ARTIFICIAL NEURAL NETWORK

An **artificial neural network (ANN)** is a subfield of Artificial Intelligence where it attempts to mimic the network of neurons that makes up a human brain so that computers will have the option to understand things and make decisions in a human-like manner. The ANN is designed by programming computers to behave simply like interconnected brain cells.

An ANN consists of an **input layer**, **hidden layers** of **nodes** (or **neurons**, or **perceptrons**), and an **output layer**. The first layer receives raw input, it is processed by multiple hidden layers, and the last layer produces the result.

Historical Note: The oldest type of neural network, known as **Perceptron**, was introduced by Frank Rosenblatt in 1958.

Rosenblatt, F. (1958). ”The perceptron: A probabilistic model for information storage and organization in the brain.” *Psychological Review*, 65(6), 386–408.

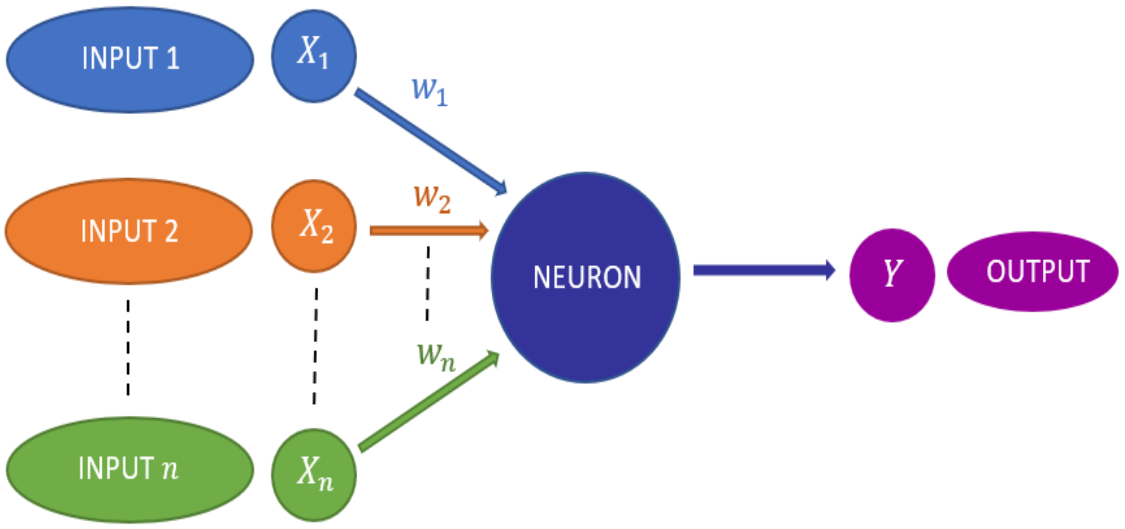
CORNELL CHRONICLE

Division of Rare and Manuscript Collections

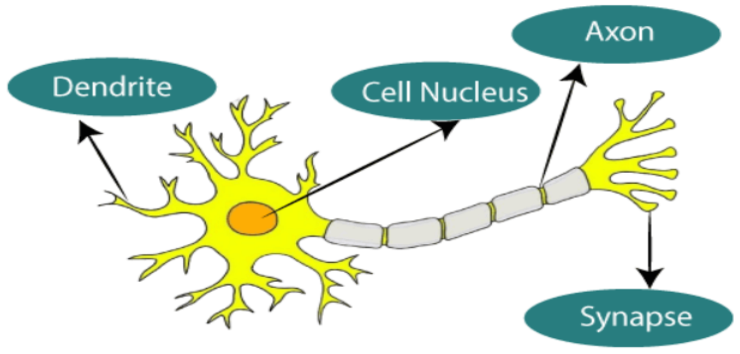
Frank Rosenblatt '50, Ph.D. '56, works on the “perceptron” – what he described as the first machine “capable of having an original idea.”

Professor's perceptron paved the way for AI – 60 years too soon

A typical ANN looks something like this:



A typical diagram of a biological neural network in the brain looks like this:

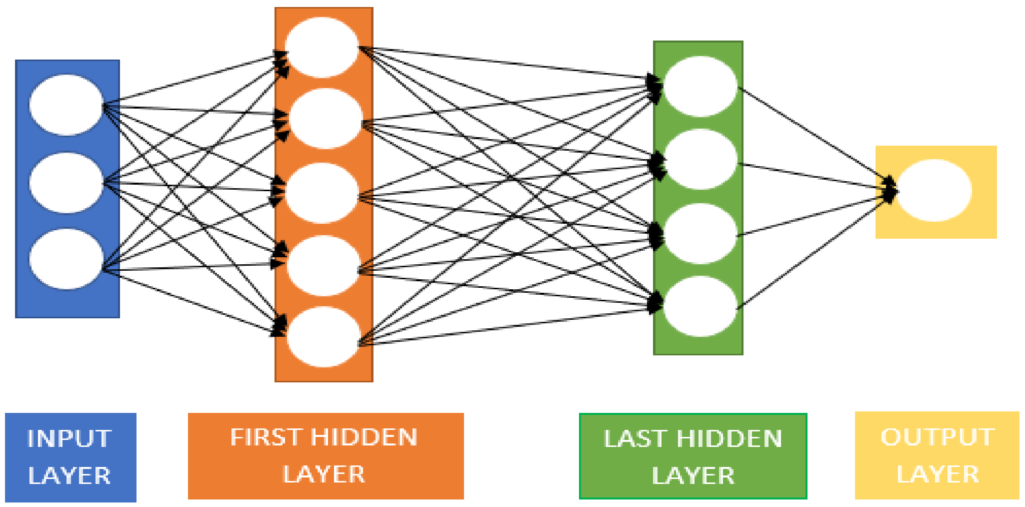


Dendrites from biological neural networks represent inputs in ANN, cell nucleus represents nodes, synapse represents weights, and axon represents output.

Glossary

- **Dendrite** is a short-branched extension of a nerve cell, along which impulses received from other cells at synapses are transmitted to the cell body.
- **Synapse** is a junction between two nerve cells, consisting of a minute gap across which impulses pass by diffusion of a neurotransmitter.
- **Axon** is a long threadlike part of a nerve cell along which impulses are conducted from the cell body to other cells.

To understand the concept of the architecture of an ANN, we have to understand what a neural network consists of. In order to define a neural network that consists of a large number of artificial neurons, which are nodes arranged in a sequence of layers. Let us look at three types of layers available in an ANN: input layer, hidden layer, and output layer.



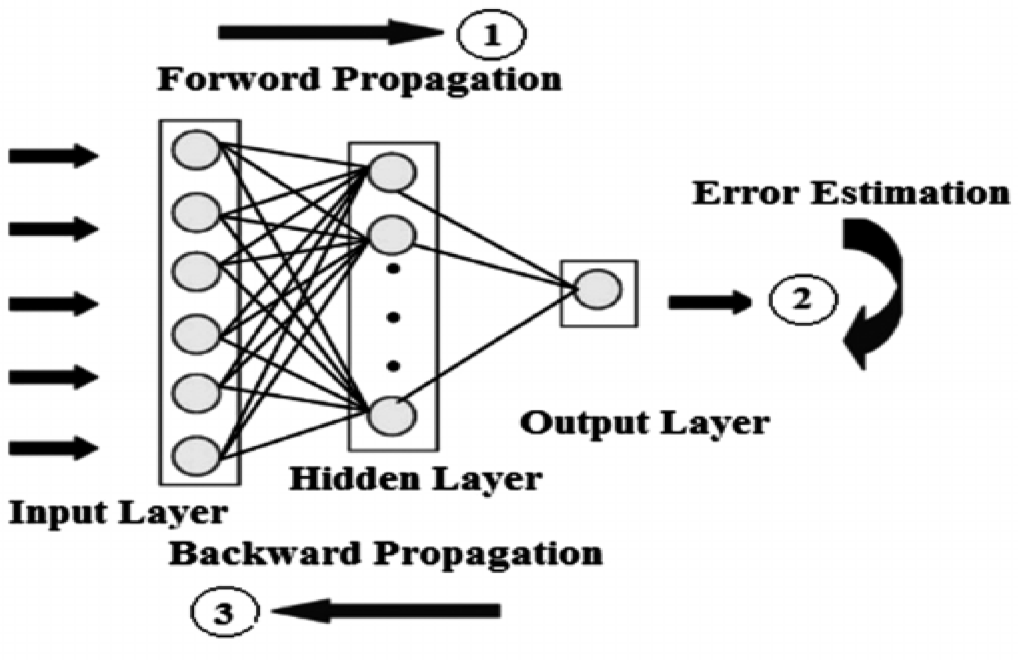
Input Layer accepts inputs provided by a programmer. The **input features** (or the predictor variables) can be categorical or numeric.

Hidden Layer performs all the calculations to find hidden features and patterns.

Output Layer consists of the output variable (or response variable). For regression ANNs, the output variable is numeric; for binary ANN, the output variable is binary, and for multinomial ANN, the output variable assumes multi-class values.

In an ANN, the input goes through a series of transformations using the hidden layer, which finally results in the output expressed as a linear combination of weighted input features with a bias term included.

It determines the weighted total that is passed as an input to an **activation function** to produce the output. Activation functions choose whether a node should fire or not. Only those who are fired make it to the output layer. This process is called **feed forward** (or **forward propagation**). After producing the output, an error (or loss) is calculated and a correction is sent back to the network. This process is known as **back propagation** (or **backward propagation**).



