Neural Networks

Irfan Kanat

September 27, 2018

```
library(psych) # Just for the sake of describe() function
library(neuralnet) # The workhorse of this activity
library(ggplot2) # To draw nicer plots
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following objects are masked from 'package:psych':
##
##     %+%, alpha
```

```
library(gridExtra) # To arrange multiple plots in a panel
set.seed(5096) # For reproducibility
```

Neural Networks

Neural networks (NN), like regression trees can be used for both continuous and categorical dependent variables with minor tweaks. Here we will discuss the neural networks over a continuous example. One nice thing about a neural net is that it fits a much broader range of data. Regressions generally assume a linear mechanism producing the results. With a neural net, we are freed of these kinds of assumptions. In fact, NN are called universal function approximators for this.

While this may sound exciting, the NN have a significant downside. Their structure is not very conducive to interpretation. Which is why they are sometimes referred to as a black box. We observe the inputs and outputs, but we can not really use the relations between nodes to explain how NN reached the outcome the way we do with regression coefficients. In that sense, NN are good for prediction, but not for explanation. So unless your data is produced by a non-linear mechanism, it can still be preferable to use a linear model as opposed to the computationally more demanding NN.

Here is roughly how it works. The Neural Networks are inspired by the workings of a brain. A series of simple nodes (neurons) are connected to each other. While each node handles a specific function, combined together a network of nodes can identify if something is round or not for example.

The simplest example of a neural network is a network with one layer and one node. This specific configuration is called a perceptron.

The perceptron takes an inputs (x) and a bias modifier. These inputs through an activation function (essentially each node does something to inputs. You can think of it as a simple transformation) if the activation function output meets a certain threshold, the neuron will "fire" or activate. If the neuron fires, the result of activation function is outputted.

Essentially perceptron has a series of inputs (x), weights(w), a bias (b), an activation function (g), and a threshold.

$g(w_1x_1 + w_2x_2 + \ldots + w_nx_n + b)$

I will use an example to explain what an activation function (g) is. A popular activation function is called rectified linear unit (reLU), essentially as long as the sum of weighted inputs and bias is below 0, the reLU will return 0, if the sums are greater than 0 it will return whatever the sum is.

In this example, the threshold is 0. If sum is greater than 0, the node activates, else returns 0.

