Regression/Prediction

Road Map

- Why is prediction supervised learning, how it differs from classification.
- Statistical and machine learning techniques for regression
- Example with a synthetic dataset

What is Prediction?

A **prediction** or forecast is a statement about the way things will happen in the future, often but not always based on experience or knowledge.

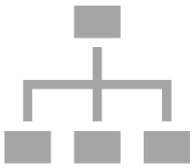
Why is Prediction Supervised Learning

We use labeled training data to learn and make prediction.

Supervised learning means we have an already marked dataset giving us what the learning process should give you.

Forms of data analysis to predict future data trends

There are two forms of data analysis that can be used for extracting models describing important classes or to predict future data trends. These two forms are as follows



Classification

Regression

Also, called prediction in earlier literature

Classification vs Regression

Classification	Regression/Prediction
It predicts categorical class labels	It predicts continuous valued functions
Example: A bank loan officer wants to analyze the data in order to know which customer (loan applicant) are risky or which are safe. A model or classifier is constructed to predict the categorical labels. These labels are risky or safe for loan application data	**Example:** Suppose the marketing manager needs to predict how much a given customer will spend during a sale at his company. In this example we are bothered to predict a numeric value.

Reference: https://www.tutorialspoint.com/

Linear Regression

- Simple linear regression is useful for finding relationship between two continuous variables.

One Variable → Predictor or independent variable

Second Variable → Response or dependent variable

$$\boxed{Y(pred) = b0 + b1^{*}x}$$

The values b0 and b1 must be chosen so that they minimize the error.

Example of Simple Linear Regression

Year	Sales (Million Euro)	Advertising (Million Euro)
1	651	23
2	762	26
3	856	30
4	1,063	34
5	1,190	43
6	1,298	48
7	1,421	52
8	1,440	57
9	1,518	58

The table shows some data from the early days of the Italian clothing company.

Each row in the table shows sales for a year and the amount spent on advertising that year.

Our outcome of interest is sales—it is what we want to predict.

Predictor variable → **Advertising**
Response variable → **Sales**

linear regression estimates that

Sales = 168 + 23 * (Advertising)

Reference: https://www.displayr.com/

Metrics used to evaluate regression model

- Mean Squared Error(MSE)
- Root-Mean-Squared-Error(RMSE)
- Mean Absolute Percentage Error (MAPE) → We will use this
- R^2 or Coefficient of Determination

Mean Absolute Percentage Error (MAPE)

- The **mean absolute percentage error** (MAPE) is a measure of how accurate a forecast system is.

- It measures this accuracy as a percentage.

- It can be calculated as the average absolute percent error for each time period minus actual values divided by actual values.

Mean Absolute Percentage Error (MAPE)

- **The mean absolute percentage error (MAPE)** is the mean or average of the absolute percentage errors of forecasts.

- **Error** is defined as actual or observed value minus the forecasted value.

- Percentage errors are summed without regard to sign to compute MAPE. This measure is easy to understand because it provides the error in terms of percentages.

- Absolute percentage errors are used so that the problem of positive and negative errors canceling each other out is avoided

Reference: (2000) MEAN ABSOLUTE PERCENTAGE ERROR (MAPE). In: Swamidass P.M. (eds) Encyclopedia of Production and Manufacturing Management. Springer, Boston, MA

Mean Absolute Percentage Error (MAPE)

$$M = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right| * 100$$

A_t is the actual value
F_t is the predicted value

MAPE function in R:

```
mape <- function(actual,pred){
  mape <- mean(abs((actual - pred)/actual))*100
  return (mape)
}
```

Let's take example in R

Generate Data

```
x=data.frame(array(runif(300,0.3,0.8),dim=c(100,3)))
xt=data.frame(array(runif(150,0.3,0.8),dim=c(50,3)))
y=4.5*x[,1]^2+9.8*x[,2]*x[,3]+99.45*x[,3]^3+runif(1,1,34)
yt=4.5*xt[,1]^2+9.8*xt[,2]*xt[,3]+99.45*xt[,3]^3+runif(1,1,34)
yn=y/max(y)
ytn=yt/max(yt)
xy=data.frame(cbind(x,y))
xyt=data.frame(cbind(xt,yt))
xyn=data.frame(cbind(x,yn))
xytn=data.frame(cbind(xt,ytn))
```

Here runif will generate **300** random numbers uniformly distributed in the range of **0.3** and **0.8**

dim will be used to divide those generated numbers into table of **3 columns** and **100 rows**.