Historical Note. ANN with back propagation was introduced in Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). "Learning representations by back-propagating errors". *Nature*, 323(6088), 533–536. Most of the ANN applications in the literature utilize multi-layer feed-forward with a back propagation learning algorithm.

An **epoch** is a complete cycle through the full training set when building an ANN. An **iteration** is the number of steps through partitioned packets of the training data, needed to complete one epoch.

Learning Algorithm

An ANN starts with a set of initial weights and then gradually modifies the weights during the training cycle to settle down to a set of weights capable of realizing the input-output mapping with a minimum error.

Denote by $\mathbf{x_i} = (x_{i1}, \dots, x_{ik})'$, $i = 1, \dots, n$, the set of vectors of input variables (predictor variables), and let $\hat{\mathbf{y}} = (\hat{y}_1, \dots, \hat{y}_n)$ be the output vector. Also, suppose there is one hidden layer with m neurons $h_1, \dots, h_m$. The response of the hidden layer for the ith individual is the vector $\mathbf{h_i} = (h_{i1}, \dots, h_{im})'$. An ANN produces outputs governed by the relations:

$$\mathbf{h_i} = f\big(\mathbf{W_h}\,\mathbf{x_i} + \mathbf{b_i}\big), \text{ and } \hat{y}_i = f(\mathbf{W_i^*}\,\mathbf{h_i} + b_i^*),$$

where f is the activation function,

$$\mathbf{W_h} = \begin{bmatrix} w_{11} & \dots & w_{1k} \\ \dots & \dots & \dots \\ w_{m1} & \dots & w_{mk} \end{bmatrix}$$

is the hidden layer weight matrix, $\mathbf{W_i^*} = (w_{i1}^*, \dots, w_{im}^*)$ is the vector of output weights for individual i, $\mathbf{b_i} = (b_{i1}, \dots, b_{im})'$ is the hidden layer bias vector for individual i, and b_i^* is the output layer bias for individual i.

The activation functions that are used in SAS, R, and Python are (defined for $x \in \mathbb{R}$) **logistic** (or **sigmoid**) $f(x) = \dfrac{\exp(x)}{1+\exp(x)}$, and **hyperbolic tangent** (or **tanh**) $f(x) = \tanh(x) = \dfrac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$.

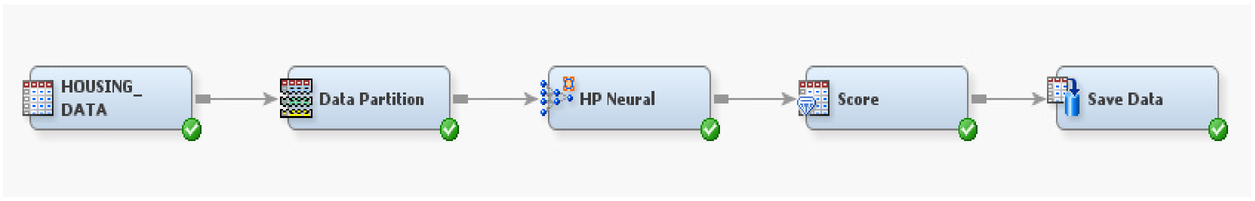
The loss functions used to compute errors in the back propagation algorithm are: mean squared error for regression and cross-entropy for classification.

The method of **steepest descent** is used to update the weights. For example, for the mean squared error loss function, the loss function is $L = \dfrac{1}{n}\sum\limits_{i=1}^{n}\Big(y_i - f(\mathbf{W_i^*}\,\mathbf{h_i} + b_i^*)\Big)^2$. The weights are updated according to the recursive relation $w_{ij}^*(new) = w_{ij}^*(old) - \lambda\dfrac{\partial L}{\partial w_{ij}^*}, j = 1, \dots, m,$, where λ is referred to as **learning rate**. The same algorithm applies to the weights in the hidden layers W_h.

Example. We fit an ANN to the housing data.

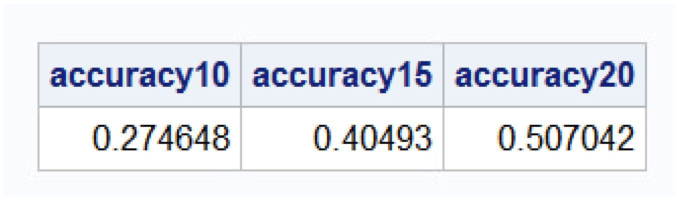
In SAS:

We run the following path diagram in Enterprise Miner, choosing logistic as the activation function.



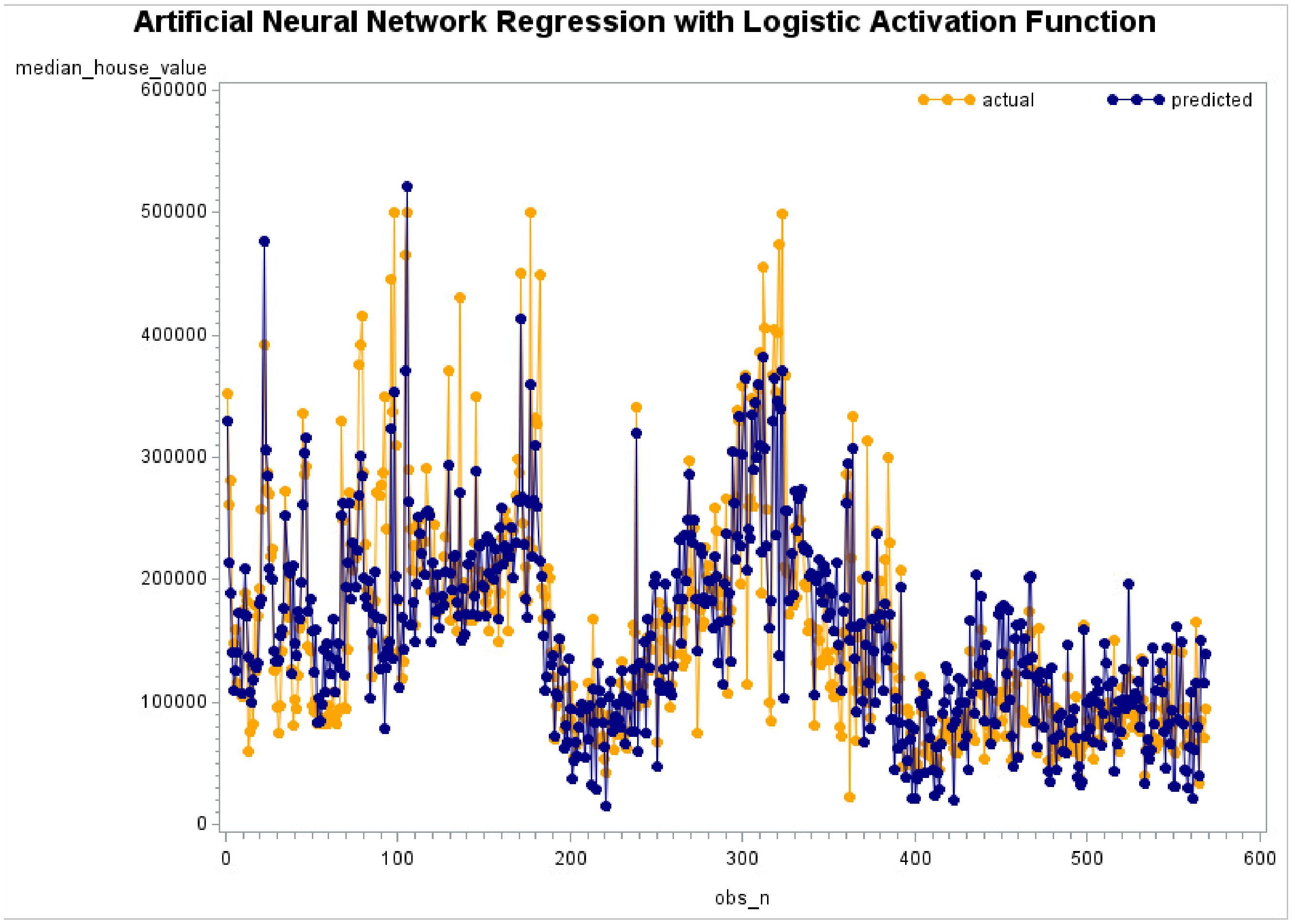
Then we run the following code to compute the accuracy.

```
data accuracy;
set tmp1.em_save_test;
ind10=(abs(R_median_house_value)<0.10*median_house_value);
ind15=(abs(R_median_house_value)<0.15*median_house_value);
ind20=(abs(R_median_house_value)<0.20*median_house_value);
obs_n=_N_;
run;

proc sql;
select mean(ind10) as accuracy10,
mean(ind15) as accuracy15, mean(ind20) as
accuracy20
from accuracy;
quit;
```