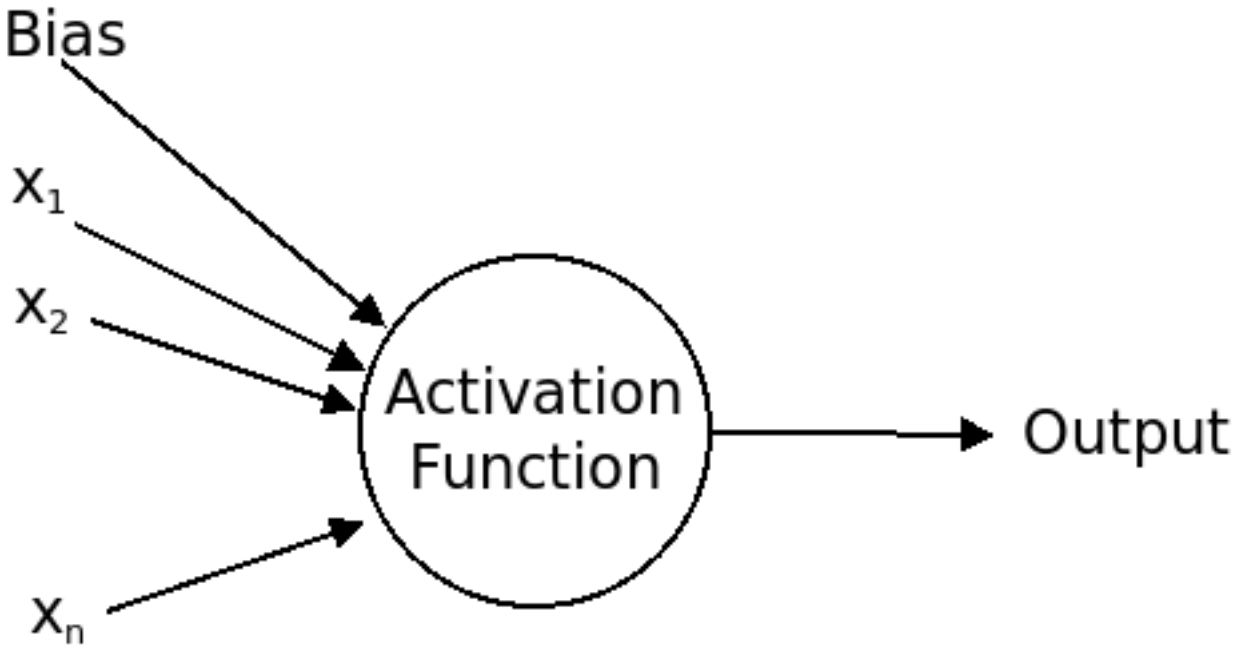


Figure 1: Perceptron

Remember the bias is added to inputs. You can think of it as the intercept in a regression. Essentially, it alters the behavior of the threshold. In reLU for example, a high bias may lead to easier activation.

While every node receives multiple inputs, yet not all inputs are treated equally. The neural network assigns and adjusts weights (w) to each source of input to maximize accuracy. You can see from above example, the weights can alter the effect of each input on the activation. They are in that sense similar to regression coefficients.

A perceptron is a single neuron, ideally we want a network of neurons.

Above is an example of a feed forward neural network. Each node in the network is connected to all the nodes in the next layer over. The nodes that are activated pass on their output to all the nodes in the next layer, hence creating a feed forward structure. We know the inputs and outputs, yet what happens in between is not our concern (remember the black box), which is why we call them the hidden layers. Given enough hidden layers a NN can estimate quite complex relations.

We have explained the role of an activation function in a perceptron (single neuron) above. Now imagine many different neurons with many different activation functions. That is what a NN is.

You know what feedforward stands for, here is another term for you: back propagation. Essentially the error in the output layer (difference between prediction and reality) is shared between nodes in the previous layer which in turn push a share of errors to the previous layer. Depending on these errors received, each node alters the biases, and weights.

Here is a little exercise for you. Google has devised the perfect tool to understand what a NN does. Check out the Neural Network Playground to understand how a neural network works.

NN is thick with terminology and we can surely loose ourselves in it. I prefer to keep things more practical. Let us move on to an example and you can learn the more advanced topic as you need them later on.

