WESTERN UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ECE 9603/9063b – Data Analytics Foundations

Assignment 2: Neural Networks

Student name: Jinliang Zhang

Student number: 250919668

Email address: jzhan964@uwo.ca

• *Description of the forecasting problem and data. This is the same as in Assignment 1. If there were issues with these sections, you should fix them. (2 points)*

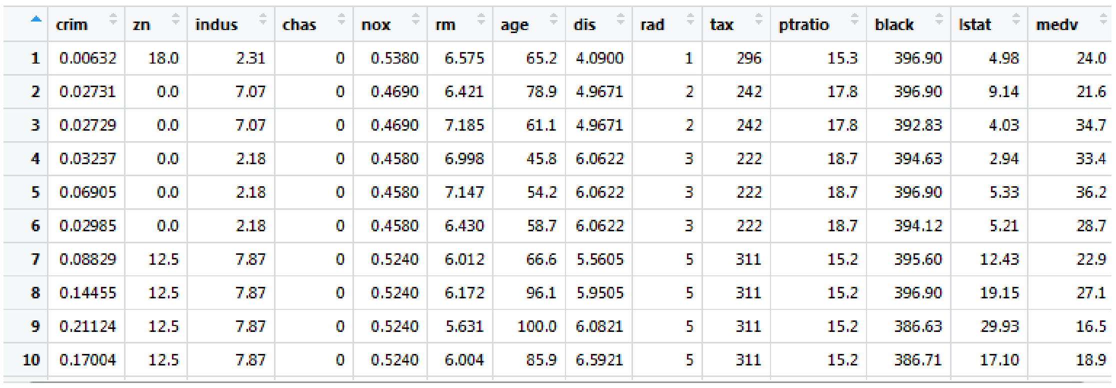
As I have used a limited number of features in Assignment 1, I change my dataset with more features and observations. I also implement 2 methods in the assignment 1 with this dataset. In the assignment 2, I choose "Housing Values in Suburbs of Boston" dataset, used as "Boston" in R datasets, to implement a regression forecasting problem. Through training from this dataset, the models are going to forecast the median value of owner-occupied homes (in \$1000s) when there are 13 input variables related to the details around the houses.

The "Boston" dataset has 506 rows and 14 columns which means it includes 506 observations on 14 variables with 13 input variables and 1 output variable. The list below is clearly shown the names and meanings of features in this dataset:

	No. of Col.	Name	Description of Variables
Input Variables	1	crim	per capita crime rate by town.
	2	zn	proportion of residential land zoned for lots over 25,000 sq.ft.
	3	indus	proportion of non-retail business acres per town.
	4	chas	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).
	5	nox	nitrogen oxides concentration (parts per 10 million).
	6	rm	average number of rooms per dwelling.
	7	age	proportion of owner-occupied units built prior to 1940.
	8	dis	weighted mean of distances to five Boston employment centers.
	9	rad	index of accessibility to radial highways.
	10	tax	full-value property-tax rate per \$10,000.
	11	ptratio	pupil-teacher ratio by town.
	12	black	1000(*Bk* - 0.63) ^2 where *Bk* is the proportion of blacks by town.
	13	lstat	lower status of the population (percent).
Output Variable	14	medv	median value of owner-occupied homes in \$1000s.

Table 1. Variables in dataset

Here are head 10 samples in "Boston" dataset:



	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
1	0.00632	18.0	2.31	0	0.5380	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
2	0.02731	0.0	7.07	0	0.4690	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
3	0.02729	0.0	7.07	0	0.4690	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
4	0.03237	0.0	2.18	0	0.4580	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
5	0.06905	0.0	2.18	0	0.4580	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2
6	0.02985	0.0	2.18	0	0.4580	6.430	58.7	6.0622	3	222	18.7	394.12	5.21	28.7
7	0.08829	12.5	7.87	0	0.5240	6.012	66.6	5.5605	5	311	15.2	395.60	12.43	22.9
8	0.14455	12.5	7.87	0	0.5240	6.172	96.1	5.9505	5	311	15.2	396.90	19.15	27.1
9	0.21124	12.5	7.87	0	0.5240	5.631	100.0	6.0821	5	311	15.2	386.63	29.93	16.5
10	0.17004	12.5	7.87	0	0.5240	6.004	85.9	6.5921	5	311	15.2	386.71	17.10	18.9

Figure 1. Samples of dataset

In the experiment, all data (506 observations) is used to implement regression forecasting.