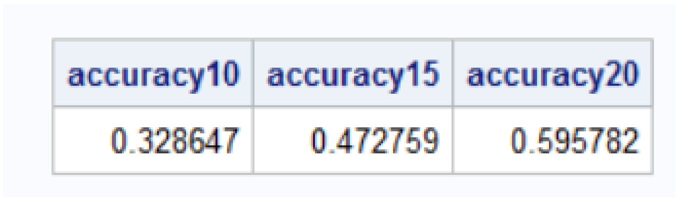


accuracy10	accuracy15	accuracy20
0.274648	0.40493	0.507042

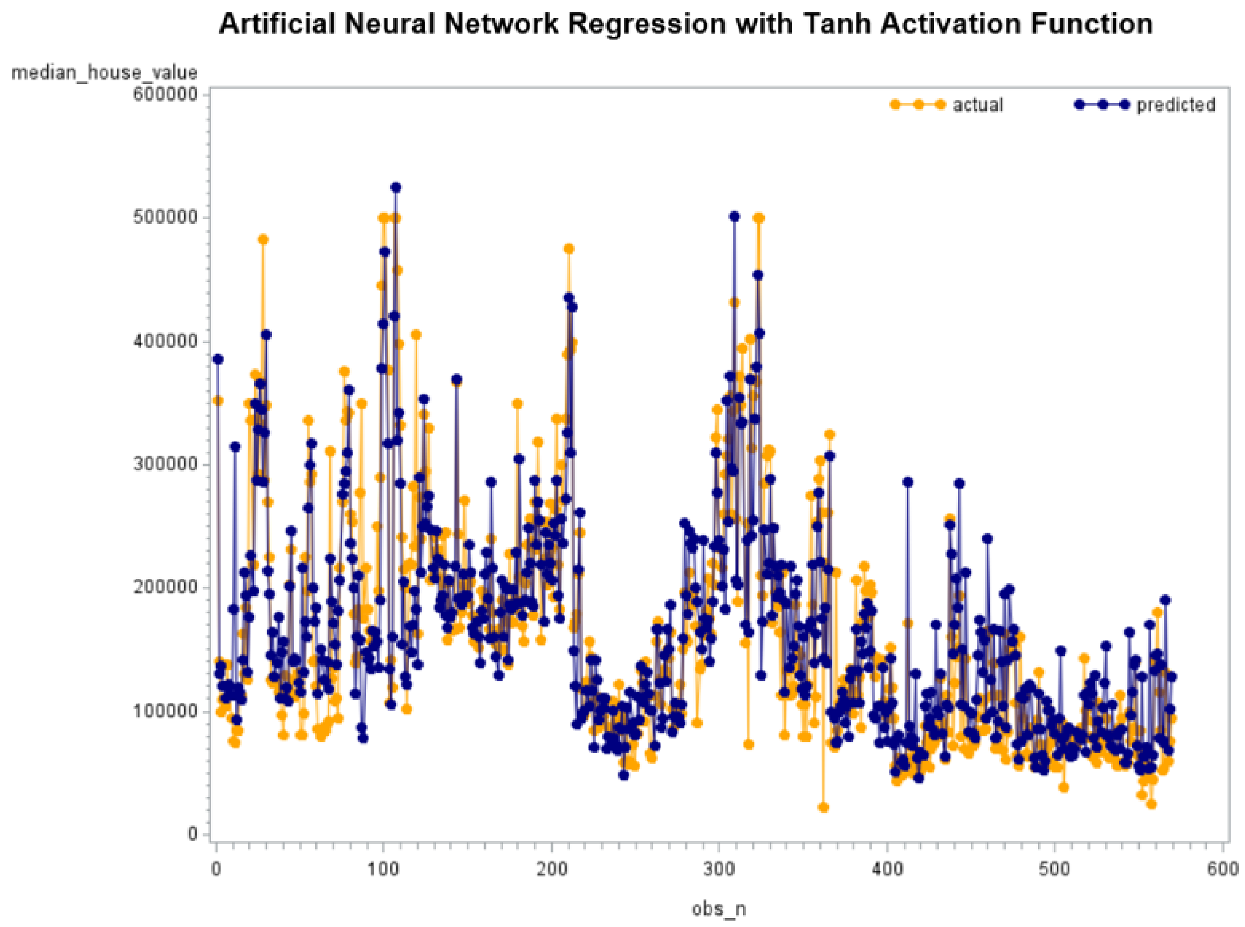
```
/*PLOTTING ACTUAL AND PREDICTED VALUES FOR TESTING DATA*/;
goptions reset=all border;
title1 "Artificial Neural Network Regression with Logistic Activation Function";
symbol1 interpol=join value=dot color=orange;
symbol2 interpol=join value=dot color=navy;
legend1 value=("actual" "predicted")
position=(top right inside) label=none;
proc gplot data=accuracy;
plot median_house_value*obs_n
EM_PREDICTION*obs_n/ overlay legend=legend1;
run;
```



Next, we run the same diagram, changing the activation function to the default function tanh. The accuracy and plot are given below.



accuracy10	accuracy15	accuracy20
0.328647	0.472759	0.595782

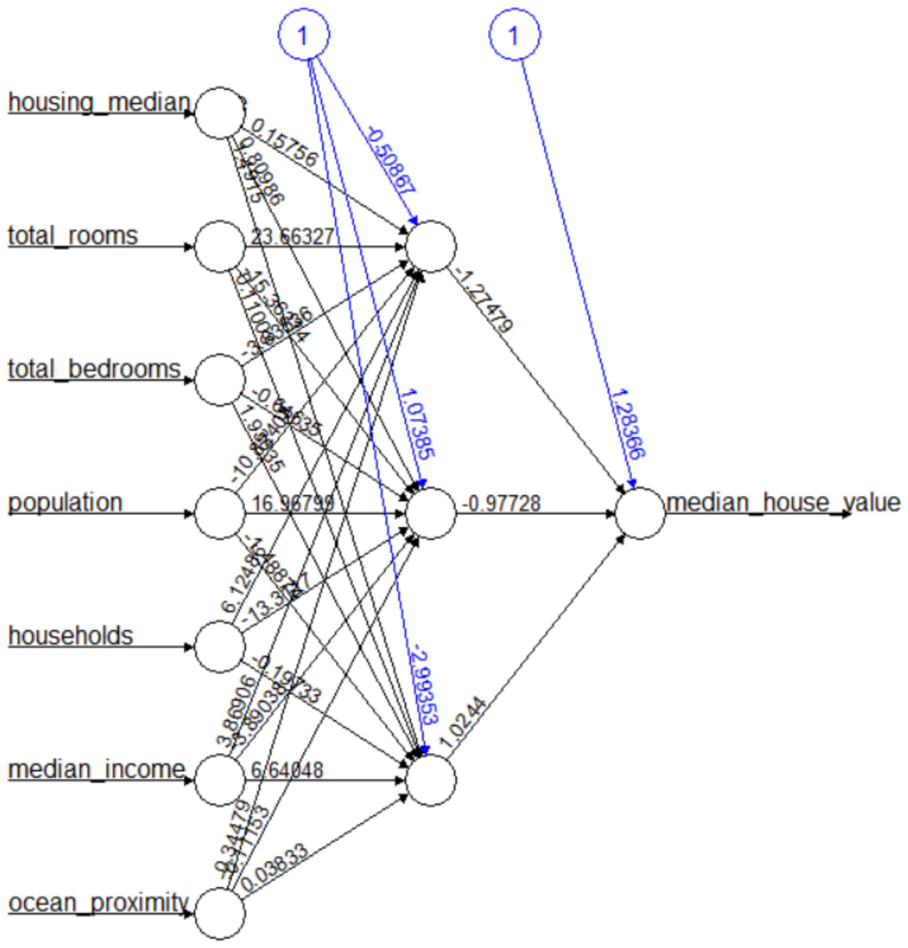


The fitted ANN model with the tanh activation function has a higher accuracy than that with the logistic activation function.

In R:

```
housing.data<- read.csv(file="./housing_data.csv", header=TRUE, sep=",")

housing.data$ocean_proximity<- ifelse(housing.data$ocean_proximity=='<1H OCEAN', 1,
ifelse(housing.data$ocean_proximity=='INLAND',2, ifelse(housing.data$ocean_proximity=='NEAR
BAY',3,4)))

#SCALING VARIABLES TO FALL IN [0,1]
library(dplyr)
```

```
scale01 <- function(x){
    (x-min(x))/(max(x)-min(x))
}

housing.data<- housing.data %>% mutate_all(scale01)

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(346634)
sample <- sample(c(TRUE, FALSE), nrow(housing.data), replace=TRUE, prob=c(0.8,0.2))
train<- housing.data[sample,]
test<- housing.data[!sample,]
test.x<- data.matrix(test[-8])
test.y<- data.matrix(test[8])

#FITTING ANN WITH LOGISTIC ACTIVATION FUNCTION
#install.packages("neuralnet")
library(neuralnet)
ann.reg<- neuralnet(median_house_value ~ housing_median_age+total_rooms+total_bedrooms
+population+households+median_income +ocean_proximity, data=train, hidden=3, act.fct="logistic")

#PLOTTING THE DIAGRAM
plot(ann.reg)
```



```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.y<- predict(ann.reg, test.x)

#accuracy within 10%
accuracy10<- ifelse(abs(test.y-pred.y)<0.10*test.y,1,0)

#accuracy within 15%
accuracy15<- ifelse(abs(test.y-pred.y)<0.15*test.y,1,0)

#accuracy within 20%
accuracy20<- ifelse(abs(test.y-pred.y)<0.20*test.y,1,0)

print('Prediction Accuracy')
print(paste('within 10%:', round(mean(accuracy10),4)))
```

```
 "within 10%: 0.2702"
```