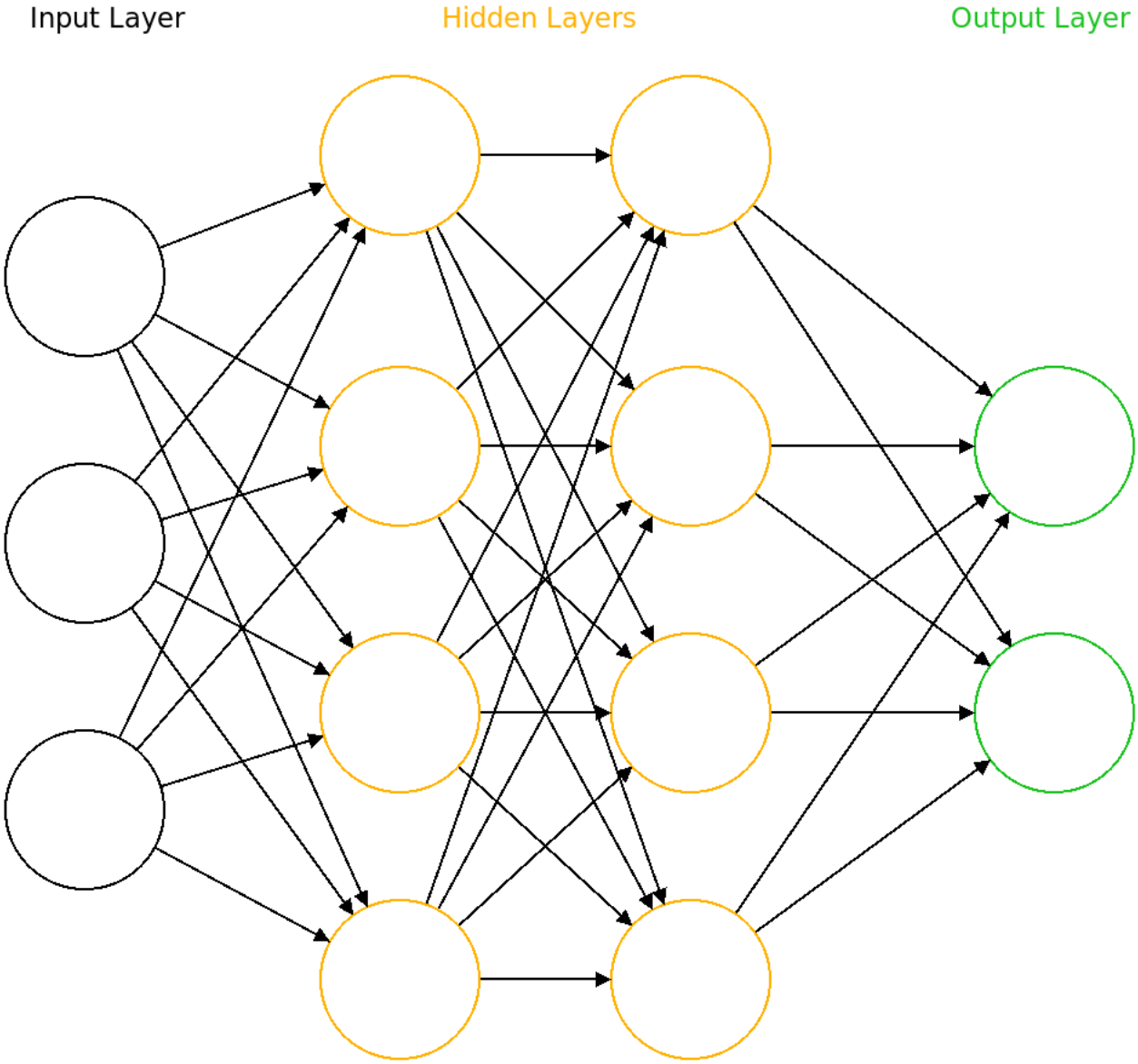


Figure 2: Neural Network

UCLA Housing Data

We will be using the UCLA housing data (same one we used for regression trees) for this exercise. The data can be found among Kaggle datasets online.

Import the dataset.

```
housing <- read.csv("data/housingdata.csv")
```

The data is at town level.

CRIM: Crime Rate

ZN: Proportion of residential land zoned

INDUS: Non-retail business acres per town

CHAS: River boundary

NOX: Nitric oxide concentration

RM: Average number of rooms

AGE: Proportion of owner occupied units built before 1940

DIS: Distance to five employment centres

RAD: Accessibility to highway

TAX: Property tax

PTRATIO: Pupil-Teacher Ratio

B: Proportion of blacks

LSTAT: proportion of lower status

MEDV: Median value of homes

Preprocessing

When the dataset is not standardized, NN can have difficulty converging. It is standard practice at this point to standardize inputs. Let us take care of that. You may remember this from Module 2, Activity 5 Transformations.

```
housingStd <- as.data.frame(scale(housing))
# See the effect of standardization
describe(housingStd)
```

```
##         vars   n mean sd median trimmed  mad   min  max range  skew
## CRIM       1 506    0  1  -0.39   -0.22 0.04 -0.42 9.92 10.34  5.19
## ZN         2 506    0  1  -0.49   -0.27 0.00 -0.49 3.80  4.29  2.21
## INDUS      3 506    0  1  -0.21   -0.03 1.37 -1.56 2.42  3.98  0.29
## CHAS       4 506    0  1  -0.27   -0.27 0.00 -0.27 3.66  3.94  3.39
## NOX        5 506    0  1  -0.14   -0.08 1.12 -1.46 2.73  4.19  0.72
## RM         6 506    0  1  -0.11   -0.05 0.73 -3.88 3.55  7.43  0.40
## AGE        7 506    0  1   0.32    0.09 1.03 -2.33 1.12  3.45 -0.60
## DIS        8 506    0  1  -0.28   -0.12 0.91 -1.27 3.96  5.22  1.01
## RAD        9 506    0  1  -0.52   -0.09 0.34 -0.98 1.66  2.64  1.00
## TAX       10 506    0  1  -0.46   -0.05 0.64 -1.31 1.80  3.11  0.67
## PTRATIO   11 506    0  1   0.27    0.10 0.79 -2.70 1.64  4.34 -0.80
## B         12 506    0  1   0.38    0.29 0.09 -3.90 0.44  4.34 -2.87
```

```
## LSTAT     13 506    0  1  -0.18   -0.11 1.00 -1.53 3.55  5.07  0.90
## MEDV      14 506    0  1  -0.14   -0.11 0.64 -1.91 2.99  4.89  1.10
##         kurtosis   se
## CRIM       36.60 0.04
## ZN          3.95 0.04
## INDUS      -1.24 0.04
## CHAS        9.48 0.04
## NOX        -0.09 0.04
## RM          1.84 0.04
## AGE        -0.98 0.04
## DIS         0.46 0.04
## RAD        -0.88 0.04
## TAX        -1.15 0.04
## PTRATIO    -0.30 0.04
## B           7.10 0.04
## LSTAT       0.46 0.04
## MEDV        1.45 0.04
```

