Lecture 10: Neural Networks

Suzer-Gurtekin

March 2025

Overview

Overview

Today, we will start with a broad **overview and illustration**. What is a neural network?

Then, we will **define concepts** needed for understanding neural networks. I will reference the illustration to aid in understanding these definitions.

We will go through the steps of **setting up a neural network**. What decisions need to be made?

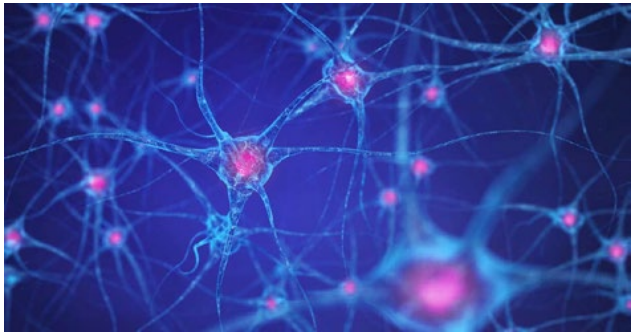
Walk through an **example** following the steps from the “set-up” description.

Overview

Neural Networks were initially meant to model the functioning of a human brain.

However, brains don't actually work in the way the model implies.

More correct to call these "artificial neural networks."



Overview

Neural networks start from input data.

Linear combinations of the input data are created.

These combinations are then used to predict an outcome.

Labeled outcomes are needed in order to learn to predict the outcome.

Understanding Hyperparameters

Definition:
A hyperparameter is a configuration that is external to the model and whose value cannot be estimated from the data.

Contrast with Model Parameters:
Model parameters, such as weights in a neural network, are learned during training.

Hyperparameters must be set before training begins.

Hyperparameters control the behavior of the training process.

They have a significant impact on the performance and effectiveness of the model.

Understanding Hyperparameters

K-means Clustering:

Number of clusters (k): This is the most important hyperparameter for K-means, as it determines how many groups or clusters the data should be divided into.

Initialization method: The method used to initially choose the cluster centers, such as random initialization or the k-means algorithm.

Maximum number of iterations: The maximum number of times the algorithm will run the update process before stopping.

Understanding Hyperparameters

Neural Networks:

Learning rate: Controls the size of the steps taken during the optimization process.

Number of layers and units per layer: Defines the architecture of the network, including how deep (number of layers) and how wide (number of units per layer) it is.

Batch size: The number of samples to use in one iteration for updating the model weights.

Activation function: The function applied to the output of each layer, such as ReLU, sigmoid, or tanh.

Dropout rate: The fraction of input units to drop in order to prevent overfitting.

Understanding Hyperparameters

Logistic Regression:

Regularization strength (C): Inverse of regularization strength; smaller values specify stronger regularization.

Penalty type: The norm used for penalization, such as L1 or L2 regularization.

Solver: The algorithm used for optimization, such as 'liblinear', 'saga', or 'lbfgs'.

Regularization

A technique to prevent overfitting in machine learning and statistical modeling.
Ensures the model learns underlying patterns rather than noise.

Adds a penalty on the complexity of the model.
Encourages generalization to new, unseen data.

Regularization

L1 Regularization (Lasso)

- Adds a penalty equal to the absolute value of the magnitude of coefficients
- Can lead to sparse models by reducing some weights to zero.
- Effective for feature selection in high-dimensional datasets.

L2 Regularization (Ridge)

- Adds a penalty equal to the square of the magnitude of coefficients
- Shrinks coefficients, retaining all features without eliminating them.
- Helps reduce model complexity and tackle multicollinearity.

Elastic Net

- Combines L1 and L2, balancing with a mixing parameter
- Leverages benefits of both L1 and L2 methods.
- Useful when many features are correlated.

Regularization

Trade-off Management

Regularization controls the balance between fitting well and keeping the model simple.

Hyperparameter Tuning

The strength of regularization can be adjusted.

Tuning finds the optimal balance for a given problem.

Illustration

Before turning to a more complex example, let's look at the illustration presented earlier. In the illustration, our data are:

Table: Variables

Variable	Description	Values
`Free`	I am free right now? i.e. not busy?	1=Free,0=Busy
`Interest`	Is this topic interesting to me?	1=Yes, 0=No
`Incentive`	Incentive in dollars	$0, $5, $10, $15

We also need labels – a variable with participation decision yes/no (`Participate`)