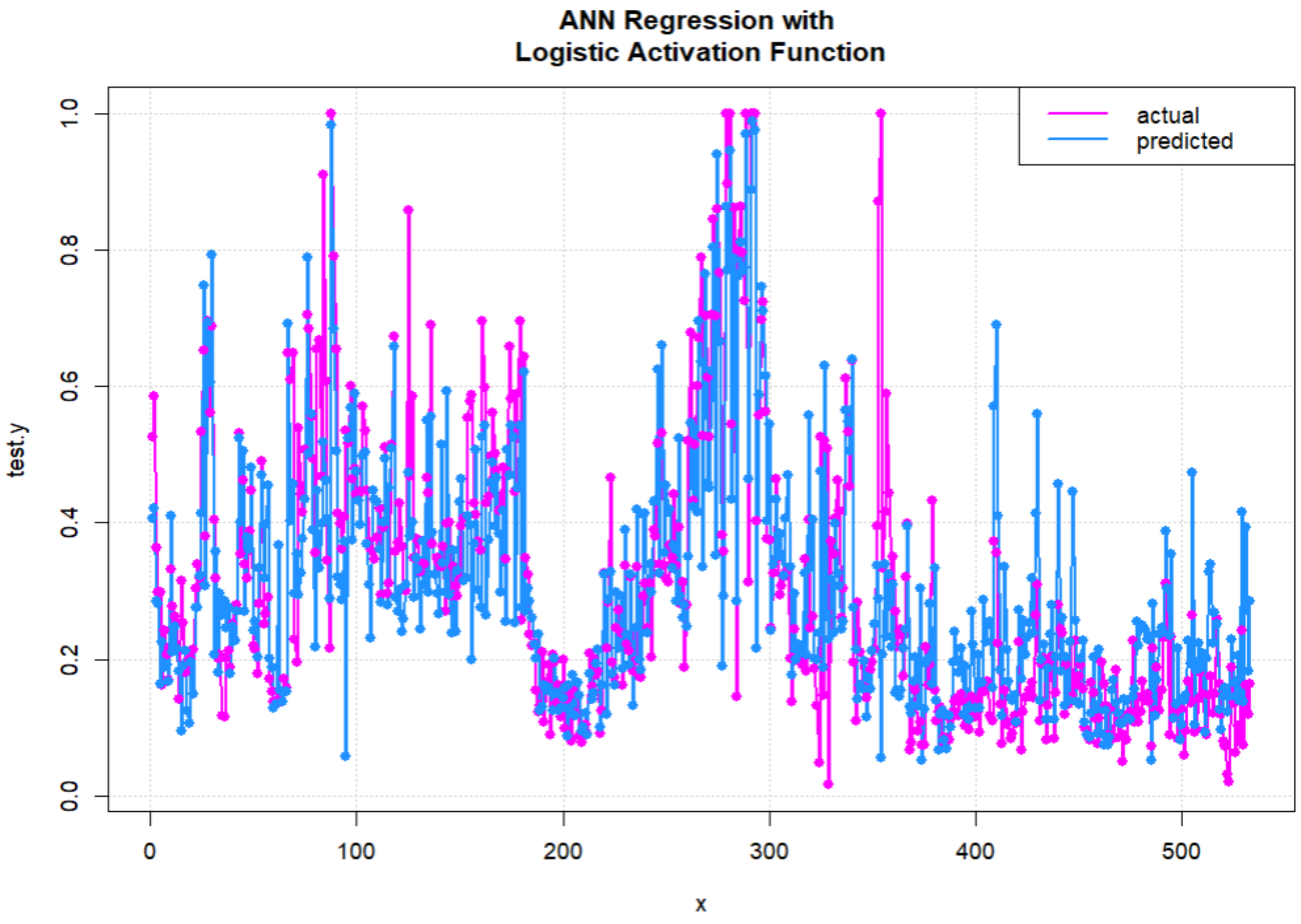
```
print(paste('within 15%:', round(mean(accuracy15),4)))
```

```
"within 15%: 0.3809"
```

```
print(paste('within 20%:', round(mean(accuracy20),4)))
```

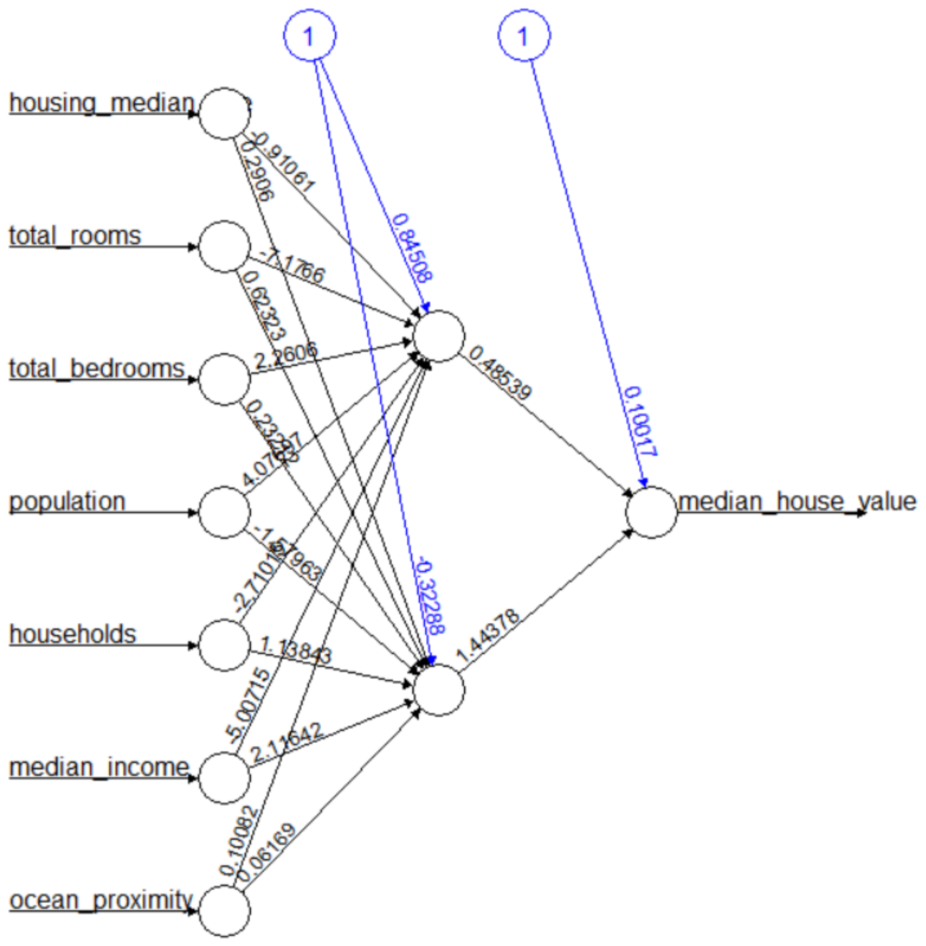
```
"within 20%: 0.4709"
```

```
#PLOTTING ACTUAL AND PREDICTED VALUES FOR TESTING DATA
x<- 1:length(test.y)
plot(x, test.y, type="l", lwd=2, col="magenta", main="ANN Regression with Logistic Activation
Function", panel.first=grid())
lines(x, pred.y, lwd=2, col="dodgerblue")
points(x,test.y, pch=16, col="magenta")
points(x, pred.y, pch=16, col="dodgerblue")
legend("topright", c("actual", "predicted"), lty=1, lwd=2, col=c("magenta","dodgerblue"))
```



#FITTING ANN WITH TANH ACTIVATION FUNCTION ann.reg<- neuralnet(median_house_value ~ housing_median_age+total_rooms+total_bedrooms+population +households+median_income +ocean_proximity, data=train, hidden=2, act.fct="tanh")

#PLOTTING THE DIAGRAM
plot(ann.reg)



```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.y<- predict(ann.reg, test.x)

#accuracy within 10%
accuracy10<- ifelse(abs(test.y-pred.y)<0.10*test.y,1,0)

#accuracy within 15%
accuracy15<- ifelse(abs(test.y-pred.y)<0.15*test.y,1,0)

#accuracy within 20%
accuracy20<- ifelse(abs(test.y-pred.y)<0.20*test.y,1,0)

print('Prediction Accuracy')
print(paste('within 10%:', round(mean(accuracy10),4)))
```

```
 "within 10%: 0.2383 "
```