• *Overview of the network architecture(s) you have used including important parameters that affect the network operation. Make sure you include parameters you will tune for your problem. (4 points)*

Among several deep learning architectures, such as feedforward neural network(FFNN), generative adversarial networks(GANs), convolutional neural networks(CNN) and recurrent neural networks(RNN), I used FFNN to complete this assignment. Next, I would analysis these architectures and the reasons I choose FFNN to solve this regression forecasting problem.

GANs:

GANs consist of two networks (generative network and discriminator network), one for generating contents and the other for identifying generated contents. Through training, generative network updates the parameters and generates more believable contents to deceive discriminator network. Otherwise, discriminator network is trained on the samples from the dataset and the contents from generative network in order to classify the output as real or synthetic. This creates a kind of competition: discriminator network is better at distinguishing between real data and synthetic data, and generative networks is also becoming better at generating contents which are harder to predict.

I am going to solve a regression forecasting problem in this assignment. For GANs, this architecture is commonly used to observe real and synthetic data at the same time and judge whether the data is true or not. The main application of this architecture is classification instead of regression. Therefore, GANs is not a suitable architecture style for my problem.

CNN:

CNN is a class of deep, feedforward neural network. CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers. This structure allows CNN can capture the two-dimensional structure of the input data (usually are images) and special relations of nearby points. It is good at building position and rotation invariant features from raw image data. CNN also performs very well in speech analysis.

As my problem is a regression forecasting problem in this assignment. The most common application of CNN is classification. Also, the common inputs of CNN are images and voice data as CNN is mainly used to process image data and speech analysis. Therefore, CNN is also not a suitable architecture style for my problem.

RNN:

RNN is a class of feedforward neural network. The connections between units form a directed graph along a sequence which allows it to exhibit dynamic temporal behavior for a time sequence. It includes a feedback loop that it uses to learn from sequences and model the time aspect of data. And the output at each time-step is based on not only the current input but also the input at all previous time steps. The features of RNN makes it perform good at modelling temporal behavior, such as time series, handwriting recognition and speech recognition.

In my chosen dataset, the input is 13 features of the details around the houses and the output is the median value of houses. They do not have any relationship in time series. As RNN is good at modelling temporal behavior, it is not a suitable architecture style for solving my regression forecasting problem in this assignment.

FFNN:

In the feedforward neural network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes and to the output nodes. There are no cycles or loops in the network. Normally, layers are fully connected to neighboring layers which means each neuron in a layer is connected to every neuron in another layer. The architecture I used in this assignment is multi-layer perceptron (MLP) which is a typical architecture of FFNN. It consists of multiple layers of computational units, usually interconnected in a feed-forward way. Each neuron in one layer has directed connections to the neurons of the subsequent layer. In general, it is trained by back propagation algorithm and it can establish the relationship between input data and output data.

In this assignment, it is a regression problem. Firstly, MLP is more relaxed for the input variables compared with the architectures above. It does not need the input variable to be images or time-series datasets so that it can perform well. Next, MLP is an application of supervised learning, it can compare the output values with the correct answer to compute the value of some predefined error-function. In this way, the model adjusts the weights of each connection in order to reduce the value of the error function. Through tuning the parameters in the MLP, there will be a good model that fits the dataset well, and it can be used to implement regression forecasting. Based on the above reasons, MLP is a suitable architecture style to solve my problem.

Some important parameters in MLP, such as the number of hidden layers, the number of neurons in every hidden layer, threshold of error function, the maximum steps for the training, the number of repetitions for the neural network's training, learning rate, error function, activation function and the model is for regression or classification, can affect the network operation. In this assignment, I built a MLP model in R language, and I tuned 5 kinds of parameters to optimize the algorithm, which are the number of hidden layers, the number of neurons in hidden layers, threshold of error function, the maximum steps for the training of the neural network and learning rate.

In neuralnet package, the arguments I tuned are presented as below:

Name	Description of Arguments
hidden	a vector of integers specifying the number of hidden neurons in each layer.
threshold	a numeric value specifying the threshold for the partial derivatives of the error function as stopping criteria.
stepmax	the maximum steps for the training of the neural network. Reaching this maximum leads to a stop of the neural network's training process.
learningrate	a numeric value specifying the learning rate used by traditional backpropagation. Used only for traditional backpropagation.