Data will look like this

View(x)

	X1	X2	X3
1	0.4316818	0.6898455	0.5849657
2	0.7369424	0.4383378	0.6786981
3	0.6167242	0.6886222	0.3265534
4	0.6996586	0.7996451	0.3081884
5	0.3267022	0.6284825	0.3906798
6	0.6949486	0.3394825	0.4198198
7	0.4307763	0.5181644	0.4794293
8	0.5420672	0.4380126	0.6005047
9	0.5195857	0.6823469	0.4379652
10	0.7949684	0.3171315	0.3852136
11	0.7476107	0.3370799	0.7610011
12	0.7132139	0.4564759	0.7044068
13	0.5026017	0.5715201	0.4016921
14	0.6324253	0.3384096	0.5338413
15	0.7162864	0.6651050	0.3171963
16	0.6367444	0.5021912	0.5397307

Linear Regression in R

```
rg=lm(y~.,xy)
rg$fit
rg$coeff
mean(100*abs(y-rg$fit)/y)
ytp=predict(rg,xt)
mean(100*abs(yt-ytp)/yt)
```

Output:

```
> rg=lm(y~.,xy)
> rg$fit
        1         2         3         4         5         6         7         8         9        10        11 
 8.287530 58.780871 49.097472 50.823214 48.668865 54.476276 55.543849 34.943672 41.165091 45.630923 36.990703 
       12        13        14        15        16        17        18        19        20        21        22 
26.452686 56.207065 44.244764 11.725108 20.704086 18.971110 49.192855 54.996552 43.232410 36.581743 44.888777 
       23        24        25        26        27        28        29        30        31        32        33 
38.711407 15.941148 46.572410 10.960769 18.233461 34.292369 53.389469 49.831852 19.952174 58.052150 30.033776 
       34        35        36        37        38        39        40        41        42        43        44 
47.551920 10.756635 25.149276  5.912229 49.559999 58.212764 35.164841 45.737915 49.948902 24.078522 16.455608 
       45        46        47        48        49        50        51        52        53        54        55 
18.797455 34.266803 44.988344 13.151118 28.278329 11.432525 17.931799 24.720918 42.525139 21.117222 12.609382 
       56        57        58        59        60        61        62        63        64        65        66 
38.054001 19.283061 57.349967 51.001691 31.590009 31.127530 11.555928 42.412536 44.799577 47.512773 11.066296 
       67        68        69        70        71        72        73        74        75        76        77 
30.351995 43.928757  9.217881  8.786397 12.331101 15.660139 23.417174 16.879007 53.676813 14.375929 24.508842 
       78        79        80        81        82        83        84        85        86        87        88 
11.925584 50.582847 51.612638 36.767328 16.256636 50.173751 11.235027 21.432071 50.104004 13.104860 16.810715 
       89        90        91        92        93        94        95        96        97        98        99 
41.358406 41.304977 31.405505 25.508668 24.068898  7.563596 55.813646 15.482629 57.432380 44.350239 55.943496 
      100 
49.581100 
> rg$coeff
(Intercept)          X1          X2          X3 
 -27.899533    2.230619    6.475809  100.903011 
> mean(100*abs(y-rg$fit)/y)
[1] 10.36791
> ytp=predict(rg,xt)
> mean(100*abs(yt-ytp)/yt)
[1] 32.74954
```

Linear Regression in R

```
rg=lm(y~.,xy)
rg$fit
rg$coeff
mean(100*abs(y-rg$fit)/y)
ytp=predict(rg,xt)
mean(100*abs(yt-ytp)/yt)
```

lm command takes the variables in the format:
lm([target variable] ~ [predictor variables], data = [data source])

Coeff are the values of intercept and slope.
Intercept over here is -27. 899533 and slope values for x1, x2 & x3 are respectively.