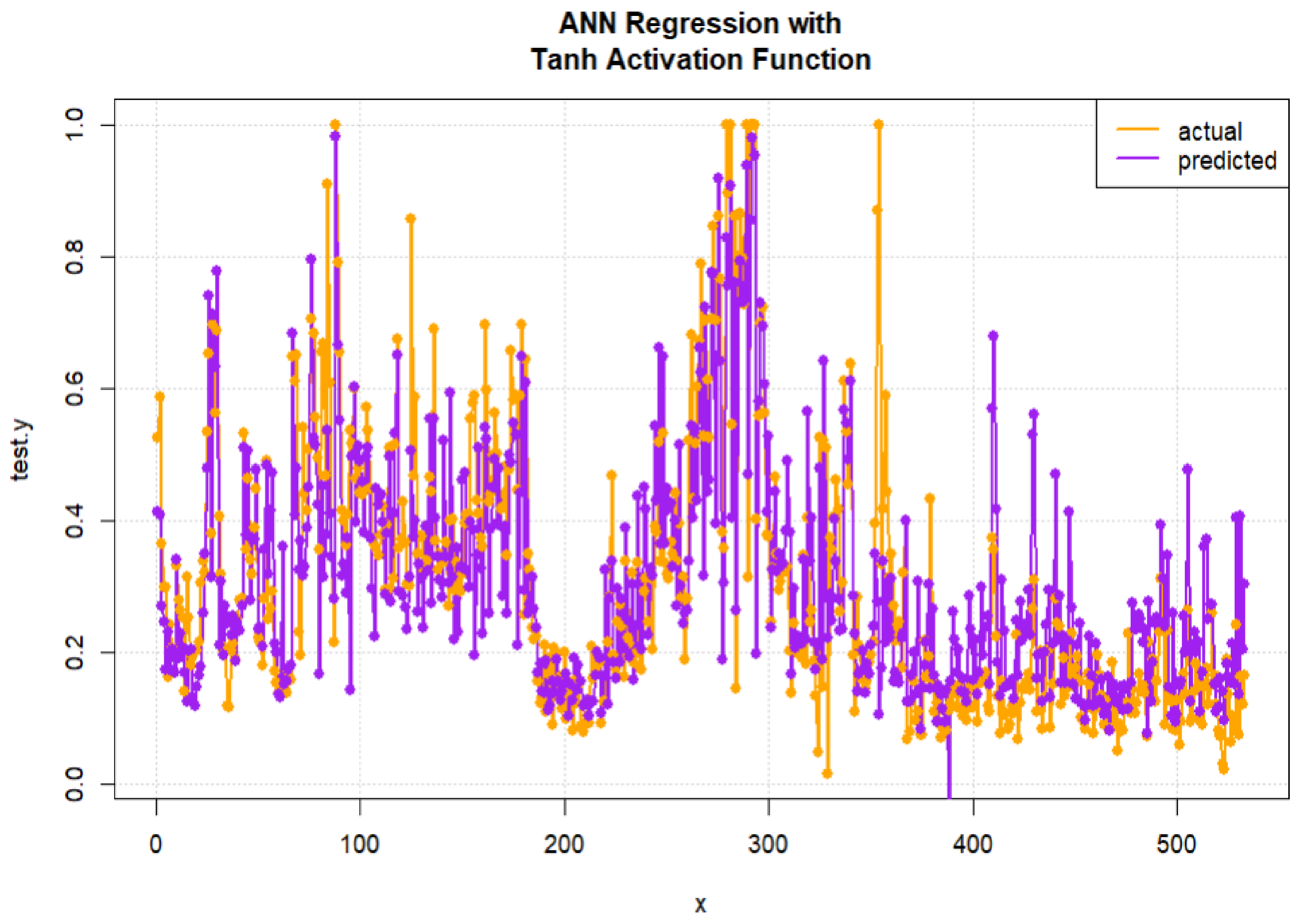
```
print(paste('within 15%:', round(mean(accuracy15),4)))
```

```
"within 15%: 0.3265"
```

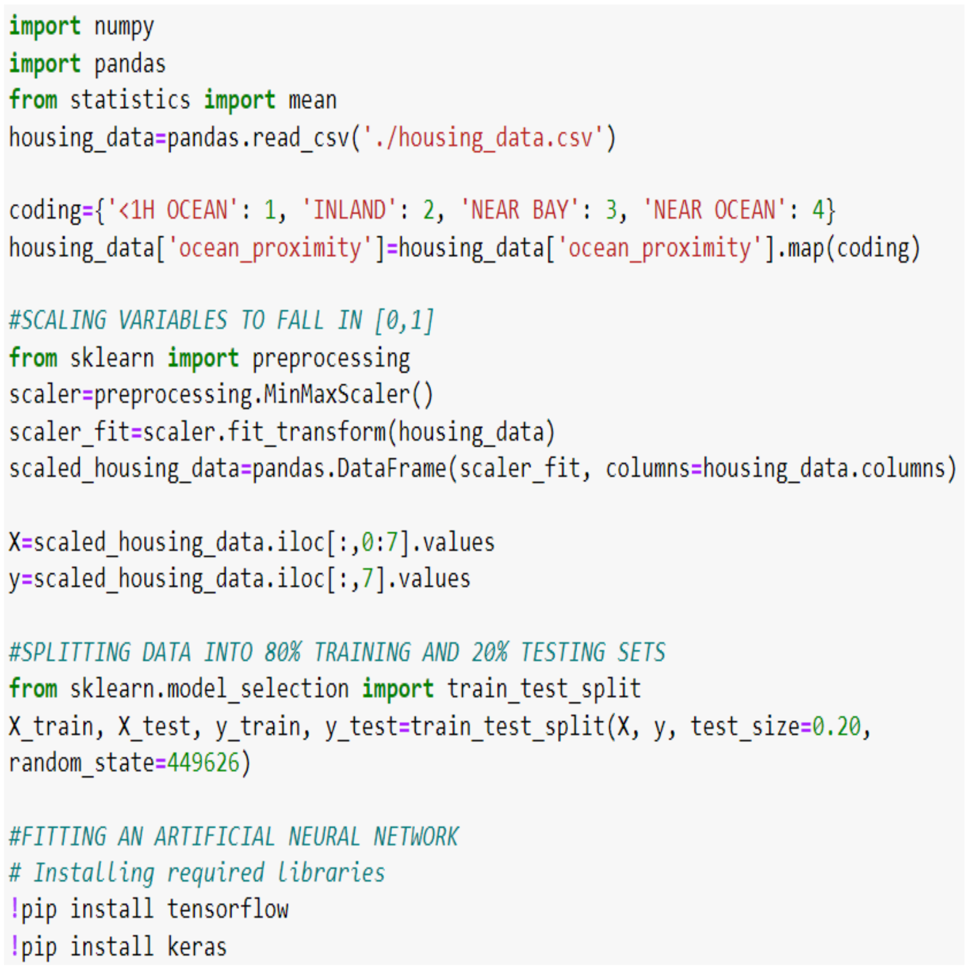
```
print(paste('within 20%:', round(mean(accuracy20),4)))
```

```
"within 20%: 0.454"
```

```
#PLOTTING ACTUAL AND PREDICTED VALUES FOR TESTING DATA
x<- 1:length(test.y)
plot(x, test.y, type="l", lwd=2, col="orange", main="ANN Regression with Tanh Activation Function", panel.first=grid())
lines(x, pred.y, lwd=2, col="purple")
points(x,test.y, pch=16, col="orange")
points(x, pred.y, pch=16, col="purple")
legend("topright", c("actual", "predicted"), lty=1, lwd=2, col=c("orange","purple"))
```



In Python: We will an ANN with tanh and sigmoid (logistic) activation functions.

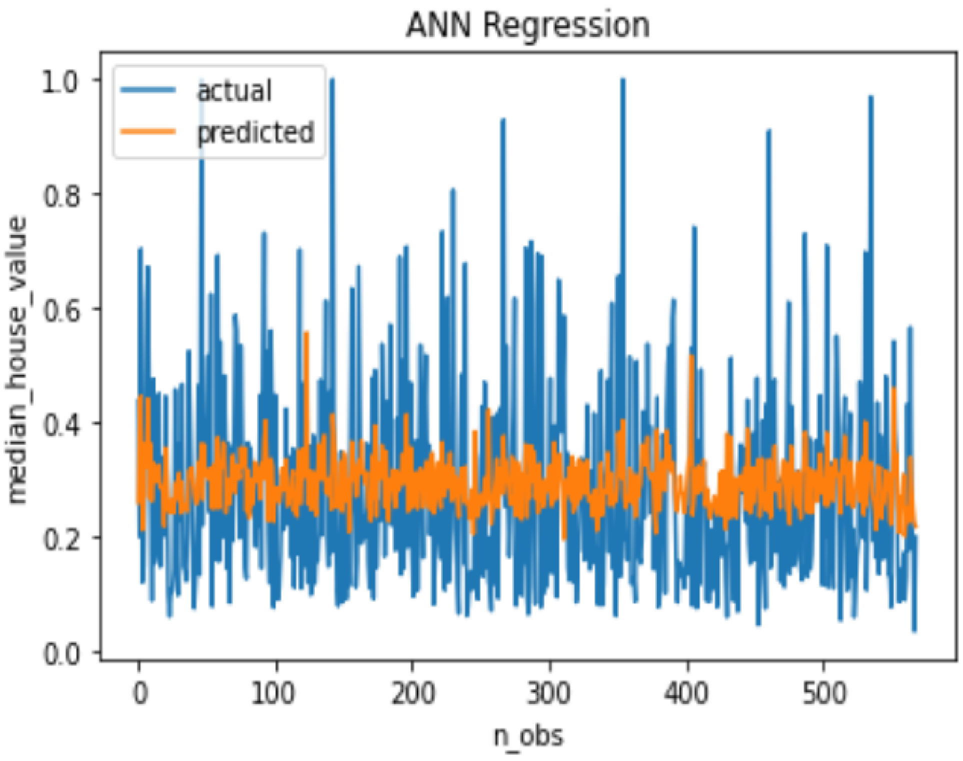


```python
import numpy
import pandas
from statistics import mean
housing_data=pandas.read_csv('./housing_data.csv')

coding={'<1H OCEAN': 1, 'INLAND': 2, 'NEAR BAY': 3, 'NEAR OCEAN': 4}
housing_data['ocean_proximity']=housing_data['ocean_proximity'].map(coding)

#SCALING VARIABLES TO FALL IN [0,1]
from sklearn import preprocessing
scaler=preprocessing.MinMaxScaler()
scaler_fit=scaler.fit_transform(housing_data)
scaled_housing_data=pandas.DataFrame(scaler_fit, columns=housing_data.columns)

X=scaled_housing_data.iloc[:,0:7].values
y=scaled_housing_data.iloc[:,7].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=449626)

#FITTING AN ARTIFICIAL NEURAL NETWORK
# Installing required libraries
!pip install tensorflow
!pip install keras
```

```
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()

#Defining the input layer and one hidden layer
model.add(Dense(units=3, input_dim=7, kernel_initializer='uniform',
activation='tanh'))

#Defining the output neuron
model.add(Dense(1))

#Compiling the model
model.compile(loss='mean_squared_error')

#Fitting the ANN to the training set
model.fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=model.predict(X_test)

ind10=[]
ind15=[]
ind20=[]

for sub1, sub2 in zip(y_pred, y_test):
    ind10.append(1) if abs(sub1-sub2)<0.10*sub2 else ind10.append(0)
    ind15.append(1) if abs(sub1-sub2)<0.15*sub2 else ind15.append(0)
    ind20.append(1) if abs(sub1-sub2)<0.20*sub2 else ind20.append(0)
```

