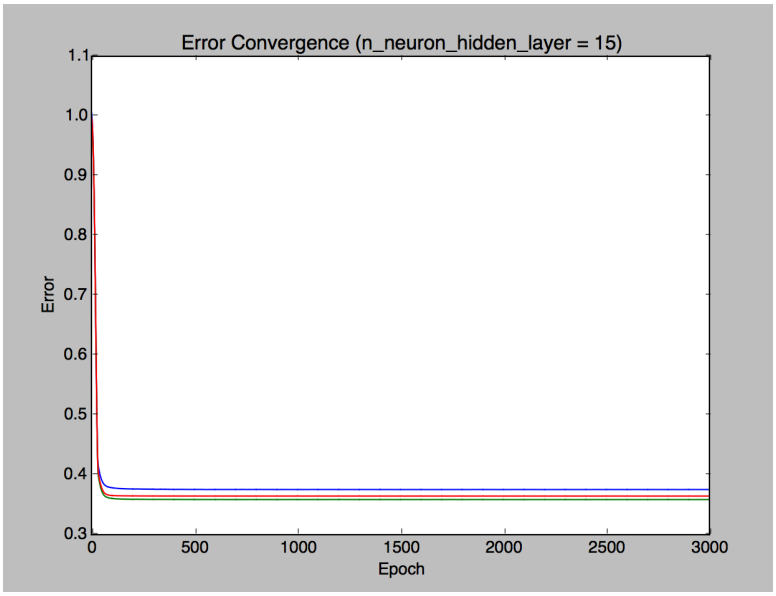

Figure 27: Loss convergence graph for 3-fold cross validation when the number of neurons per hidden layer is 15.

e. Best Configuration

Based on the experiments conducted in the previous section, the best neural network configuration for the housing price regression problem is:

1) **Learning rate:** 0.0001
2) **Maximum epoch:** 3000
3) **Number of hidden layers:** 2
4) **Number of neurons per hidden layer:** 15

Using this configuration, a final neural network model is trained using the full training data set (70% of the whole data set). This final model is then evaluated using the testing data (the remaining 30% of the whole data set) that has been put aside from the beginning of the experiments. The final loss value is 0.376234 (note that the loss value is based on the predicted normalized house price).

5. Problem 2 – California Housing Price (RBF Neural Network)

a. Neural Networks Architecture

1) Kernel-based Neural Network

A kernel-based neural network is a type of neural network which is an approach to approximate a non-linear function by representing the function using a linear combination of fixed non-linear functions known as basis functions. This type of neural network only has 1 input layer, 1 hidden layer, and 1 output layer. As this neural network model is trying to solve the same problem in the previous section, there are 8 input neurons and 1 output neuron.

In a kernel-based neural network, the weights between the input layer and the hidden layer are all the same. However, the weights between the hidden layer and the output are determined based on the training data.

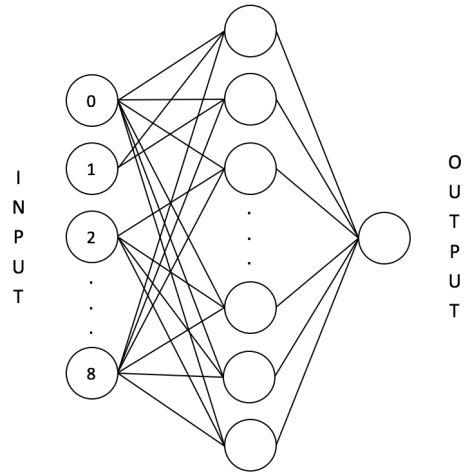

Figure 28: The kernel-based neural network with 8 input neurons, 1 hidden layer, variable number of neurons per hidden layer, and 1 output neuron.