Illustration – Step 1- Create the simulation data

```
library(nnet)
library(devtools)
library(caret)
library(neuralnet)
library(NeuralNetTools)
library(reshape)
#import the function from
Githubsource_url('https://gist.githubusercontent.com/Peque/41a9e20d6687f2f3108d/raw/85e14f3a292e
126f1454864427e3a189c2fe33f3/nnet_plot_update.r')

set.seed(4322)
Free<-rbinom(1000,1,0.3)
Interest<-rnorm(1000,mean=3,sd=1)
Incentive<-sample(c(0,5,10,15),1000,replace=TRUE)
x<-data.frame(cbind(Free,Interest,Incentive))

#Here, I want to generate a participation decision
#That is a function of the inputs
pre.participate<-as.data.frame(Free*3+Interest/2+Incentive/5)
pre.pct<-pre.participate/max(pre.participate)
Participate<-as.numeric(runif(100,min=0,max=1)>pre.pct)
df<-data.frame(cbind(x,Participate))
```

Illustration

Step 2- Examine simulated data
- Use empirical log plots

Step 3- Fit a logistic model
- Interpret the coefficients
- Evaluate the goodness of fit

Step 4- Partition the data into training and test sets (small data here)

Step 5- Fit a feed-forward NN on the training dataset
- Try refitting model with 100 attempts
- Report whether you have found the true minimum
- Examine the estimate weights
- Plot the fitted network
- Summarize the fitted network
- Compute R^2 as in Faraway example
- Compute variable importance metrics (using NeuralNetTools)

Step 6- Predict the test data

Step 7 - Now using the neuralnet function, and the rep function to search across starting weights

Step 8 – Plot accuracy vs. weight decay
- Use cross-validation to determine the weight decay parameter. We are selecting a range of possible values: 0 to 2 by 0.05.

Illustration

Now, we learn the weights from some labeled data.

Here, using the `nnet` function

```
#Using the nnet function as shown in Faraway
bestrss<-10000
for (i in 1:100){
mod1<-nnet(df.std.train[,c(1:3)],df.std.train[,4],
           size=1,entropy=TRUE)
cat(mod1$value,"\n")
if (mod1$value<bestrss){
bestmod1<-mod1
bestrss<-mod1$value}
}
summary(bestmod1)
```

Illustration

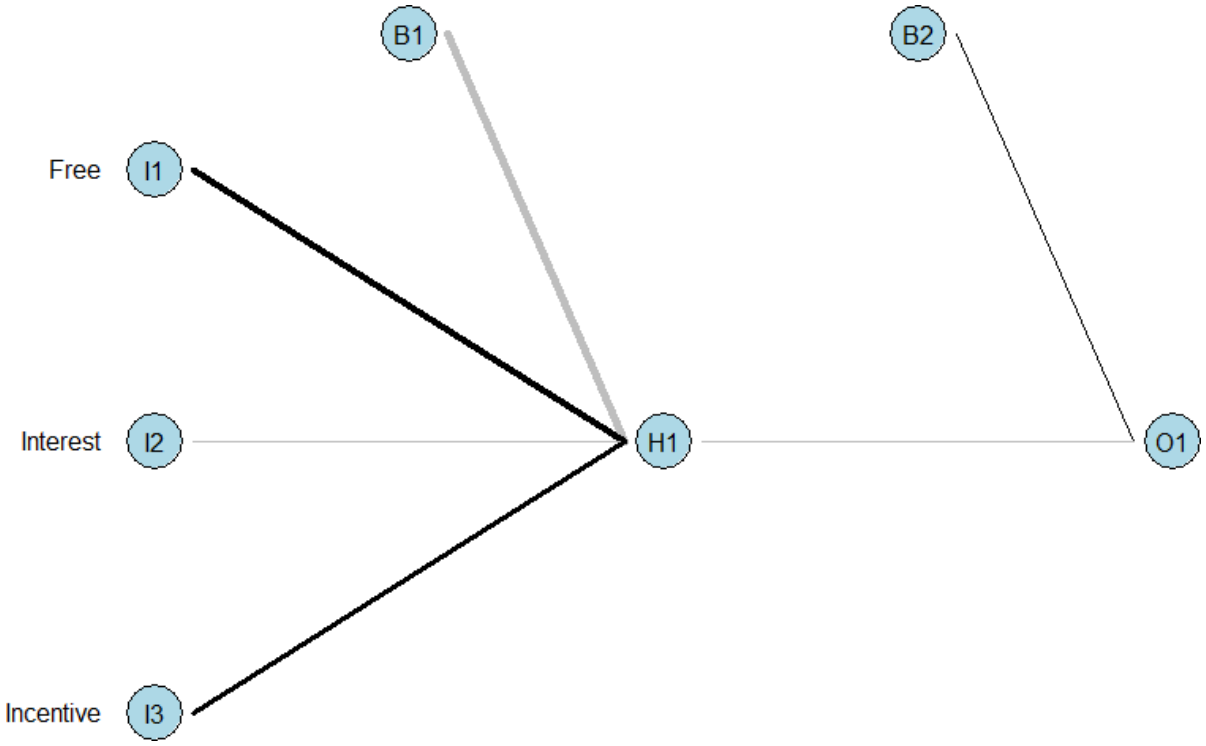
An activation function is used to convert the weighted sum into an output – either “yes” or “no” to the participation decision.