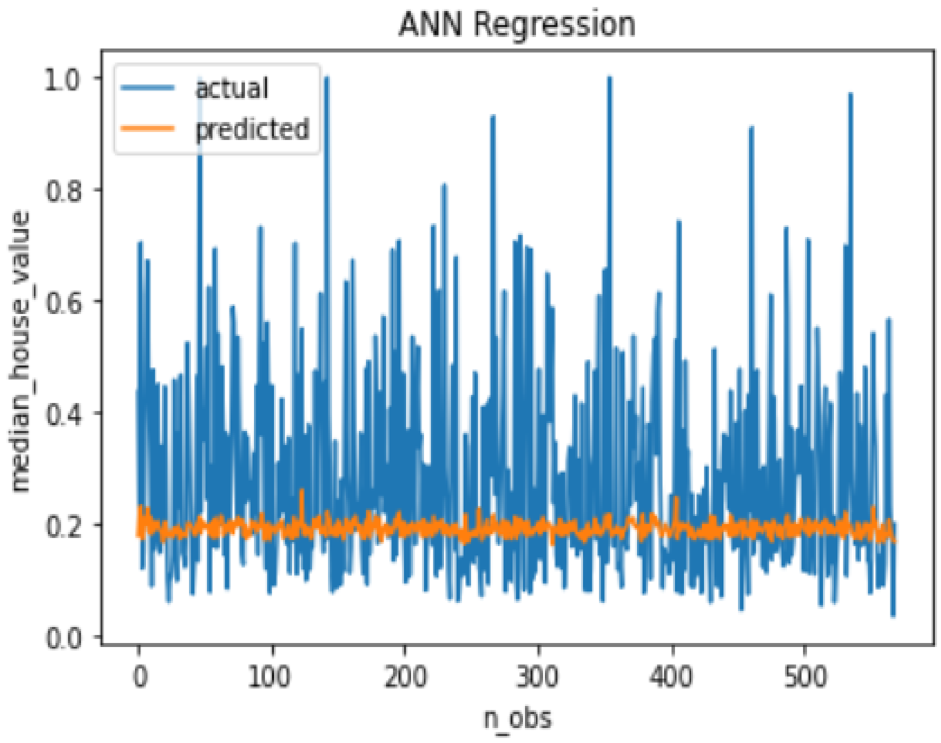
```python
#accuracy within 10%
accuracy10=mean(ind10)
print('accuracy within 10% =', round(accuracy10,4))

#accuracy within 15%
accuracy15=mean(ind15)
print('accuracy within 15% =', round(accuracy15,4))

#accuracy within 20%
accuracy20=mean(ind20)
print('accuracy within 20% =', round(accuracy20,4))


#plotting actual and predicted obsevations vs. observation number
import matplotlib.pyplot as plt

n_obs=list(range(0,len(y_test)))
plt.plot(n_obs, y_test, label="actual")
plt.plot(n_obs, y_pred, label="predicted")
plt.xlabel('n_obs')
plt.ylabel('median_house_value')
plt.title('ANN Regression')
plt.legend()
plt.show()
```

```
accuracy within 10% = 0.1301
accuracy within 15% = 0.2144
accuracy within 20% = 0.2865
```



```
model.add(Dense(units=3, input_dim=7, kernel_initializer='uniform',
activation='sigmoid'))
```

```
accuracy within 10% = 0.0879
accuracy within 15% = 0.1424
accuracy within 20% = 0.2162
```



□

ANN Binary Classifier

For an ANN binary classifier, the loss function is the average cross-entropy across all data points

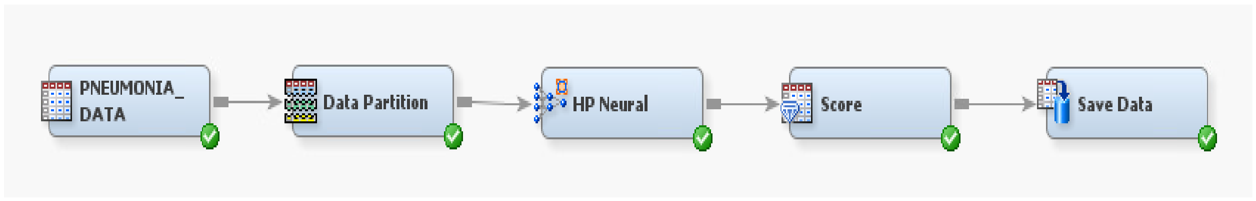
$$L = \frac{1}{n}\sum_{i=1}^{n}\left[y_i \ln \hat{y}_i + (1-y_i)\ln(1-\hat{y}_i)\right]$$

$$= \frac{1}{n}\sum_{i=1}^{n}\Big[y_i \ln\big(f(\mathbf{W}_\mathbf{i}^*\,\mathbf{h}_\mathbf{i} + b_i^*)\big) + (1-y_i)\ln\big(1 - f(\mathbf{W}_\mathbf{i}^*\,\mathbf{h}_\mathbf{i} + b_i^*)\big)\Big].$$

Example. For the pneumonia data, we fit an ANN.

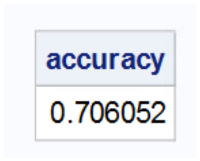
In SAS:

In Enterprise Miner we run the following diagram:



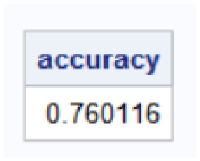
Note that the scale for the target variable must be specified as "nominal". Then we run the following code to compute the accuracy for the model with the logistic activation function:

```
data accuracy;
set tmp1.em_save_test;
match=(em_classification=em_classtarget);
run;

proc sql;
select mean(match) as accuracy
from accuracy;
quit;
```



accuracy
0.706052

and that for the model with the tanh activation function:

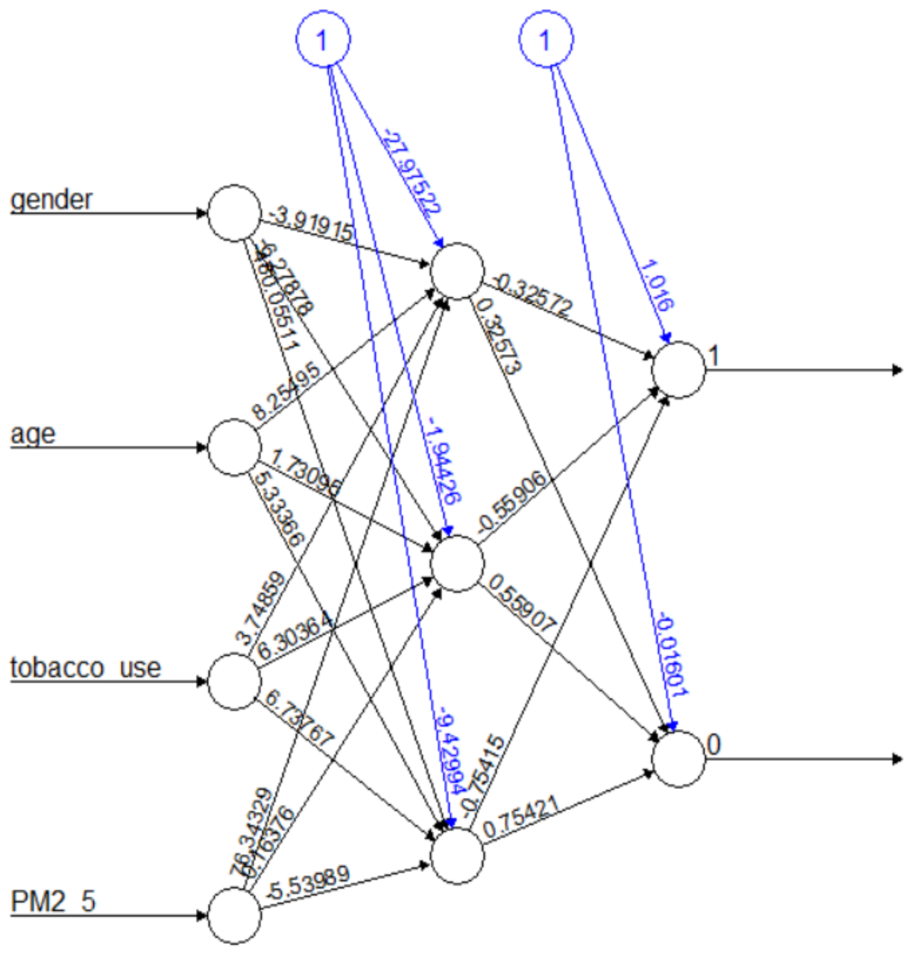


accuracy
0.760116

Note that the model with the tanh activation function has a higher prediction accuracy.