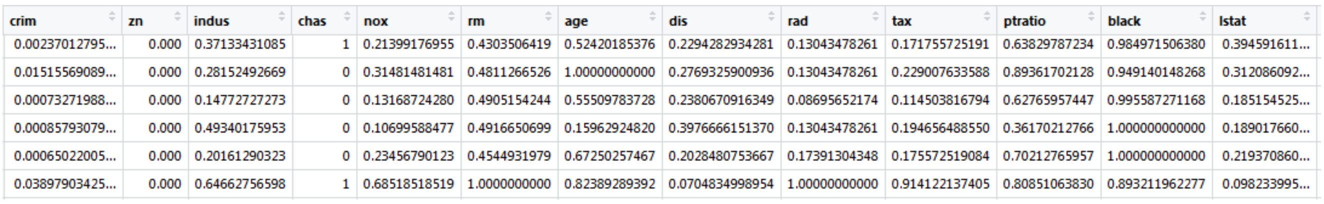
Table 2. Tuned arguments

• *Description of the process you have used including data pre-processing, feature generation, model training, and evaluation. (7 points)*

Data pre-processing: As the dataset is a matrix with 506 rows and 14 columns, and the observations are all values, FFNN can operate on it so that I do not need to vectorize the dataset. As 13 input features which are all related to output feature, none of them needs to be thrown away. Then, I split the dataset into two parts: 80% used to train the models, and 20% held back as a test set. Next, I normalize the dataset, so it is more consistent, and assign the scaled values to training and test set. For normalization, I used min-max scaling method to process the dataset. Features are scaled to a fixed range [0,1] through the algorithm below:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

And the samples of processed values are shown as below:



crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat
0.00237012795...	0.000	0.37133431085	1	0.21399176955	0.4303506419	0.52420185376	0.2294282934281	0.13043478261	0.171755725191	0.63829787234	0.984971506380	0.394591611...
0.01515569089...	0.000	0.28152492669	0	0.31481481481	0.4811266526	1.00000000000	0.2769325900936	0.13043478261	0.229007633588	0.89361702128	0.949140148268	0.312086092...
0.00073271988...	0.000	0.14772727273	0	0.13168724280	0.4905154244	0.55509783728	0.2380670916349	0.08695652174	0.114503816794	0.62765957447	0.995587271168	0.185154525...
0.00085793079...	0.000	0.49340175953	0	0.10699588477	0.4916650699	0.15962924820	0.3976666151370	0.13043478261	0.194656488550	0.36170212766	1.000000000000	0.189017660...
0.00065022005...	0.000	0.20161290323	0	0.23456790123	0.4544931979	0.67250257467	0.2028480753667	0.17391304348	0.175572519084	0.70212765957	1.000000000000	0.219370860...
0.03897903425...	0.000	0.64662756598	1	0.68518518519	1.0000000000	0.82389289392	0.0704834998954	1.00000000000	0.914122137405	0.80851063830	0.893211962277	0.098233995...

Figure 2. Samples of scaled dataset

Feature generation: In the "Boston" dataset, there are 14 variables related to housing values in suburbs of Boston. 13 variables of them are considered as independent variables which I used as input variables in this assignment. And the 14th variable is considered as dependent variable which I used as output variable. To get a better model to forecast, all features and 506 observations are used in this assignment.

Model training: To train the model, I built a MLP with parameters (hidden, stepmax, learningrate, threshold, error function and activation function). Then, I put the training set into this MLP. After training from this neural network, I put the input variables of test set into this MLP, the neural network outputs the predictions values which I can make a comparison with the original values of output variable in test set. As I tuned four parameters, there are 24 models are built to train and make predictions for test set. More details of tuning parameters and comparisons of the results of forecasting will be shown in next chapter.

Evaluation: To evaluate the algorithm with different parameters and compare with the algorithms I used in assignment 1 which are linear regression algorithm and support vector regression algorithm, root mean square error (rmse) is used as method to evaluate the accuracy of the algorithm. I first get the difference between the output values of the test set and predictions. After taking the average of their square sums and extracting a root of it, I get the rmse for one parameter condition of MLP. Then I keep modifying the parameters to find optimal model, the results of tuning are shown at next chapter.

• *Results. This should include accuracy measures (more than one) achieved through the training process, graphs demonstrating final accuracy as well as the accuracy through the tuning process. (7 points)*

I built 24 models during the tuning process. Here, I show some samples of tuning process of different parameters.