2) Basis Functions

Each neuron in the hidden layer has a basis function which will vary in different part of the experiments. There are 4 basis functions involved in this assignment:

a) Piecewise linear approximation

$$\phi_k(\| \boldsymbol{x} - \boldsymbol{\mu_k} \|) = \| \boldsymbol{x} - \boldsymbol{\mu_k} \|$$

b) Cubic approximation

$$\phi_k(\| \boldsymbol{x} - \boldsymbol{\mu_k} \|) = \| \boldsymbol{x} - \boldsymbol{\mu_k} \|^3$$

c) Thin plate splines

$$\phi_k(\| \boldsymbol{x} - \boldsymbol{\mu_k} \|) = \| \boldsymbol{x} - \boldsymbol{\mu_k} \|^2 \ log(\| \boldsymbol{x} - \boldsymbol{\mu_k} \|)$$

d) Gaussian function

$$\phi_k(\| \boldsymbol{x} - \boldsymbol{\mu_k} \|) = e^{-0.5\times(\boldsymbol{x}-\boldsymbol{\mu_k})^T \boldsymbol{C_k}^{-1}(\boldsymbol{x}-\boldsymbol{\mu_k})}$$

$$\boldsymbol{C_k} = \frac{1}{P-1}\sum_{p=1}^{P}(\boldsymbol{x}_p - \boldsymbol{\mu_k})(\boldsymbol{x}_p - \boldsymbol{\mu_k})^T$$

$$\boldsymbol{\mu_k} = \frac{1}{P}\sum_{p=1}^{P}\boldsymbol{x_p}$$

where $\boldsymbol{\mu_k}$ is the centroid of a point cluster, $\boldsymbol{x_p}$ is a point that belongs to that cluster, and P is the number of points that belong to that cluster.

3) Loss Function

The loss function used for training the neural network model is the Mean Squared Error. The equation for Mean Squared Error is as follow:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - d_i)^2$$

where d_i is the desired output and y_i is the predicted output.

4) Network Training

The weights between the hidden layer and the output layer can be calculated using this formula:

$$\boldsymbol{W} = (\boldsymbol{Z}^T\boldsymbol{Z})^{-1}\boldsymbol{Z}^T\boldsymbol{d}$$

where **W** is the weight matrix, **Z** is the hidden layer output matrix, and **d** is the desired output from the training examples.

b. Experimentation Method

In this RBF experiment, 3-ways data split method is used. The whole data is split into 3 sets: training set (70%), validation set (15%), and testing set (15%). After the data has been split, the predictor variables are normalized so that the resulting values corresponds to zero mean and unit standard deviation. The mean and standard deviation used in the normalization is calculated from the training data set (70%). This mean and standard deviation value is also used for normalizing the validation (15%) and testing data set (15%). Hence, please note that the prediction result and the loss value are both based on the normalized housing price value.

c. Results

There are 4 basis functions and 16 different number of hidden neurons tested in this experiment. The basis functions are piecewise linear, cubic, thin plate splines, and Gaussian function. The 16 different number of hidden neurons values are 1, 2, 3, 4, 5, 10, 15, 20, 25, 30, 50, 75, 100, 300, 500, and 1000.

However, please note that due to calculation overflow exception when using Gaussian basis function with high number of hidden neurons, the number of hidden neurons tested is more restricted. Hence, only 7 number of hidden neurons values are tested for Gaussian basis function. Those values are 1, 2, 3, 4, 5, 10, and 15.

1) Piecewise Linear Approximation

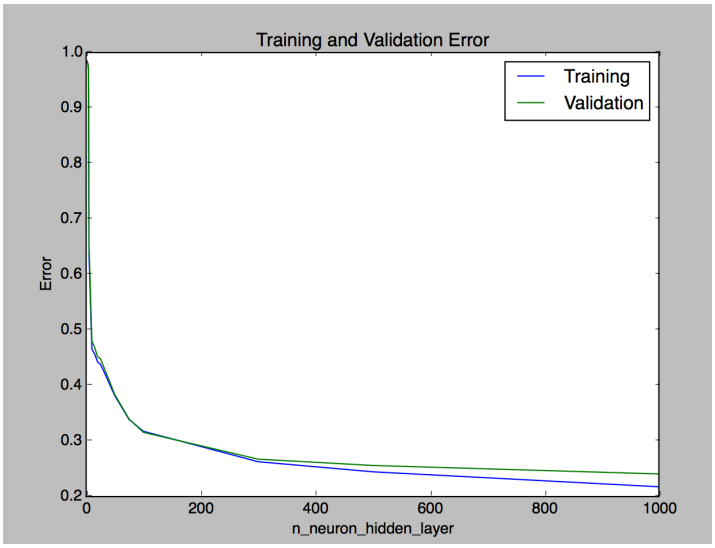

Figure 29: The training and validation set loss for different number of hidden neurons when using piecewise linear approximation.