Neural Network

- Neural networks are multi-layer networks of that we use to classify things, make predictions, etc.

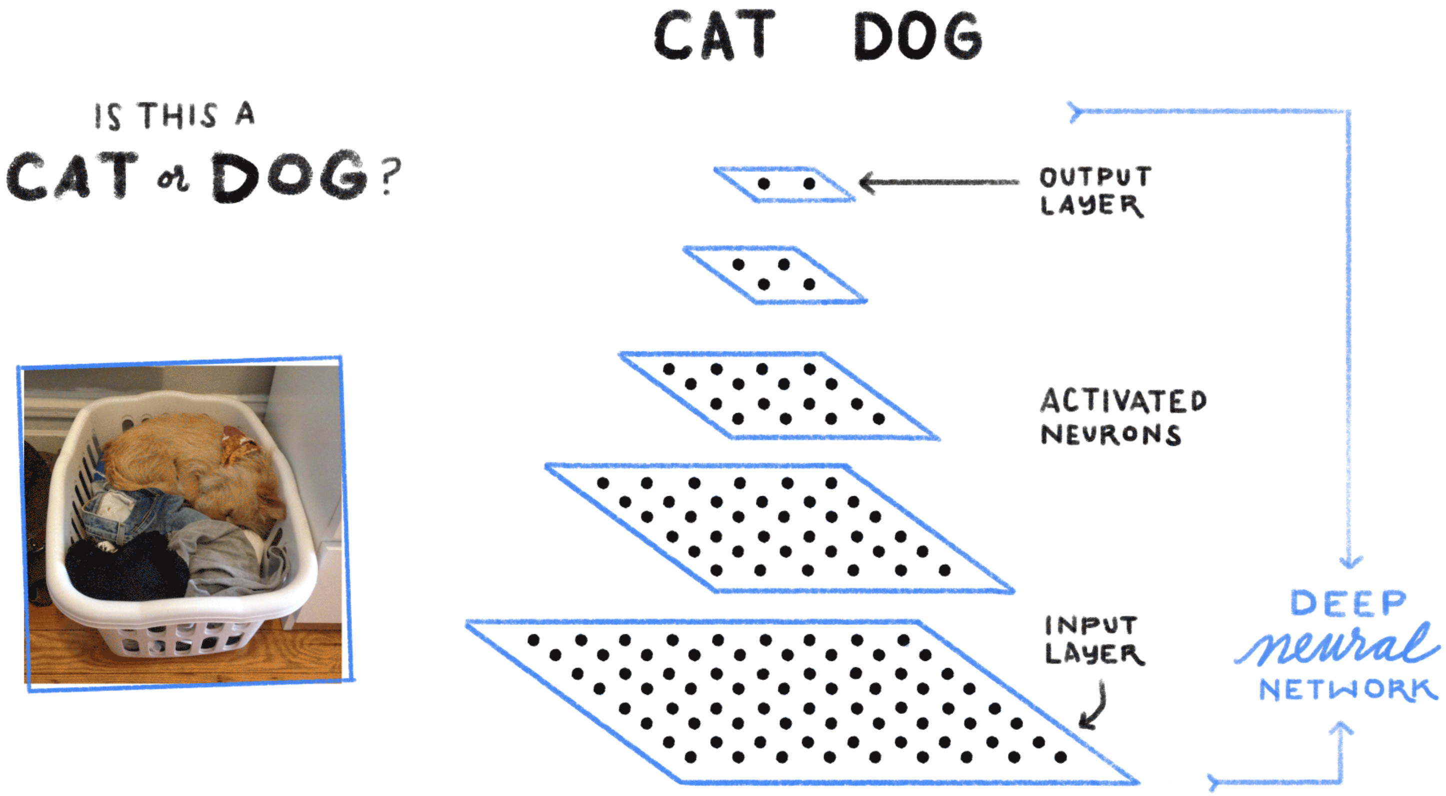
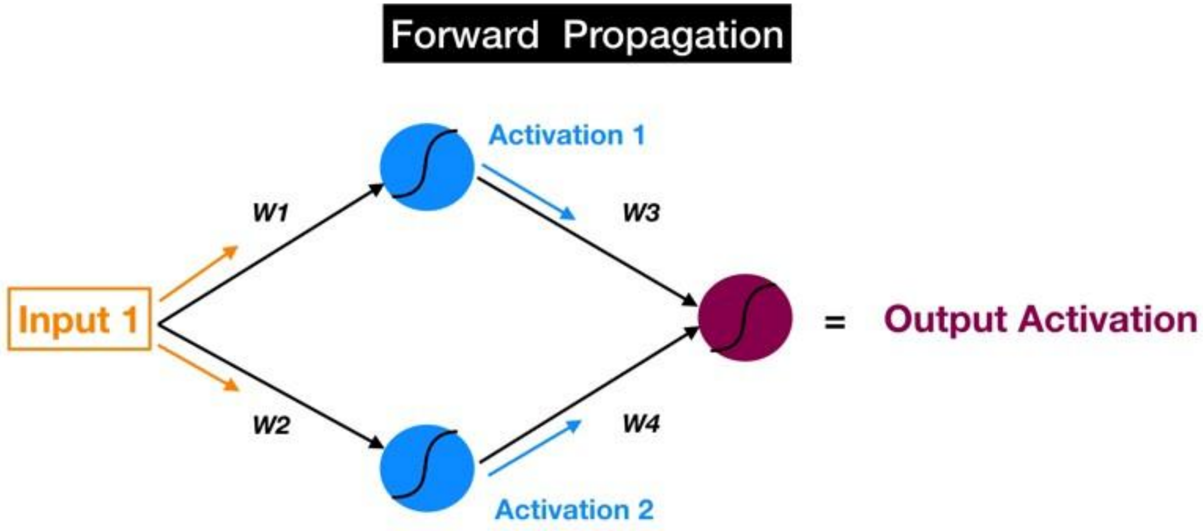