Here, we see the biases and weights for each of the layers:

```
> summary(bestmod1)
a 3-1-1 network with 6 weights
options were - linear output units
b->h1 i1->h1 i2->h1 i3->h1
0.84   0.13   0.07   0.20
b->o h1->o
4.34 -5.51
```

Illustration

Or, presenting the results graphically:



Illustration

Applying those results to the test data give the following accuracy:

```
> table(df.std.test$Participate,bestmod1.test)
bestmod1.test
    0  1
0   3  7
1   0 10
```

Illustration

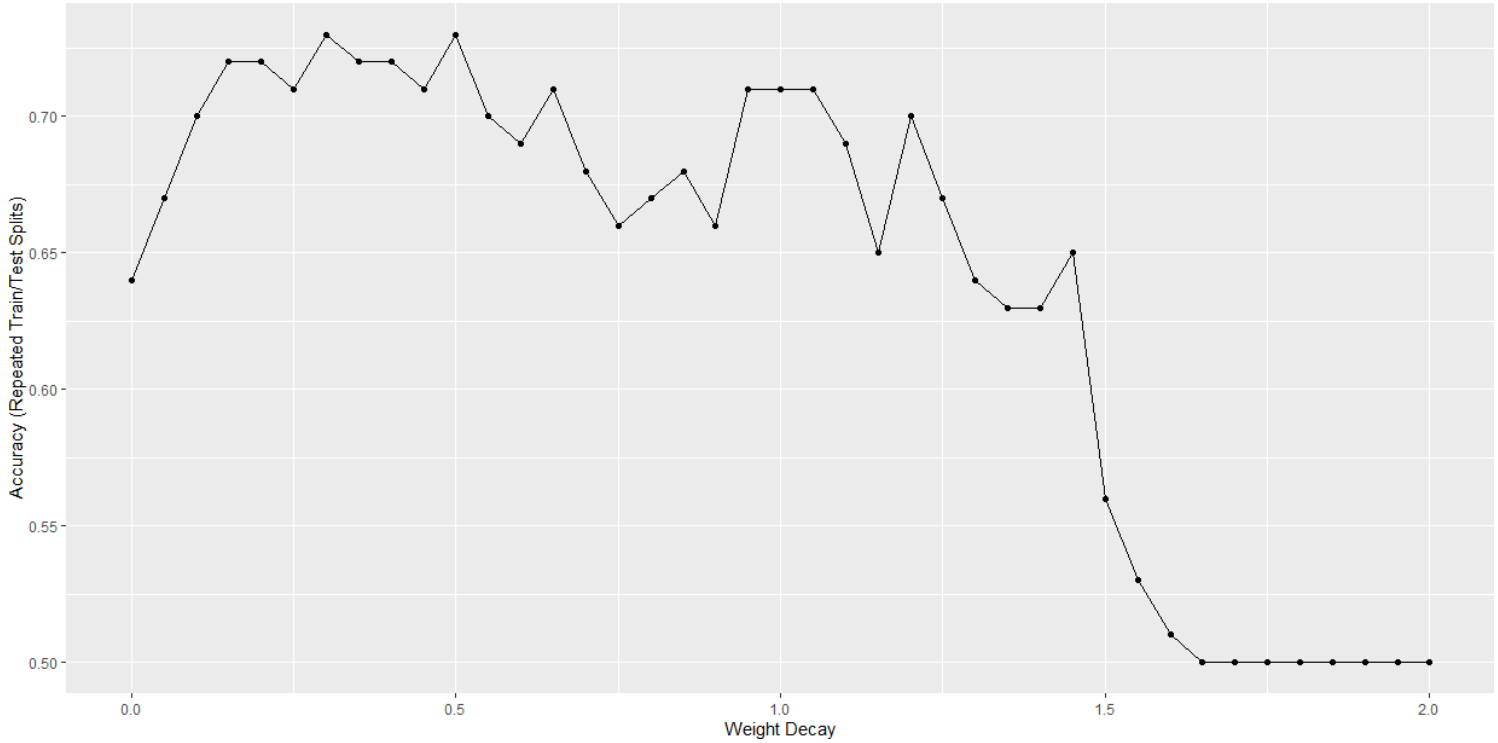
Recall that overfitting may occur.

We will use cross-validation to determine the weight decay parameter.

We are selecting a range of possible values: 0 to 2 by 0.05.