In the beginning of tuning, I initialize the parameters in the neural network with the initial values as below:

hidden=c(40,30,20,10,5), threshold = 0.01, stepmax=1000, learningrate = 0.01, err.fct = "sse", act.fct = "logistic"

Tune hidden layers:

"hidden=c(40,30,20,10,5)" means there are 5 hidden layers, through input to output, there are 40, 30, 20, 10, 5 neurons separately in every layer.

I set 4 different situations for hidden layers and 4 different values for neurons in the same hidden layer, and make other parameters be invariant.

Hidden layers	No. of layers	Root mean square error (rmse)
hidden=c(30,20,10,5)	4	3.081
hidden=c(40,30,20,10,5)	5	2.859
hidden=c(40,40,30,20,10,5)	6	2.715
hidden=c(40,40,40,30,20,10,5)	7	3.312

Table 3. rmse for different hidden layers

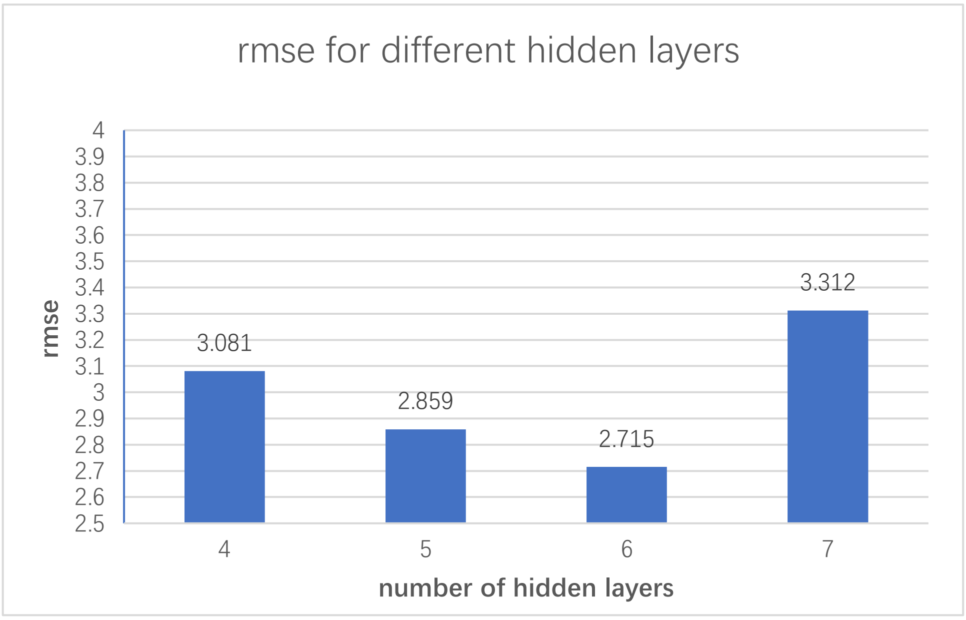


Figure 3. rmse for different hidden layers

Hidden layers	No. of neurons	Root mean square error (rmse)
hidden=c(40,40,30,20,10,5)	40	2.715
hidden=c(45,40,30,20,10,5)	45	2.703
hidden=c(50,40,30,20,10,5)	50	3.183
hidden=c(60,40,30,20,10,5)	60	3.335