Image source: http://selmandesign.com/wp-content/uploads/2016/12/SelmanDesign_Q-A_CATorDOG-flow.gif

Forward Propagation

Objective of forward propagation is to calculate the activations at each neuron for each successive hidden layer until we arrive at the output.

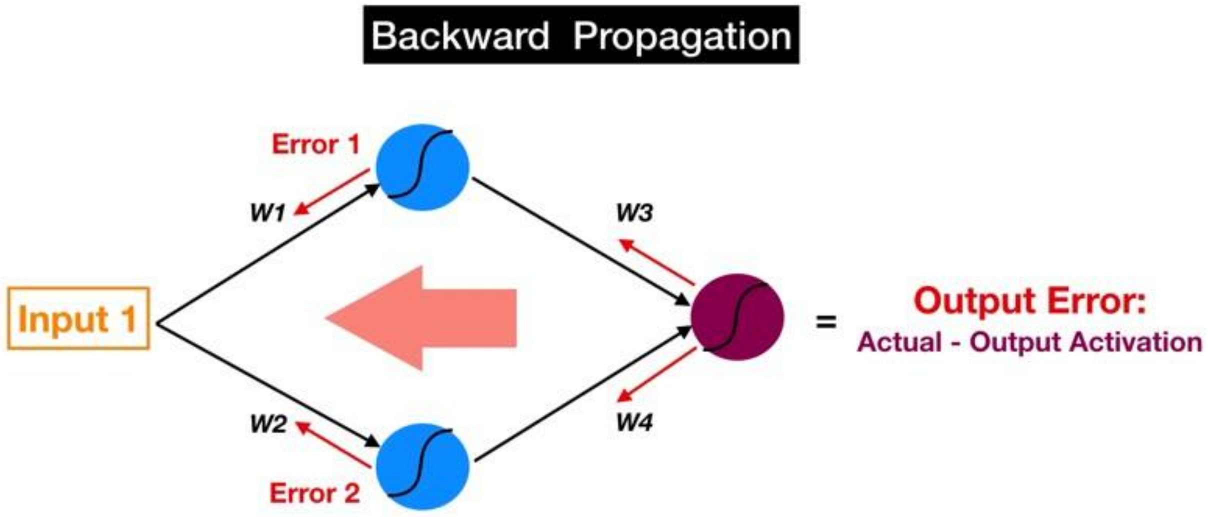
We just did forward propagation in previous steps.