Illustration

Resulting accuracy:



Illustration

Now using the cross-validated results to make a prediction for the test data:

```
> table(test.Participate,df.std.test[,4])
test.Participate 0 1
               0 9 1
               1 1 9
```

We could also use cross-validation to determine the number of nodes and layers.

Setting Up a Neural Network

Now we have the concepts defined. Let's look at the **process** of setting up a neural network.

Feature Engineering I

First, examine the data.

- Missing data?
- Outliers?
- Near-zero variance?
- Variable type?

Feature Engineering II

We can follow steps similar to those we have seen before for feature engineering.

We want the input to the nodes to be suitable for the activation function.

Best to standardize the data.

- As we have seen before (LASSO and K-means), leaving the data on different scales essentially places differential importance on each.
- Unstandardized data make it more difficult to select starting weights and opens the door to getting stuck in local optima.

Holdout Sample

Train-test split of the data.

Save a proportion of the data for testing the final model.

Random subset, possibly stratified by the outcome.

Hyperparameters

Define the hyperparameters and functions of the network.

- Layers? Generally, one is sufficient
- Hidden units? Possibly worth exploring multiple hidden units
- Activation function (see above)
- Cost function (see above).

Number of Layers

Generally, a single hidden layer is sufficient.

Possible to try multiple layers.

Adding layers increases computation time.

In complex problems, more hidden layers may improve accuracy. Treat this as another hyperparameter.

Number of Nodes

Some general guidance on the number of nodes.

- Minimum: the number of output nodes.
- Maximum: the number of input nodes, i.e. features.
- Rule of thumb: Mean of minimum and maximum.
- Possible to try range of options.

Initial Weights

Choose initial weights. May want to do this multiple times.

- Using loop for `nnet`.
- Using `rep=` for `neuralnet`.

Search for solution that minimizes loss/error function.

Model evaluation

Our focus is on classification. So, look at accuracy of predictions.

- Compare predicted and observed values, `confusionMatrix` function from `caret` package.
- Use a train-test approach. Test the accuracy of the predicted classes/probabilities using the test set, and `trainControl` function from `caret` package.
- Check variable importance.
- Check results against another classification method, e.g. logistic regression.

Example

Now, having seen the process, let's look in detail at an **example** application.

This problem will ask that we classify the price of a mobile phone based on its features.

Example: Background

The example comes from Kaggle.com. Here is the description:

> "Bob has started his own mobile company. He wants to give tough fight to big companies like Apple,Samsung etc.
>
> He does not know how to estimate price of mobiles his company creates. In this competitive mobile phone market you cannot simply assume things. To solve this problem he collects sales data of mobile phones of various companies.
>
> Bob wants to find out some relation between features of a mobile phone(eg:- RAM,Internal Memory etc) and its selling price. But he is not so good at Machine Learning. So he needs your help to solve this problem.
>
> In this problem you do not have to predict actual price but a price range indicating how high the price is."
>
> — Kaggle.com

Our expected output is predicted price in four categories ranging from "low cost" to "very high cost".

Example: Features

Variable	Description
battery_power	Total energy a battery can store in one time measured
blue	Has bluetooth or not
clock_speed	speed at which microprocessor executes instructions
dual_sim	Has dual sim support or not
fc	Front Camera mega pixels
four_g	Has 4G or not
int_memory	Internal Memory in Gigabytes
m_dep	Mobile Depth in cm
mobile_wt	Weight of mobile phone
n_cores	Number of cores of processor