Analysis

Our whole goal is prediction. We do not care about theory at this point. Hence we will fit the model with all the variables.

```
housingStd_nn_0 <- neuralnet(MEDV ~ CRIM + ZN + INDUS + CHAS + NOX + RM + AGE + DIS + RAD + TAX + PTRAT
```

The bare minimum necessary for fitting a basic neural network is just four parameters.

1. MEDV ~ CRIM + ZN + INDUS + CHAS + NOX + RM + AGE + DIS + RAD + TAX + PTRATIO + B + LSTAT: Is the formula, inputs and output.
2. data = housingStd: where the data comes from.
3. hidden: Number of hidden neurons in each layer. hidden = 3 is a single layer with three nodes, hidden = c(4,4) is two layers with four neurons each.
4. linear.output = T: Specifying a continuous output.

Here is a simple exercise for you: skim over the manual for neuralnet() function and try to identify the concepts you learned earlier.

```
plot(housingStd_nn_0)
```

That looks like quite a mess. Good thing we won't really do anything with that plot. Unlike linear regression where the coefficients are interpreted, in NN the weights or biases are not interpreted. Due to its predictive nature, all we care about in a NN is predictive accuracy.

Unless we cover training, testing and cross validation your understanding of predictive models especially on the machine learning side will be incomplete. I am saving this topic for Module 5, Automated Training, where we will cover it in great detail. Some of the methods we demonstrate below may look inconvenient, that is because performance of a predictive model is rarely evaluated on just one dataset. Learn what is shown below and we will build on top of these concepts in Module 5.

One measure of accuracy is the error (difference between predicted and observed values) of the model. Ideally we want error to be low.

```
housingStd_nn_0$result.matrix['error',]
```

```
## [1] 12.81585803
```