The loss values for both training and validation set are decreasing as the number of hidden neurons is increasing. Furthermore, when the number of hidden neurons is above 200, the loss is smaller than if the neural network is a multi-layer perceptron. In addition, as the number of hidden neurons is increased even more, the training loss is still decreasing while the validation loss does not change much, indicating the network seems to start overfitting the training examples.

2) Cubic Approximation

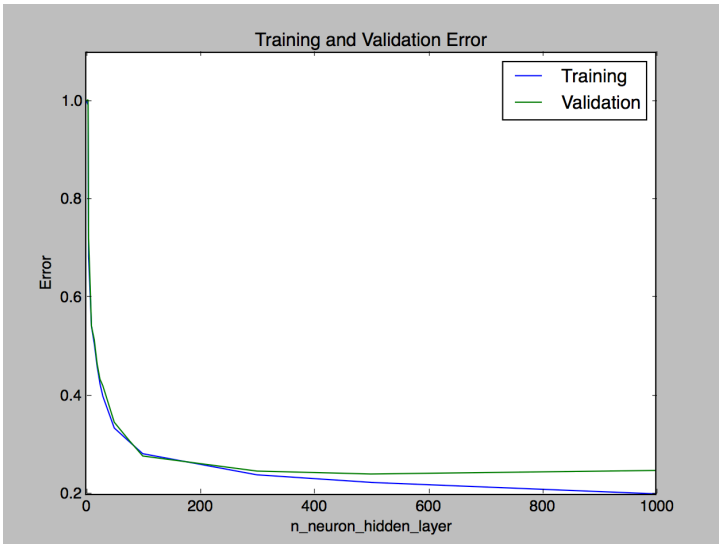

Figure 30: The training and validation set loss for different number of hidden neurons when using cubic approximation.

Similar to the piecewise linear approximation, the loss values for both training and validation set are decreasing as the number of hidden neurons is increasing. Moreover, it has better loss values than MLP if more than 200 hidden neurons are employed. It also shows a sign of overfitting as the number of hidden neurons is close to 1000.

3) Thin Plate Splines

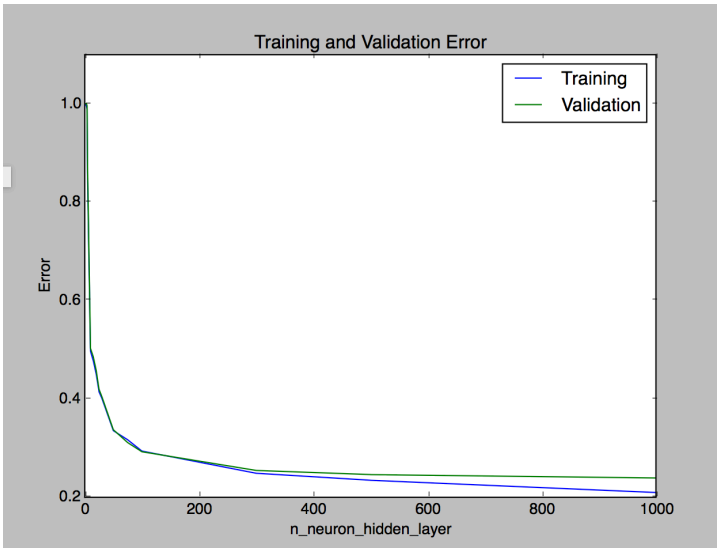

Figure 31: The training and validation set loss for different number of hidden neurons when using thin plate splines basis function.

Thin plate splines basis function also exhibits similar behaviours as piecewise and cubic approximation basis functions. This network also has a better loss value compared to the MLP based network.