Example: Features

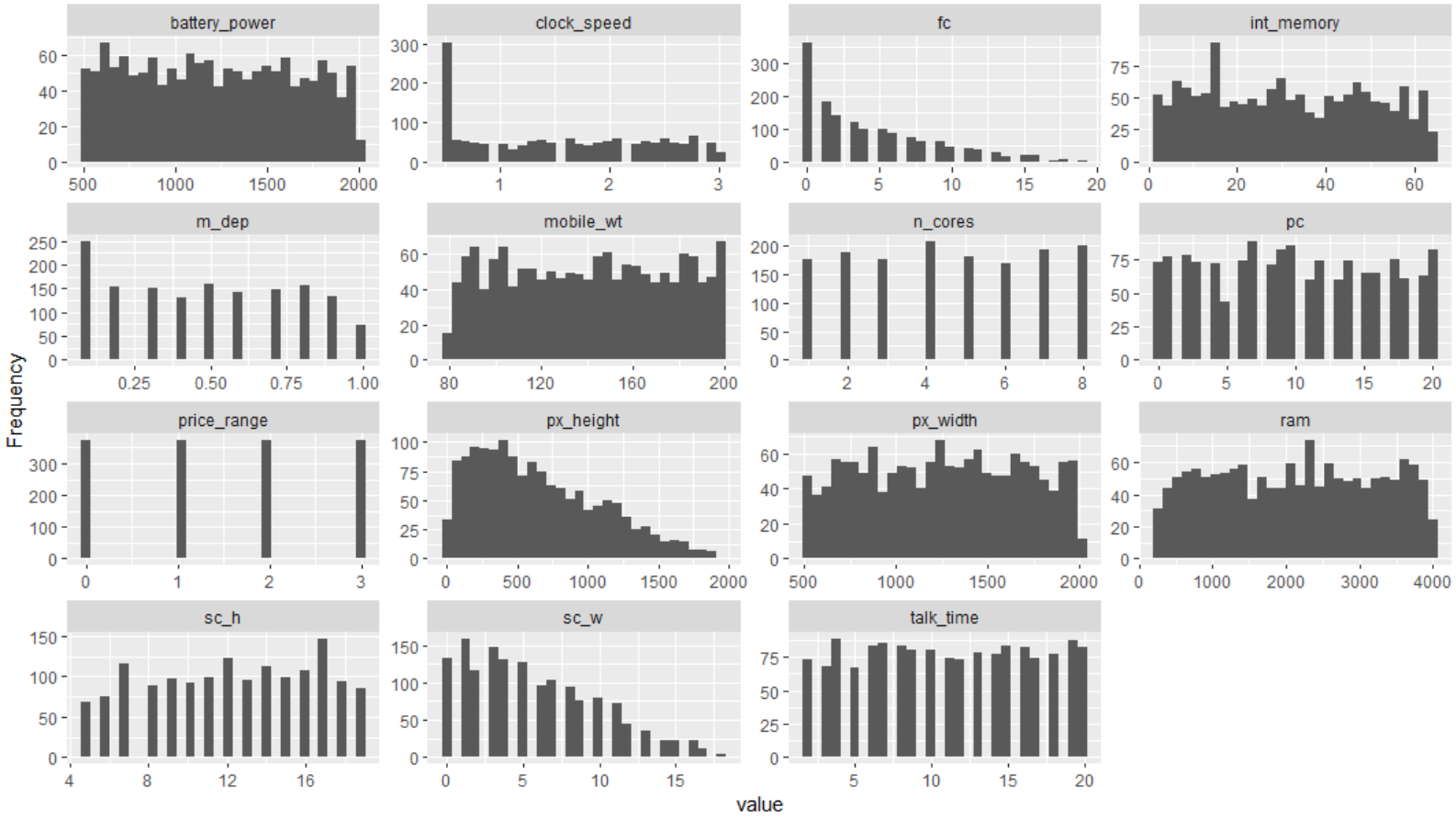
Variable	Description
pc	Primary Camera mega pixels
price_range	This is the **target** variable with value of 0(low cost) to 3(very high cost)
px_height	Pixel Resolution Height
px_width	Pixel Resolution Width
ram	Random Access Memory in Mega Bytes
sc_h	Screen Height of mobile in cm
sc_w	Screen Width of mobile in cm
talk_time	longest time that a single battery charge will last when y
three_g	Has 3G or not
touch_screen	Has touch screen or not
wifi	Has wifi or not

In-class Exercise

1- Using the DataExplorer package. Take a comprehensive review of the data.
2- Feature Engineering:
    Any missing data?
    Coding of categorical variables.
        For neuralnet package, need indicator variable for each category.
    Standardization. Important!
3- Build Neural Network
    Nodes: Set at 10. Intuition. Something better?
    Activation Function: Logistic
    Weights: Tried 10 different starting weights.
4- Evaluate the neural network on the test set:
    Compare predicted and observed values, confusionMatrix function from caret package.
    Use a train-test approach. Test the accuracy of the predicted classes/probabilities using the test set, and trainControl function from caret package.
    Variable importance
5- Run an ordered logistic regression model to see how well *ram* predicts by itself