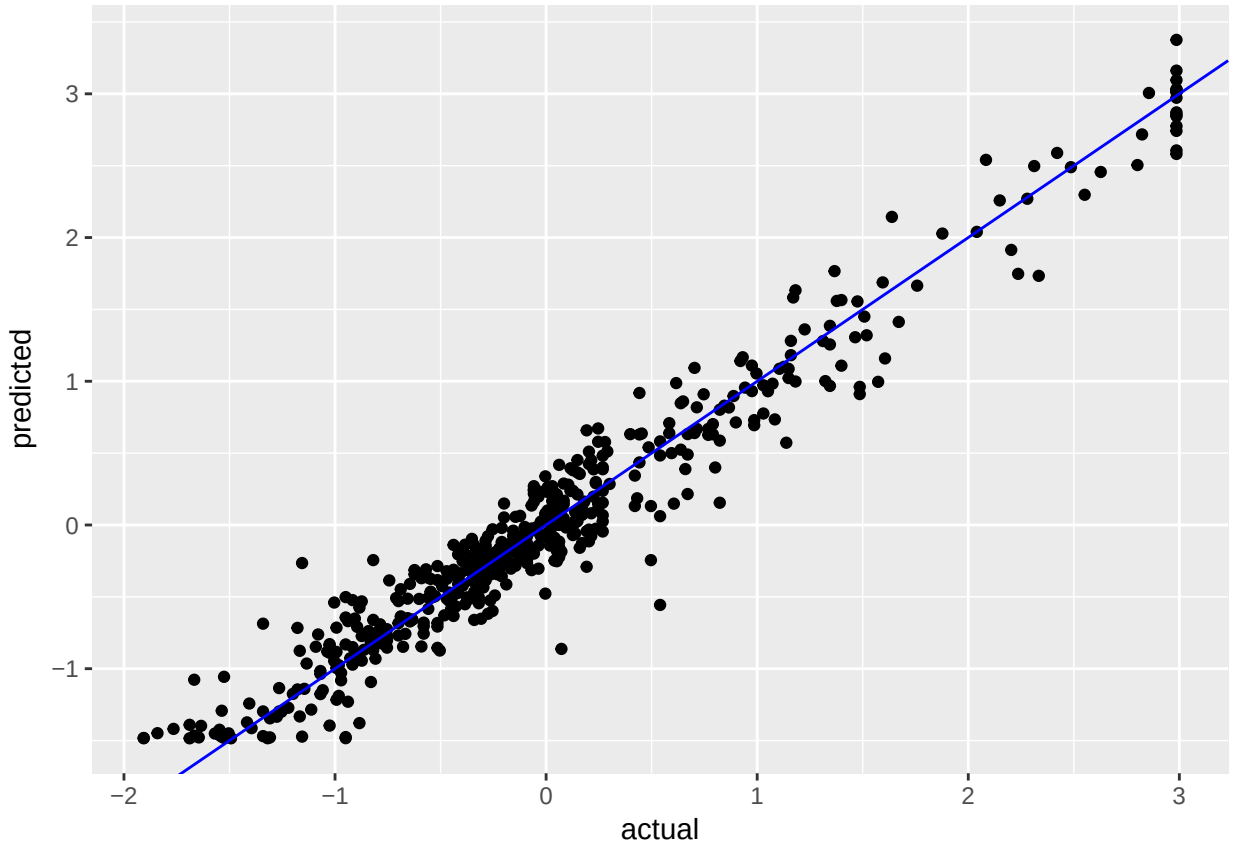
You may have already noticed but the fitted models are much like any other data structure in R. We can access parts of it like calling a column in a data.frame. Above what we are doing is, extracting the error of

the model.

Here is a simple exercise for you, fit a neural network with different hidden layer structure and compare the error to what we found above. Comment on which model to use.

Another thing we may want to do is check out predicted values and compare them to actual observations.

```
# Creating a new dataframe
# and storing predicted values from the NN and actuals from data
housingPredictedActual <- data.frame(predicted = housingStd_nn_0$net.result, actual = housingStd$MEDV)
colnames(housingPredictedActual) <- c('predicted', 'actual')
# Plot the predicted against actual
qplot(actual, predicted, data = housingPredictedActual, geom = 'point') +
  geom_abline(slope = 1, col = 'blue') # add the diagonal line
```