Backward Propagation

We want to calculate the error attributable to each neuron starting from the layer closest to the output all the way back to the starting layer of our model.

The two building blocks of a neural network are the connections that pass signals into a particular neuron (with a weight living in each connection) and the neuron itself (with a bias). **These weights and biases across the entire network are also the dials that we tweak to change the predictions made by the model.**

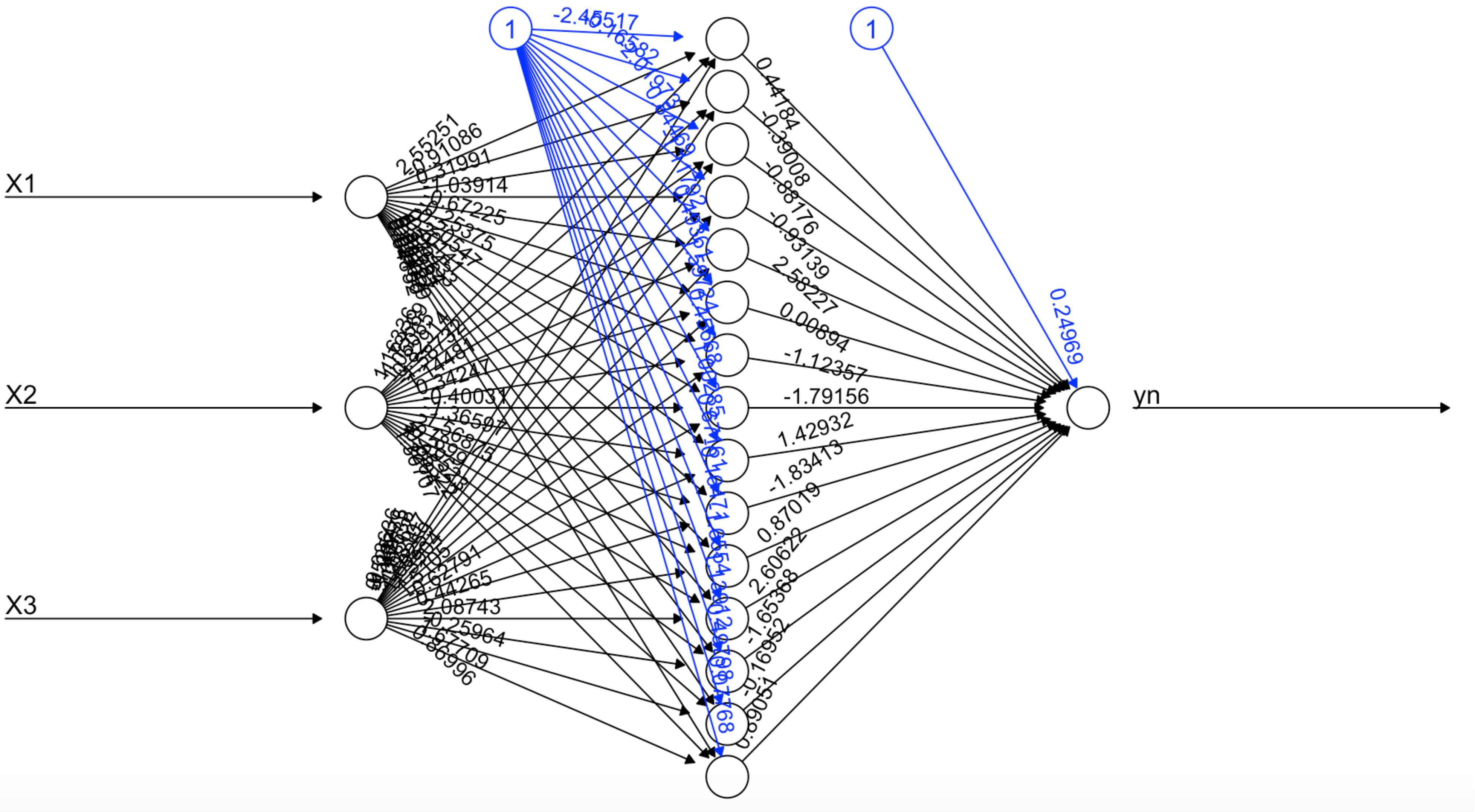


Neural network in R

We will use the same data which we used in Statistical Regression Example

	X1	X2	X3
1	0.4316818	0.6898455	0.5849657
2	0.7369424	0.4383378	0.6786981
3	0.6167242	0.6886222	0.3265534
4	0.6996586	0.7996451	0.3081884
5	0.3267022	0.6284825	0.3906798
6	0.6949486	0.3394825	0.4198198
7	0.4307763	0.5181644	0.4794293
8	0.5420672	0.4380126	0.6005047
9	0.5195857	0.6823469	0.4379652
10	0.7949684	0.3171315	0.3852136
11	0.7476107	0.3370799	0.7610011
12	0.7132139	0.4564759	0.7044068
13	0.5026017	0.5715201	0.4016921
14	0.6324253	0.3384096	0.5338413
15	0.7162864	0.6651050	0.3171963
16	0.6367444	0.5021912	0.5397307

Plot


X1
X2
X3
yn
1
1
0.24969
0.44184
-0.39008
-0.88176
-0.93139
2.58227
0.00894
-1.12357
-1.79156
1.42932
-1.83413
0.87019
2.60622
-1.03914
-0.40031
-2.08743