In R:

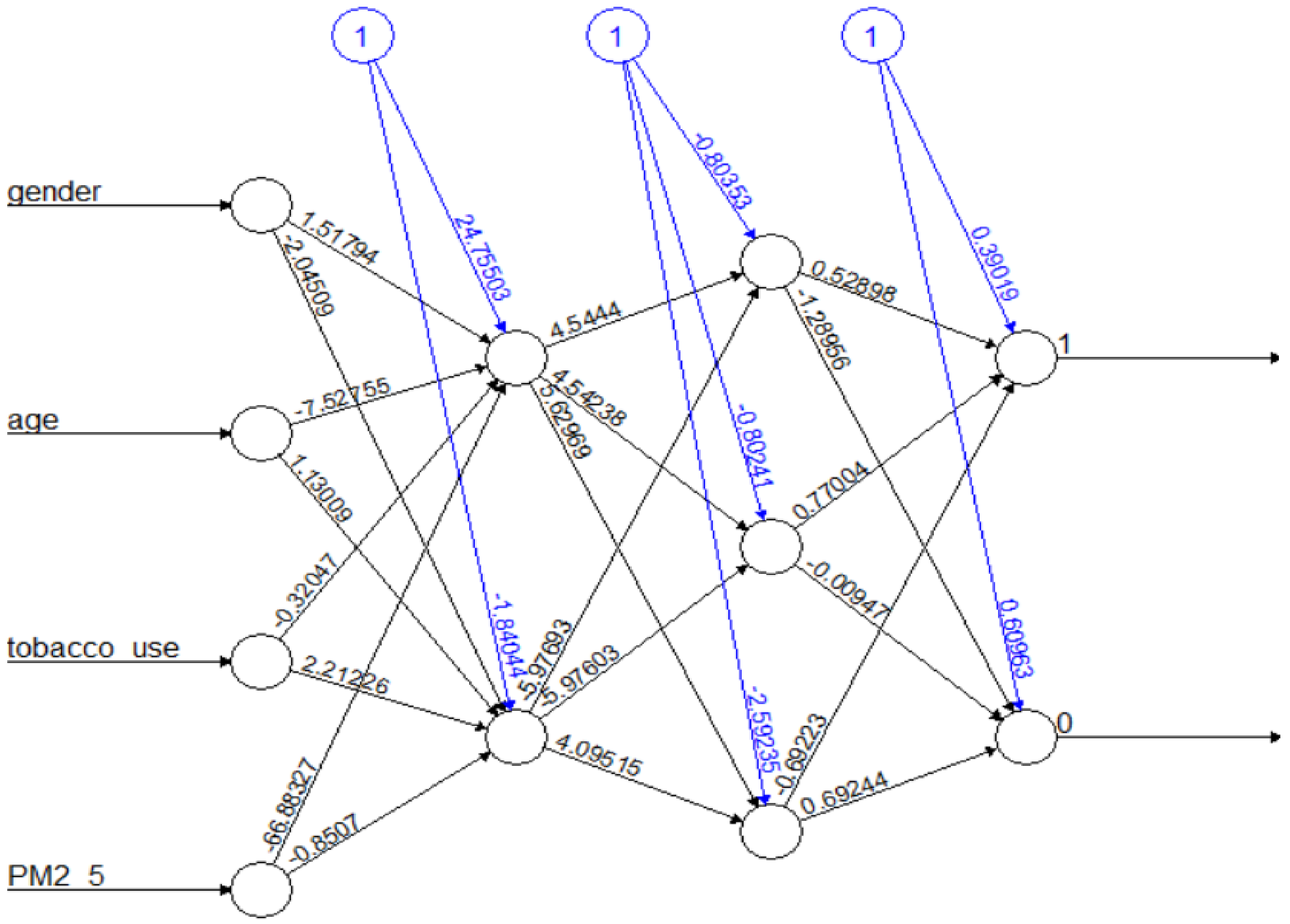
```
pneumonia.data<- read.csv(file="./pneumonia_data.csv", header=TRUE, sep=",")

pneumonia.data$pneumonia<- ifelse(pneumonia.data$pneumonia=="yes",1,0)
pneumonia.data$gender<- ifelse(pneumonia.data$gender=='M',1,0)
pneumonia.data$tobacco_use<- ifelse(pneumonia.data$tobacco_use=='yes',1,0)

#SCALING VARIABLES TO FALL IN [0,1]
library(dplyr)

scale01 <- function(x){
    (x-min(x))/(max(x)-min(x))
}

pneumonia.data<- pneumonia.data %>% mutate_all(scale01)

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(503548)
sample <- sample(c(TRUE, FALSE), nrow(pneumonia.data), replace=TRUE, prob=c(0.8,0.2))
train<- pneumonia.data[sample,]
test<- pneumonia.data[!sample,]

train.x<- data.matrix(train[-5])
train.y<- data.matrix(train[5])
test.x<- data.matrix(test[-5])
test.y<- data.matrix(test[5])

library(neuralnet)

#FITTING ANN WITH LOGISTIC ACTIVATION FUNCTION AND ONE LAYER WITH THREE
NEURONS
ann.class<- neuralnet(as.factor(pneumonia) ~ gender + age + tobacco_use + PM2_5, data=train,
hidden=3, act.fct="logistic")

#PLOTTING THE DIAGRAM
plot(ann.class)
```



```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.prob<- predict(ann.class, test.x)[,1]

match<- c()
pred.y<- c()
for (i in 1:length(test.y)){
    pred.y[i]<- ifelse(pred.prob[i]>0.5,1,0)
    match[i]<- ifelse(test.y[i]==pred.y[i],1,0)
}

print(paste("accuracy=", round(mean(match), digits=4)))
```

```
"accuracy= 0.2747"
```

```
#FITTING ANN WITH LOGISTIC ACTIVATION FUNCTION AND C(2,3) LAYERS
ann.class<- neuralnet(as.factor(pneumonia) ~ gender + age + tobacco_use + PM2_5, data=train,
hidden=c(2,3), act.fct="logistic")

#PLOTTING THE DIAGRAM
plot(ann.class)
```

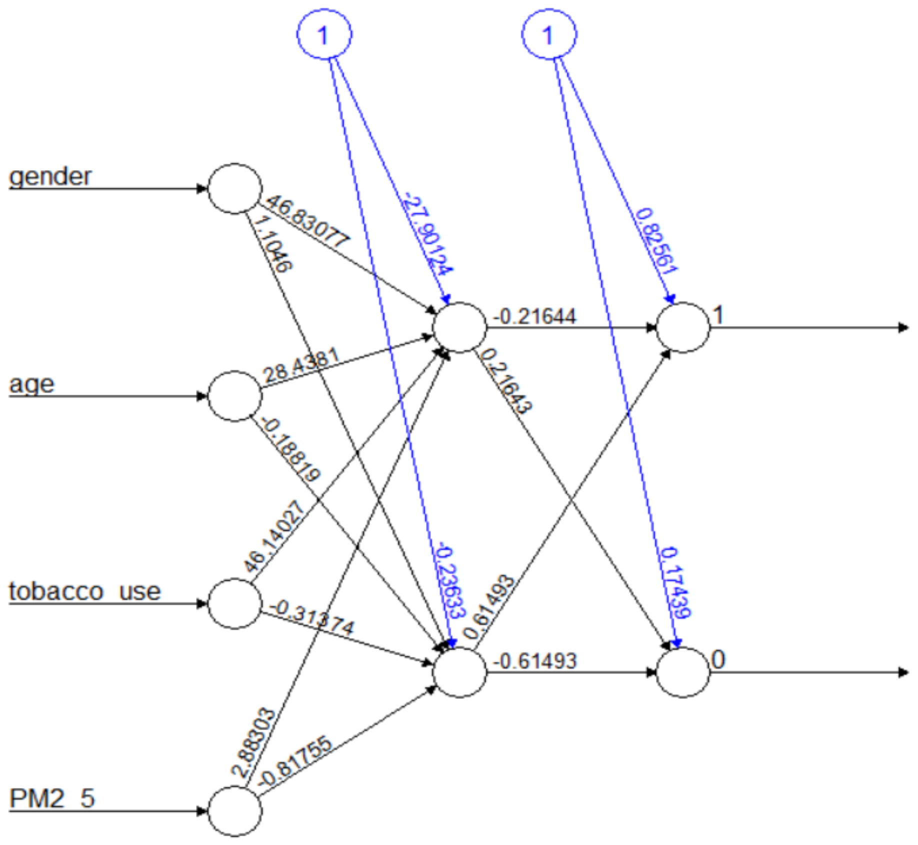


```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.prob<- predict(ann.class, test.x)[,1]

match<- c()
pred.y<- c()
for (i in 1:length(test.y)){
    pred.y[i]<- ifelse(pred.prob[i]>0.5,1,0)
    match[i]<- ifelse(test.y[i]==pred.y[i],1,0)
}
```

```
print(paste("accuracy=", round(mean(match), digits=4)))
```

```
"accuracy= 0.2623"
```

```
#FITTING ANN WITH TANH ACTIVATION FUNCTION
ann.class<- neuralnet(as.factor(pneumonia) ~ gender + age + tobacco_use + PM2_5, data=train,
hidden=2, act.fct="tanh")

#PLOTTING THE DIAGRAM
plot(ann.class)
```



```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.prob<- predict(ann.class, test.x)[,1]

match<- c()
pred.y<- c()
for (i in 1:length(test.y)){
    pred.y[i]<- ifelse(pred.prob[i]>0.5,1,0)
    match[i]<- ifelse(test.y[i]==pred.y[i],1,0)
}

print(paste("accuracy=", round(mean(match), digits=4)))
```

```
"accuracy= 0.3025"
```