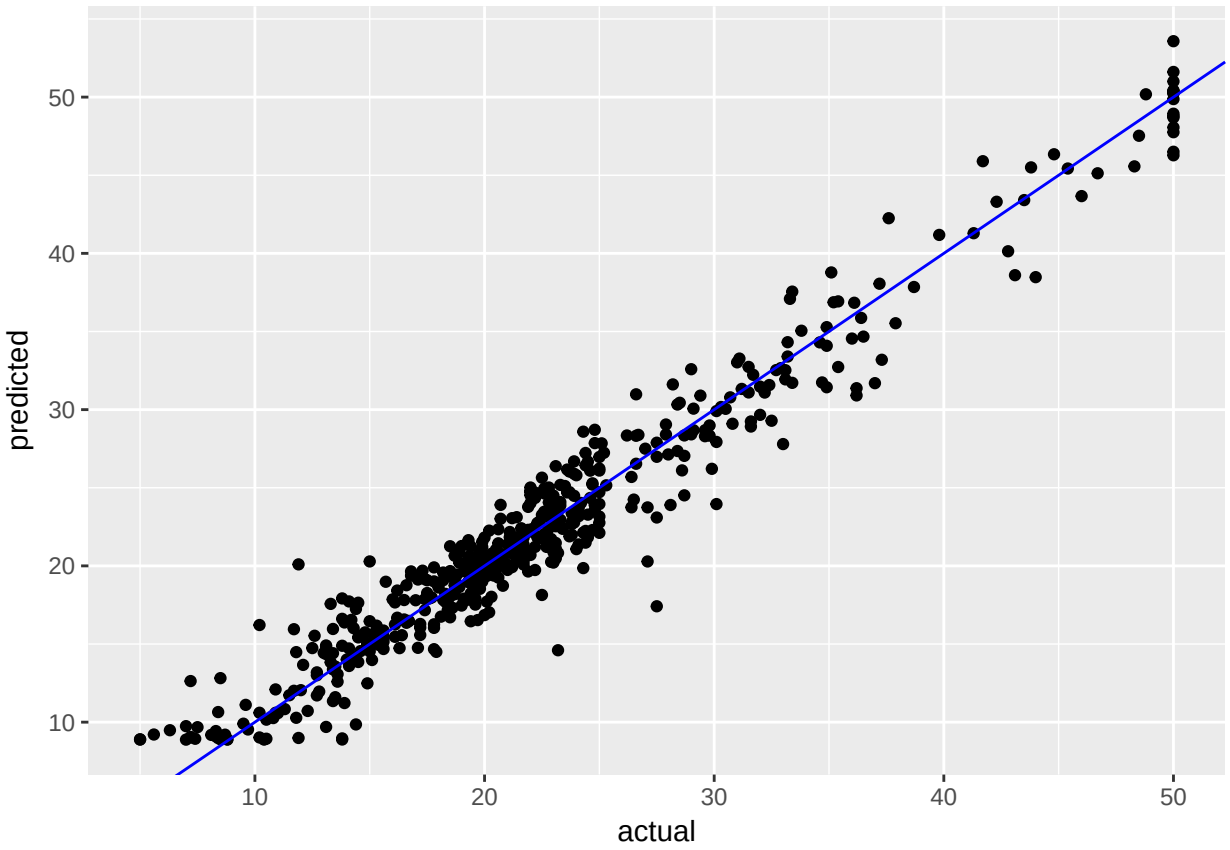


I added a line at 45 degrees in blue to the plot. If the model was perfectly accurate, all points would line up on that line.

Remember also that these are standardized results. They are not in the same scale as the original dependent variable. If you wanted to convert them back you would have to rescale it to the same range as MEDV.

```
# Reversing the standardization (multiply by sd and add mean)
housingPredictedActual2 <- (housingPredictedActual * sd(housing$MEDV)) + mean(housing$MEDV)

qplot(actual, predicted, data = housingPredictedActual2, geom = 'point') +
  geom_abline(slope = 1, col = 'blue') # add the diagonal line
```



Here is a simple exercise for you. Plot the predicted vs actual for the neural net you fit earlier. Compare against this model. Comment on which model is a better fit.

Predictions with NN

The goal in any NN model is making predictions on new data. Unlike other models we have seen thus far, neural net package's neural networks don't support predict function. Instead we use compute.

```
# I am creating a new dataset to predict.
# This dataset will have two rows full of information
newHousingStdData <- data.frame(CRIM = c(-.4, .4), ZN = c(.05, -.05), INDUS = c(-.5, .5), CHAS = c(-.27
compute(housingStd_nn_0, newHousingStdData)$net.result
```

```
##               [,1]
## [1,] -0.3855904724
## [2,]  1.1214644442
```

Final Note

This activity was designed to serve as a primer to Neural Networks. While this much is sufficient to start with, there is so much more to learn about neural networks. I encourage students to research on other types of activation functions, and network structures to achieve a more complete picture.

Solutions to Exercises

1 - Here is a little exercise for you. Google has devised the perfect tool to understand what a NN does. Check out the Neural Network Playground to understand how a neural network works.

I trust you all went to NN playground. We are all outstanding citizens here after all.

2 - Here is a simple exercise for you: skim over the manual for neuralnet() function and try to identify the concepts you learned earlier.

Ditto this one also.

3 - Here is a simple exercise for you, fit a neural network with different hidden layer structure and compare the error to what we found above. Comment on which model to use.

```
# One layer 3 nodes
housingStd_nn_0b <- neuralnet(MEDV ~ CRIM + ZN + INDUS + CHAS + NOX + RM + AGE + DIS + RAD + TAX + PTRA
housingStd_nn_0b$result.matrix['error',]
```

```
## [1] 21.24324531
```