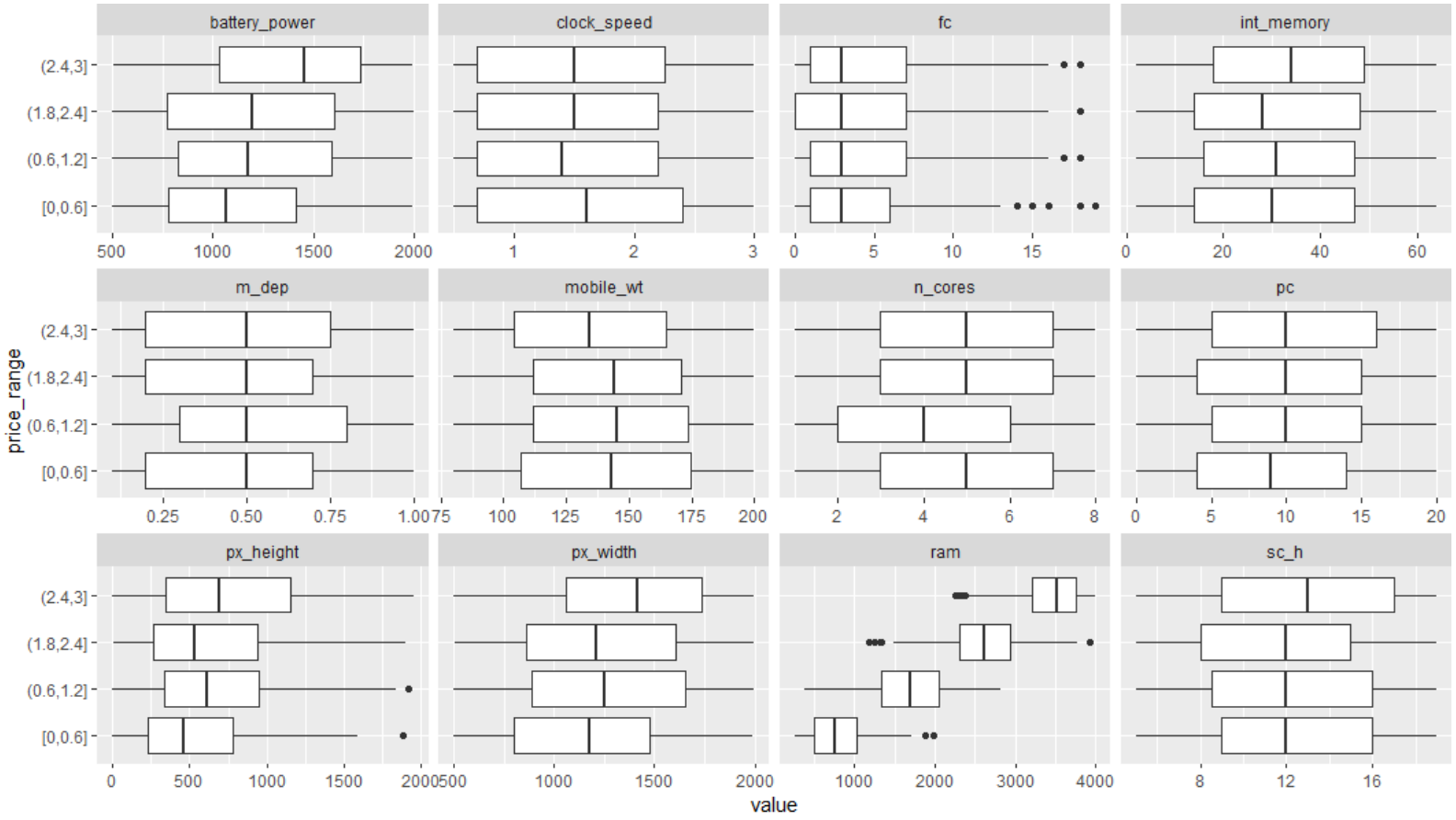
Explore Data

Using the `DataExplorer` package. Take a comprehensive review of the data.



Explore Data

Using the `DataExplorer` package.



Feature Engineering

- Any **missing data**?
- Coding of categorical variables. For `neuralnet` package, need indicator variable for each category.
- **Standardization**. Important!