Neural net in R

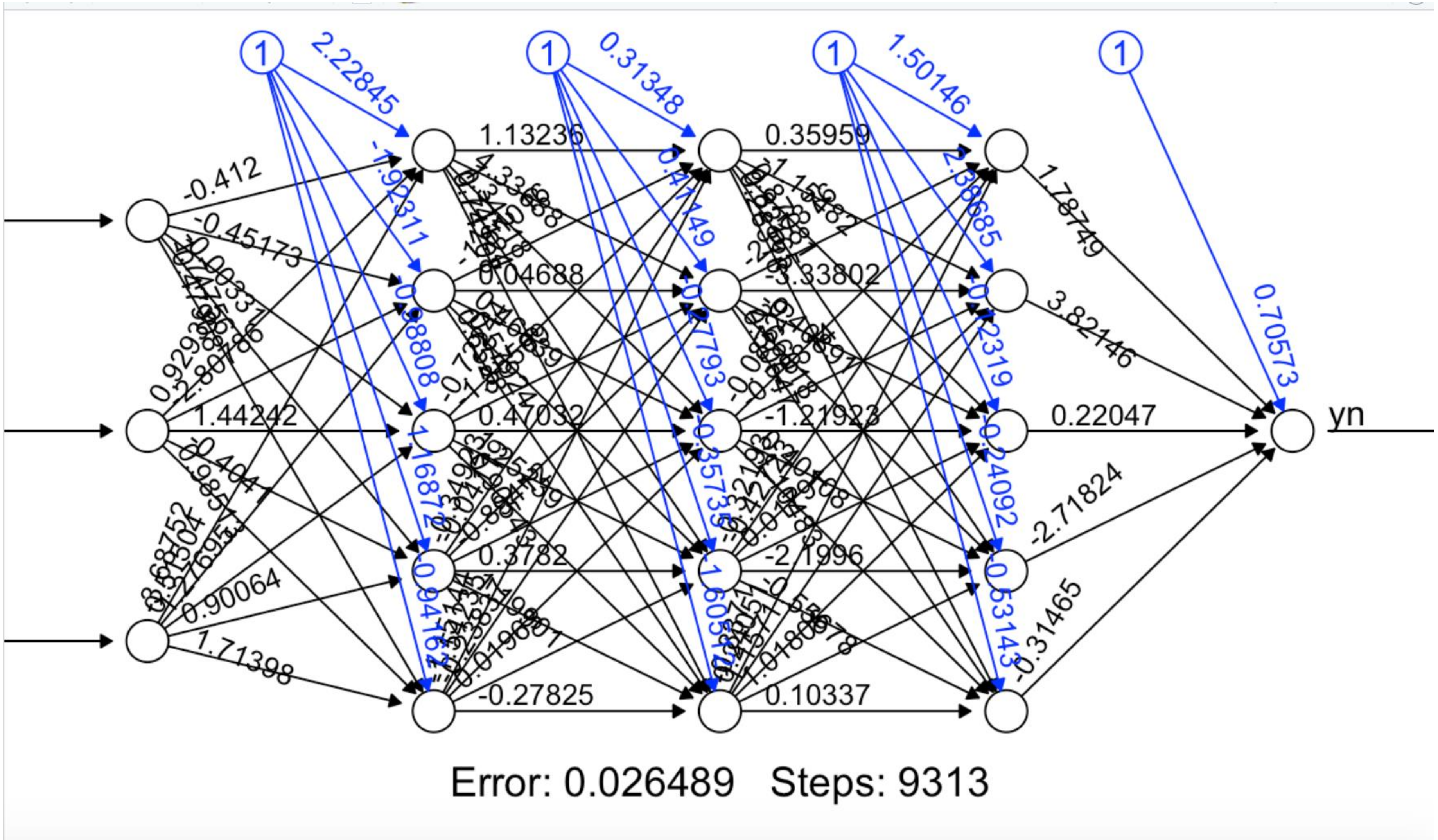
```
install.packages("neuralnet")
library(neuralnet)
nn=neuralnet(yn~X1+X2+X3,data=xyn,hidden=15, act.fct="logistic",linear.out=F
             algorithm="backprop",learningrate=0.01)
nn$net.result[1]
mean(100*abs(nn$net.result[[1]]-yn)/yn)
nnp=compute(nn,covariate=xt)
mean(100*abs(array(nnp$net.result)-ytn)/ytn)
plot(nn,rep="best")
dev.off()
```

Output:

```
> nn=neuralnet(yn~X1+X2+X3,data=xyn,hidden=15, act.fct="logistic",linear.out=F,
+             algorithm="backprop",learningrate=0.01)
> nn$net.result[1]
[[1]]
           [,1]
 [1,] 0.3557004
 [2,] 0.3623581
 [3,] 0.6685760
 [4,] 0.4544843
 [5,] 0.6521025
 [6,] 0.7667168
 [7,] 0.8202123
 [8,] 0.4110078
 [9,] 0.4250628
[10,] 0.6366724
[11,] 0.7721598
```

```
> mean(100*abs(nn$net.result[[1]]-yn)/yn)
[1] 5.57832
> nnp=compute(nn,covariate=xt)
> mean(100*abs(array(nnp$net.result)-ytn)/ytn)
[1] 25.13558
```

NN architecture with three hidden layers



NN architecture with three hidden layers in R

```
# Try with three hidden layers

nn=neuralnet(yn~X1+X2+X3,data=xyn,hidden=c(5,5,5),
             act.fct="logistic",linear.out=F,
             algorithm="backprop",learningrate=0.01)
nn$net.result[1]
mean(100*abs(nn$net.result[[1]]-yn)/yn)
nntp=compute(nn,covariate=xt)
mean(100*abs(array(nntp$net.result)-ytn)/ytn)
plot(nn,rep="best")
dev.off()
```