4) Gaussian Function

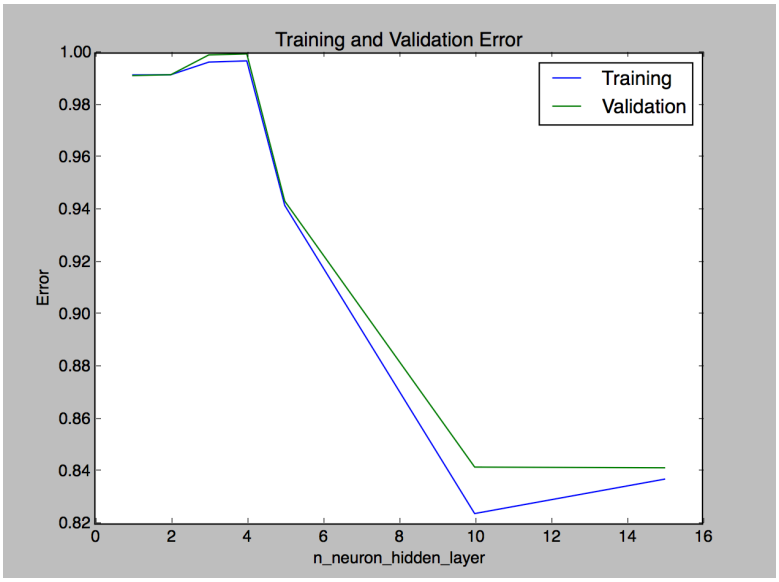

Figure 32: The training and validation set loss for different number of hidden neurons when using Gaussian basis function.

In this experiment, the Gaussian basis function is not tested on a network with more hidden neurons as it will generate a calculation overflow in Python. This issue occurs when the inverse of a covariance matrix is too large and the power term in the Gaussian function will be too big, resulting in overflow.

The network does not perform really well when using a Gaussian basis function as can be seen in Figure 32. In every experiment, the loss value is still higher than that when using a MLP based network model.

d. Best Configuration

Based on the experiments conducted in the previous section, the best neural network configuration for the housing price regression problem using kernel based neural network is:

1) **Basis function:** cubic approximation
2) **Number of neurons:** 300

Although the loss value for both cubic and thin plate splines function are more or less the same, calculating cubic approximation is easier and less demanding in terms of computation power. Using this configuration, a final neural network model is trained using the full training and validation data set (85% of the whole data set). This final model is then evaluated using the testing data (the remaining 15% of the whole data set) that has been put aside from the beginning of the experiments. The final loss value is 0.2527 (note that the loss value is based on the predicted normalized house price).

6. Challenges

There are multiple challenges faced during the course of the experimentations. Those challenges are:

a. **Limited Computing Resource**
   There are too many possible configurations that can be tested. However, since one model training may take more than 10 minutes to finish, it is very hard to test a lot of configurations. Moreover, one configuration is tested using 3-fold cross validation. Thus, it takes much more time.

b. **Dependencies between Configuration Variables**
   Although the experiments are conducted by changing the value of one variable at a time, a variable may be actually dependent on another variable. Thus, in order to find the very best configuration, we may need to do an exhaustive grid search cross validation for many different values for each variable and this will take a lot of time to finish.

c. **Calculation Overflow**
   There is an issue when calculating the result of the Gaussian basis function. The issue occurs when the inverse of the covariance matrix is too large. As a result, the result of the multiplication becomes too large as well and it exceeds Python's double range. Thus, when the number of hidden neurons is too big compared to the number of input neurons, exceptions are thrown.

7. References

[1] Hinton, G., Srivastava, N., & Swersky, K. Lecture 6a Overview of Mini-batch Gradient Descent. [Online]. Available: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[2] CS287, University of California, Berkeley. CS287, Fall 2015 Problem Set Backpropagation, RMSProp, and Guided Policy Search. [Online]. Available: https://people.eecs.berkeley.edu/~pabbeel/cs287-fa15/assignments/EC3-GPS.pdf