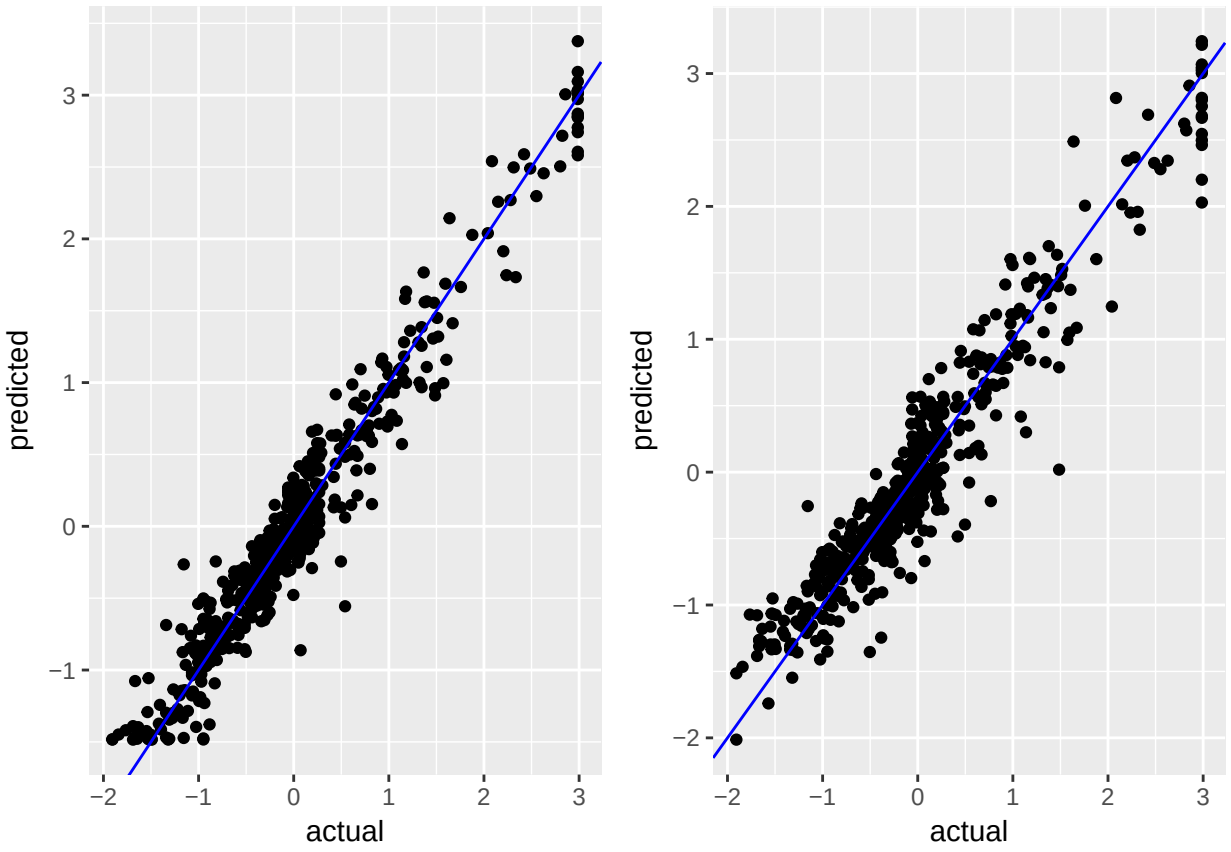
The error with the one layer, three node model is greater than the error of model with two layers with 4, and 3 nodes. In this case we would prefer the more complex neural network.

4 - Here is a simple exercise for you. Plot the predicted vs actual for the neural net you fit earlier. Compare against this model. Comment on which model is a better fit.

```
# Create a new dataset to plot
housingPredictedActual3 <- data.frame(predicted = housingStd_nn_0b$net.result, actual = housingStd$MEDV
colnames(housingPredictedActual3) <- c('predicted', 'actual')

# Store plot in an object
housingNN_Plot1 <- qplot(actual, predicted, data = housingPredictedActual, geom = 'point') +
  geom_abline(slope = 1, col = 'blue') # add the diagonal line
# Store the plot from the other model
housingNN_Plot2 <- qplot(actual, predicted, data = housingPredictedActual3, geom = 'point') +
  geom_abline(slope = 1, col = 'blue') # add the diagonal line

# Put two plots side by side
grid.arrange(housingNN_Plot1, housingNN_Plot2, nrow = 1)
```



As we expected from the error figures, the points are scattered closer in the first model (2 layers with 4 and 3 nodes) in comparison to the second model (1 layer with 3 nodes).