Table 4. rmse for different neurons

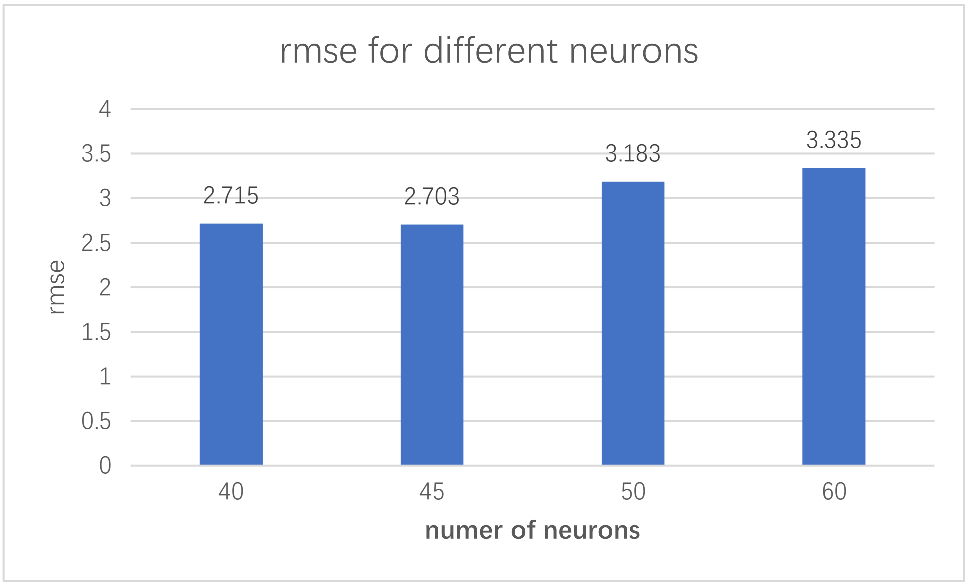


Figure 4. rmse for different neurons

Tune threshold:

Threshold is a numeric value specifying the threshold for the partial derivatives of the error function as stopping criteria. First, I set the hidden layer as "hidden=c(45,40,30,20,10,5)" and tuned 3 different values of threshold.

Values of threshold	Root mean square error (rmse)
threshold = 0.01	2.703
threshold = 0.05	2.908
threshold = 0.15	2.974

Table 5. rmse for different thresholds

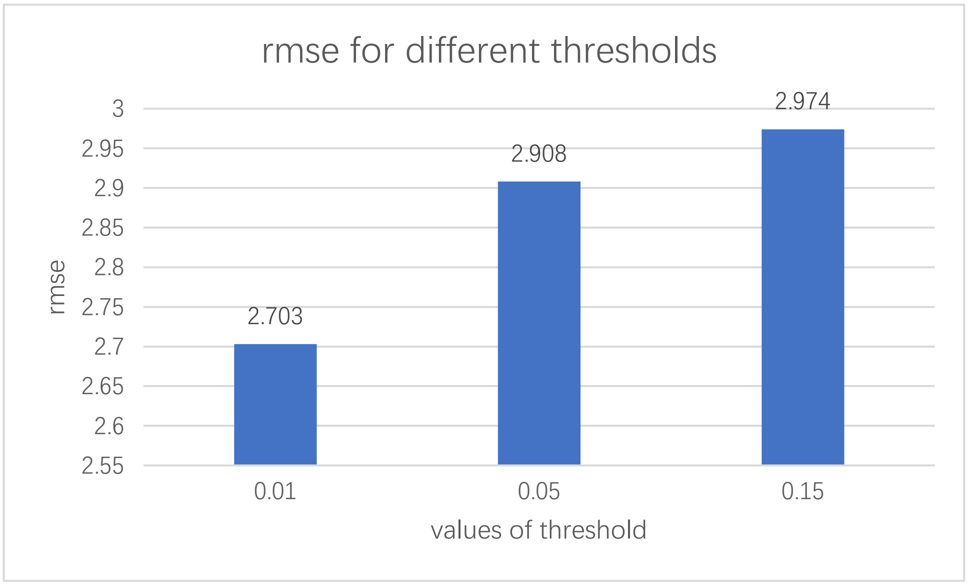


Figure 5. rmse for different thresholds

Tune learning rate

I set the hidden layer as “hidden=c(45,40,30,20,10,5)” and “threshold= 0.01”. Then I tuned 3 different values of learning rate.

Values of learning rate	Root mean square error (rmse)
learningrate= 0.01	2.703
learningrate= 0.05	2.703
learningrate= 0.1	2.703

Table 6. rmse for different learning rate

For this dataset, changes of learning rate in the range of [0.01,0.1] cause unobvious changes of rmse.

Tune the maximum of iterations

“stepmax” is used to set the maximum number of iterations. I set the hidden layer as “hidden=c(45,40,30,20,10,5)” , “threshold= 0.01” and “learningrate = 0.01”. Then, I tuned 3 different values of “stepmax”.

Values of stepmax	Root mean square error (rmse)
stepmax = 200	algorithm did not converge in 1 of 1 repetition(s) within the stepmax.
stepmax = 500	2.703
stepmax = 1000	2.703

Table 7. rmse for different stepmax

In this MLP, as I do not tune the training repetitions of it, the initial value of repetition is 1. When stepmax is not enough, it cannot to make the algorithm converge.