Output:

```
> nn=neuralnet(yn~X1+X2+X3,data=xyn,hidden=c(5,5,5),
+              act.fct="logistic",linear.out=F,
+              algorithm="backprop",learningrate=0.01)
> nn$net.result[1]
[[1]]
          [,1]
  [1,] 0.4072923
  [2,] 0.4137022
  [3,] 0.6286215
  [4,] 0.4658868

> mean(100*abs(nn$net.result[[1]]-yn)/yn)
[1] 2.01242
> nntp=compute(nn,covariate=xt)
> mean(100*abs(array(nntp$net.result)-ytn)/ytn)
[1] 23.91774
> plot(nn,rep="best")
```

Support Vector Machine (SVM)

- SVM or Support Vector Machine is a linear model for classification and regression problems.
- The algorithm creates a line or a hyperplane which separates the data into classes.

SVM in R

We will use the same data which we used in Neural Network Example

	X1	X2	X3
1	0.4316818	0.6898455	0.5849657
2	0.7369424	0.4383378	0.6786981
3	0.6167242	0.6886222	0.3265534
4	0.6996586	0.7996451	0.3081884
5	0.3267022	0.6284825	0.3906798
6	0.6949486	0.3394825	0.4198198
7	0.4307763	0.5181644	0.4794293
8	0.5420672	0.4380126	0.6005047
9	0.5195857	0.6823469	0.4379652
10	0.7949684	0.3171315	0.3852136
11	0.7476107	0.3370799	0.7610011
12	0.7132139	0.4564759	0.7044068
13	0.5026017	0.5715201	0.4016921
14	0.6324253	0.3384096	0.5338413
15	0.7162864	0.6651050	0.3171963
16	0.6367444	0.5021912	0.5397307

SVM in R

```r
install.packages("e1071")
library(e1071)
svr=svm(x,y)
yp=predict(svr,x)
mean(100*abs(y-yp)/y)
ytp=predict(svr,xt)
mean(100*abs(yt-ytp)/yt)
svr=svm(x,y,kernel="radial")
yp=predict(svr,x)
mean(100*abs(y-yp)/y)
ytp=predict(svr,xt)
mean(100*abs(yt-ytp)/yt)
svr=svm(x,y,kernel="polynomial",degree="3")
yp=predict(svr,x)
mean(100*abs(y-yp)/y)
svr=svm(x,yn,kernel="sigmoid")
ynp=predict(svr,x)
mean(100*abs(yn-ynp)/yn)
```

Output:

```
> svr=svm(x,y)
> yp=predict(svr,x)
> mean(100*abs(y-yp)/y)
[1] 16.93969
> ytp=predict(svr,xt)
> mean(100*abs(yt-ytp)/yt)
[1] 69.41927
> svr=svm(x,y,kernel="radial")
> yp=predict(svr,x)
> mean(100*abs(y-yp)/y)
[1] 16.93969
> ytp=predict(svr,xt)
> mean(100*abs(yt-ytp)/yt)
[1] 69.41927
> svr=svm(x,y,kernel="polynomial",degree="3")
> yp=predict(svr,x)
> mean(100*abs(y-yp)/y)
[1] 20.16147
> svr=svm(x,yn,kernel="sigmoid")
> ynp=predict(svr,x)
> mean(100*abs(yn-ynp)/yn)
[1] 54.2352
```

Comparison

Type	Training Error (MAPE)	Test Error (MAPE)
Linear Regression	10.36%	32.75%
NN with 1 hidden layer (15)	5.58%	25.14%
NN with 3 hidden layer (5, 5, 5)	2.02%	23.92%
SVM	16.94%	69.42%
SVM (Kernel = “radial”)	16.94%	69.42%
SVM (Kernel = “polynomial”, degree=“3”)	20.16%	NA
SVM (Kernel = “sigmoid”)	54.23%	NA

The smaller the MAPE the better the forecast.