Build Neural Network

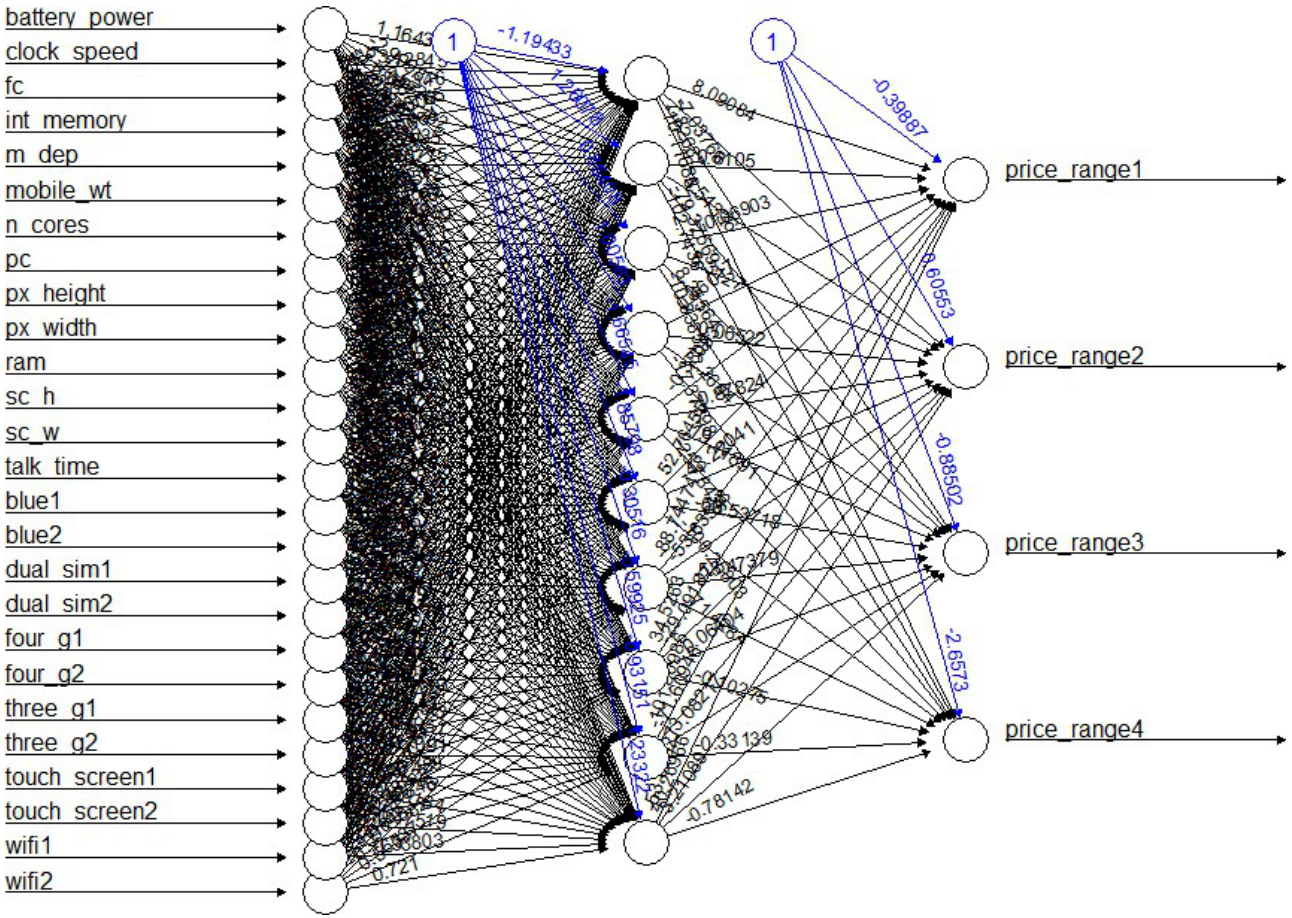
```
mods<-neuralnet(frmla,data=train.ind,rep=10,act.fct="logistic",
linear.output=FALSE,lifesign="minimal",hidden=10)
```

Nodes: Set at 10. Intuition. Something better?

Activation Function: Logistic

Weights: Tried 10 different starting weights.