Tune the loss function and activation function:

In neuralnet package, there are two options for loss function, ‘sse’ and ‘ce’, which stand for the sum of squared errors and the cross-entropy can be used. As ‘ce’ is used for binary classification, I can only use the ‘sse’ for loss function.

Also, there are two options for activation function, ‘logistic’ and ’tanh’, which are possible for the logistic function and tangent hyperbolicus. As ‘tanh’ is not a suitable function for the input values so I just used the logistic function.

Results:

I show graphs which demonstrates final accuracy of 24 models below with consistent parameters “stepmax=1000, err.fct = "sse", act.fct = "logistic”.

	Parameters in every model	rmse
1	hidden=c(40,30,20,10,5), threshold = 0.01, learningrate= 0.01	2.859
2	hidden=c(40,30,20,10,5), threshold = 0.01, learningrate= 0.1	2.858
3	hidden=c(40,30,20,10,5), threshold = 0.05, learningrate= 0.01	2.942

4	hidden=c(40,30,20,10,5), threshold = 0.05, learningrate= 0.1	2.943
5	hidden=c(40,30,20,10,5), threshold = 0.15, learningrate= 0.01	2.953
6	hidden=c(40,30,20,10,5), threshold = 0.15, learningrate= 0.1	2.954
7	hidden=c(40,40,30,20,10,5), threshold = 0.01, learningrate= 0.01	2.715
8	hidden=c(40,40,30,20,10,5), threshold = 0.01, learningrate= 0.1	2.716
9	hidden=c(40,40,30,20,10,5), threshold = 0.05, learningrate= 0.01	3.003
10	hidden=c(40,40,30,20,10,5), threshold = 0.05, learningrate= 0.1	3.003
11	hidden=c(40,40,30,20,10,5), threshold = 0.15, learningrate= 0.01	3.289
12	hidden=c(40,40,30,20,10,5), threshold = 0.15, learningrate= 0.1	3.290
13	hidden=c(45,40,30,20,10,5), threshold = 0.01, learningrate= 0.01	2.703
14	hidden=c(45,40,30,20,10,5), threshold = 0.01, learningrate= 0.1	2.703
15	hidden=c(45,40,30,20,10,5), threshold = 0.05, learningrate= 0.01	2.908
16	hidden=c(45,40,30,20,10,5), threshold = 0.05, learningrate= 0.1	2.908
17	hidden=c(45,40,30,20,10,5), threshold = 0.15, learningrate= 0.01	2.974
18	hidden=c(45,40,30,20,10,5), threshold = 0.15, learningrate= 0.1	2.975
19	hidden=c(50,40,30,20,10,5), threshold = 0.01, learningrate= 0.01	3.183
20	hidden=c(50,40,30,20,10,5), threshold = 0.01, learningrate= 0.1	3.183
21	hidden=c(50,40,30,20,10,5), threshold = 0.05, learningrate= 0.01	3.603
22	hidden=c(50,40,30,20,10,5), threshold = 0.05, learningrate= 0.1	3.604
23	hidden=c(50,40,30,20,10,5), threshold = 0.15, learningrate= 0.01	3.553
24	hidden=c(50,40,30,20,10,5), threshold = 0.15, learningrate= 0.1	3.553

Table 8. rmse for different MLP models

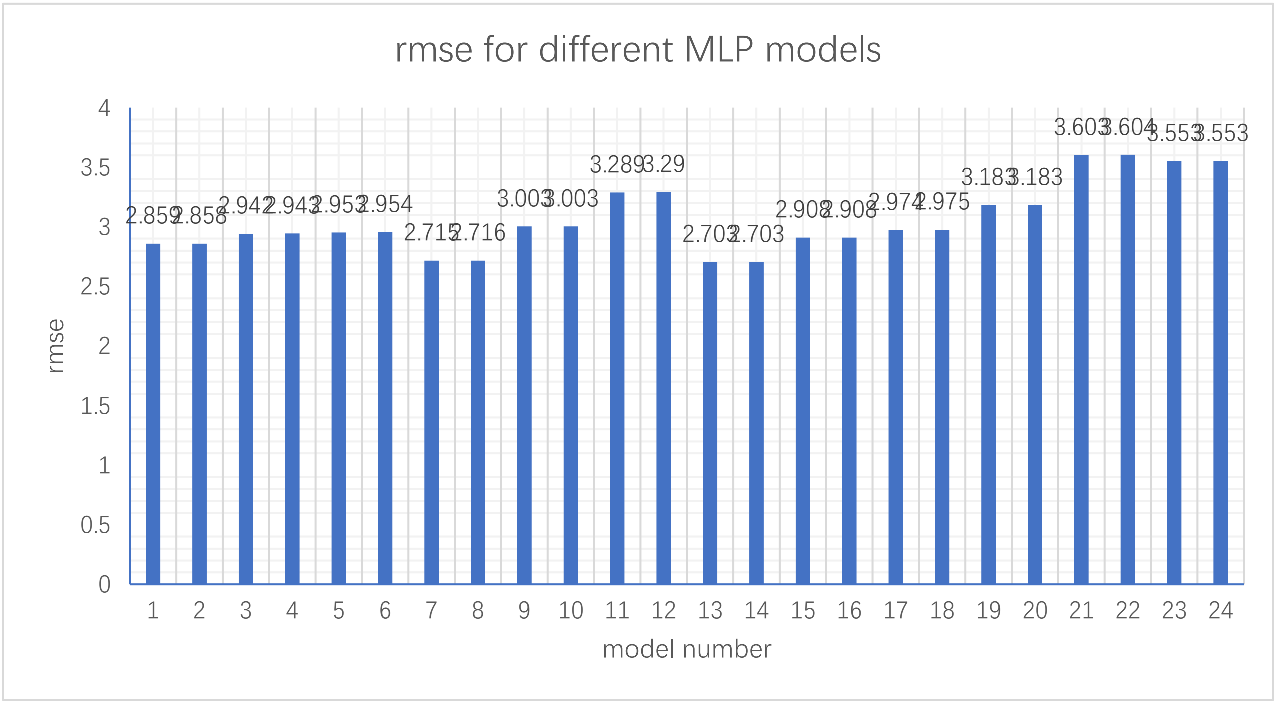


Figure 6. rmse for different MLP models

After comparing 24 MLP models, the best parameters setting is “hidden=c(45,40,30,20,10,5), threshold = 0.01, learningrate= 0.01, stepmax=1000, err.fct = "sse", act.fct = "logistic” with the lowest rmse 2.703.

Now use the chosen parameter to train the MLP model and compare its accuracy with accuracy achieved with other models in assignment 1.

From the result (Table 9), MLF is the most accurate model on the test set with the lowest rmse, followed by SVR with rmse of 4.268. The worst model among them is LR with the highest rmse of 4.835.

Model	rmse
Multi-layer perceptron (MLP)	2.703
Support vector regression (SVR)	4.268
Linear regression (LR)	4.835

Table 9. model comparsion

• *Code. Although there are no marks for the code itself, marks will be deducted if the code does not match the rest of the report.*

Programming language: R.

Integrated Development Environment: Rstudio.

```
# read data
set.seed(500)
require(MASS)
data<- Boston

apply(data,2,function(x) sum(is.na(x)))

# Split data into training and testing
sample <- sample.int(n=nrow(data), size = floor(0.8*nrow(data)))

#Standardzation
maxs <- apply(data,2,max)
mins <- apply(data,2,min)
scaled <- as.data.frame(scale(data, center = mins, scale = maxs - mins))
train <- scaled[sample,]
test <- scaled[-sample,]
test.r <- (test$medv)*(max(data$medv)-min(data$medv))
```

```
p <- (max(data$medv)-min(data$medv))

#Linear regression
modelLR <- lm(medv    ~ ., train)
modelLR
a <- test$medv
predictedLR <- predict(modelLR, test)
rmseLR <- sqrt(mean((test.r - predictedLR*p)^2))

#SVM
require (e1071)
modelSVR <- svm(medv ~ . , train, type= "eps-regression")
modelSVR

predictedSVR <- predict(modelSVR, test)
rmseSVR <- sqrt(mean((test.r - predictedSVR*p)^2))

#MLP(feed-forward neural network)
library(neuralnet)
n <- names(train)
f <- as.formula(paste("medv ~", paste(n[!n %in% "medv"], collapse = " + ")))
nn <- neuralnet(f,data=train,hidden=c(45,40,30,20,10,5), threshold = 0.01,
learningrate= 0.01, err.fct = "sse", act.fct = "logistic",stepmax=1000,linear.output= F)
#plot(nn)
pr.nn <- compute(nn,test[,1:13])
pr.nn_ <- pr.nn$net.result*p
rmseFFNN <- sqrt(mean((test.r - pr.nn_)^2))
```