Graph Neural Network



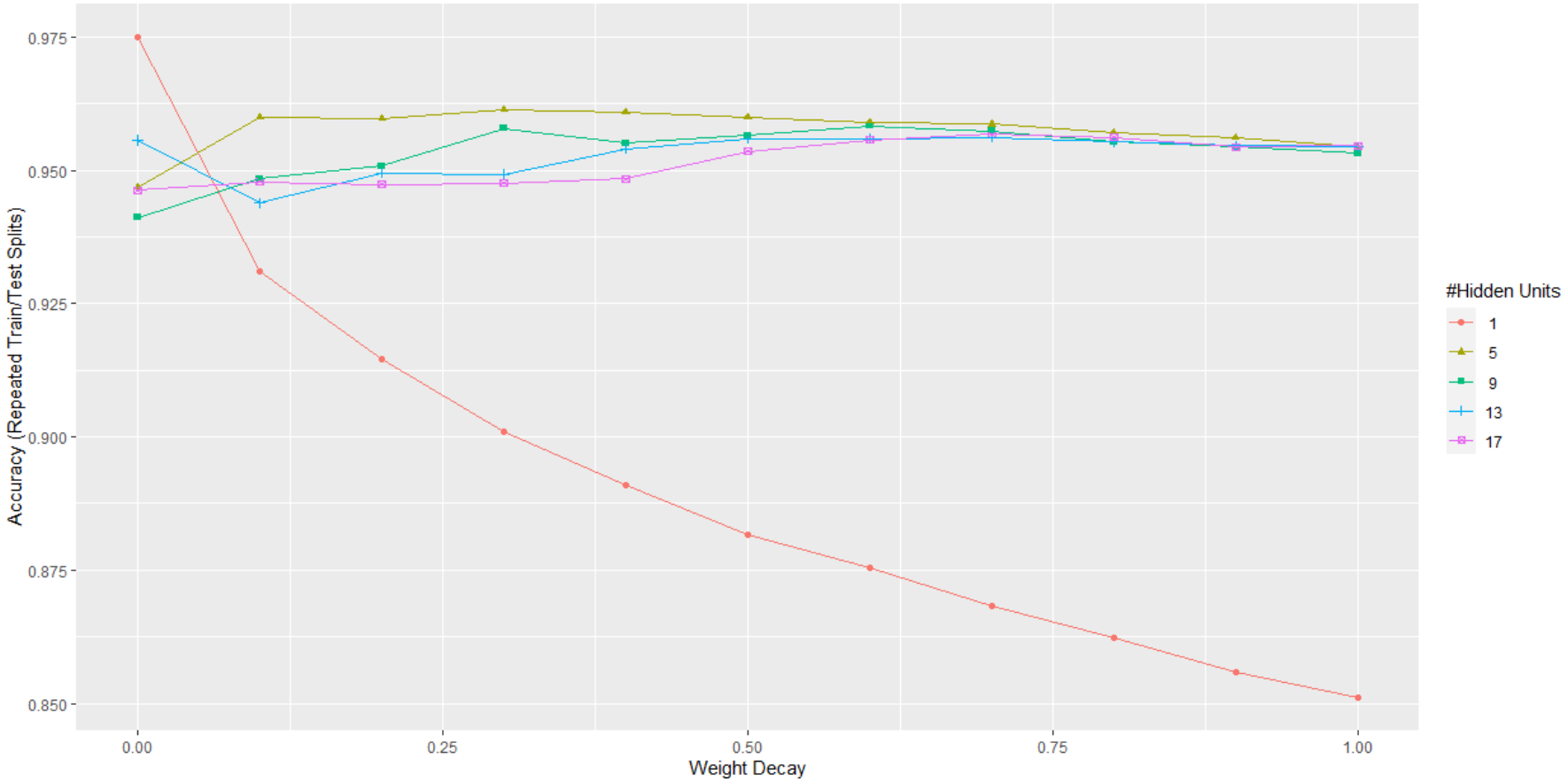
Evaluate Neural Network

Evaluate the neural network on the test set:

```
> xtab<-table(bestmod1.test$pred_price_range,bestmod1.
> confusionMatrix(xtab)
Confusion Matrix and Statistics
    0   1   2   3
0 123   2   0   0
1   2 120   4   0
2   0   3 114   3
3   0   0   6 122

Overall Statistics
Accuracy : 0.9599
95% CI : (0.9388, 0.9753)
```

Cross-Validation



Evaluate Neural Network

Look at variable importance (need an `nnet` object):

```
> imp<-varImp(best.nnet.mod)
> imp
      Overall       price_range1 price_range2 price_range3 price_range4
X1    5.148347      5.148347     5.148347     5.148347      5.148347
X2    2.325114      2.325114     2.325114     2.325114      2.325114
X3    2.469074      2.469074     2.469074     2.469074      2.469074
X4    1.951242      1.951242     1.951242     1.951242      1.951242
X5    1.925745      1.925745     1.925745     1.925745      1.925745
X6    2.655178      2.655178     2.655178     2.655178      2.655178
X7    3.309943      3.309943     3.309943     3.309943      3.309943
X8    3.070406      3.070406     3.070406     3.070406      3.070406
X9    5.184120      5.184120     5.184120     5.184120      5.184120
X10   4.694055      4.694055     4.694055     4.694055      4.694055
X11 19.162310     19.162310    19.162310    19.162310     19.162310
<...>
X26   3.516861      3.516861     3.516861     3.516861      3.516861
```

Evaluate Neural Network

On the previous side, it looked like the variable `ram` dominated the predictions.

As a check, run an ordered logistic regression model to see how well `ram` predicts by itself.

```
train$price_range <- factor(train$price_range, levels = c("0", "1", "2",
o.logit<-polr(price_range~ram,data=train)
test$olog_pred<-predict(o.logit,newdata=test)
xtab2<-table(test$price_range,test$olog_pred)
confusionMatrix(xtab2)
```

Evaluate Neural Network

The predictions are not as accurate, but not bad for a much simpler model.

The variable `ram` is an important one.

```
> confusionMatrix(xtab2)
Confusion Matrix and Statistics

    0   1   2   3
0  99  26   0   0
1  16  81  28   0
2   0  18  89  18
3   0   0  23 102

Overall Statistics
Accuracy : 0.742
```

Conclusion

- Artificial neural networks (ANNs) began as an attempt to mimic the functioning of the human brain.
- Although they may not really mimic the brain, they have provided a flexible way to make predictions.
- There are several important choices made by the analyst.
- Unclear how to make these choices other than to judge by the results.
- ANN approach has been extended in a number of ways. Convolutional and Recurrent NNs are two key examples.