In Python:

```
import numpy
import pandas
from sklearn.model_selection import train_test_split

pneumonia_data=pandas.read_csv('./pneumonia_data.csv')
code_gender={'M':1,'F':0}
code_tobacco_use={'yes':1,'no':0}
code_pneumonia={'yes':1,'no':0}

pneumonia_data['gender']=pneumonia_data['gender'].map(code_gender)
pneumonia_data['tobacco_use']=pneumonia_data['tobacco_use'].map(code_tobacco_use)
pneumonia_data['pneumonia']=pneumonia_data['pneumonia'].map(code_pneumonia)
y=pneumonia_data['pneumonia'].values

#SCALING VARIABLES TO FALL IN [0,1]
from sklearn import preprocessing
scaler=preprocessing.MinMaxScaler()
scaler_fit=scaler.fit_transform(pneumonia_data)
scaled_pneumonia_data=pandas.DataFrame(scaler_fit, columns=pneumonia_data.columns)

X=scaled_pneumonia_data.iloc[:,0:4].values
y=scaled_pneumonia_data.iloc[:,4].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20,
random_state=505606)

#FITTING AN ARTIFICIAL NEURAL NETWORK
import keras
from keras.models import Sequential
from keras.layers import Dense
```

```
biclassifier=Sequential()

#Defining the input layer and first hidden layer
biclassifier.add(Dense(units=3, activation='sigmoid'))

#Defining the output neuron
biclassifier.add(Dense(1))

#Compiling the model
biclassifier.compile(loss='binary_crossentropy')

#Fitting the ANN to the training set
biclassifier.fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
y_pred=numpy.round(biclassifier.predict(X_test),0) #predicted probability of 1

from sklearn import metrics
print('Accuracy:', round(metrics.accuracy_score(y_test, y_pred)*100, 2),'%')
```

```
Accuracy: 67.92 %
```

□

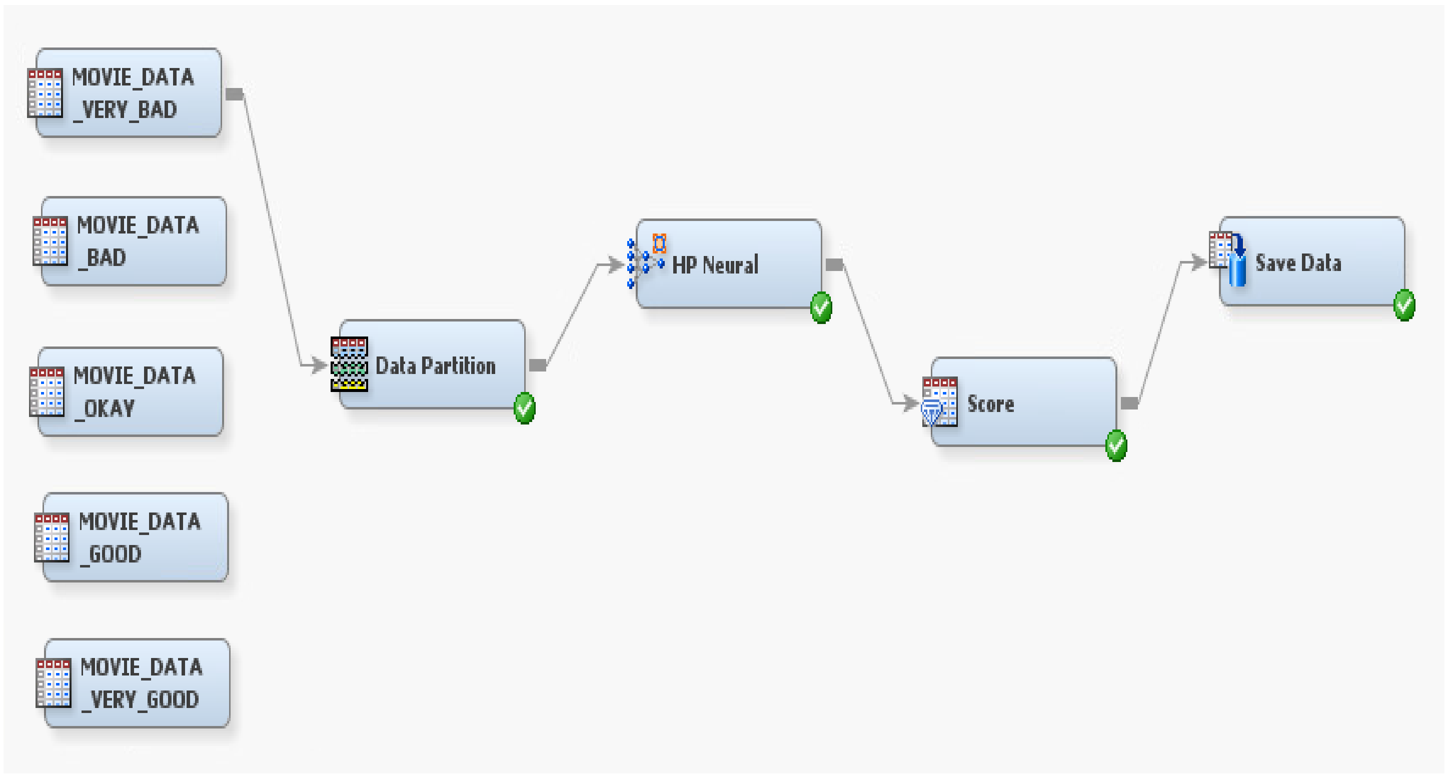
ANN Multinomial Classifier

The loss function used in multinomial classification is the multinomial cross-entropy function defined as $E = -\sum_{i=1}^{k} \left[p_i \ln p_i \right]$ where p_i is the proportion of observations in class i, $i = 1, ..., k$.

Example. We fit an ANN to the movies data.

In SAS:

In Enterprise Miner we run the following diagram, with the logistic activation function for the model. We set the indicator variables to "Target" with "Nominal" scale, and to "Rejected" all the other indicators.

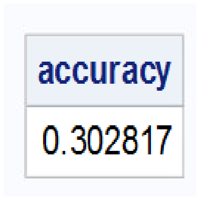


Then we run the following code to compute the accuracy.

```
data rating;
set sasuser.movies;
_dataobs_=_n_;
keep _dataobs_ rating;
run;

proc sort;
by _dataobs_;
run;

data very_bad;
set './em_save_test_verybad.sas7bdat';
predprob_verybad=em_eventprobability;
class_verybad=em_classtarget;
keep _dataobs_ class_verybad predprob_verybad;
run;

proc sort;
```

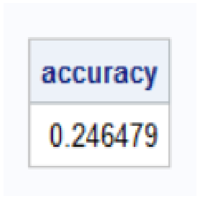
```
by _dataobs_;
run;

data bad;
set './em_save_test_bad.sas7bdat';
predprob_bad=em_eventprobability;
class_bad=em_classtarget;
keep _dataobs_ class_bad predprob_bad;
run;
proc sort;
by _dataobs_;
run;

data okay;
set './em_save_test_okay.sas7bdat';
predprob_okay=em_eventprobability;
class_okay=em_classtarget;
keep _dataobs_ class_okay predprob_okay;
run;

proc sort;
by _dataobs_;
run;

data good;
set './em_save_test_good.sas7bdat';
predprob_good=em_eventprobability;
class_good=em_classtarget;
keep _dataobs_ class_good predprob_good;
run;

proc sort;
by _dataobs_;
run;

data very_good;
set './em_save_test_verygood.sas7bdat';
predprob_verygood=em_eventprobability;
class_verygood=em_classtarget;
keep _dataobs_ class_verygood predprob_verygood;
run;
```

```
proc sort;
by _dataobs_;
run;

data all_data;
merge rating very_bad bad okay good very_good;
by _dataobs_;
if cmiss(predprob_verybad, predprob_bad,
predprob_okay, predprob_good, predprob_verygood)=0;
run;

data all_data;
set all_data;
predprob_max=max(predprob_very_bad, predprob_bad,
predprob_okay, predprob_good, predprob_very_good);
if (predprob_very_good=predprob_max) then pred_class='very good';
if (predprob_very_bad=predprob_max) then pred_class='very bad';
if (predprob_bad=predprob_max) then pred_class='bad';
if (predprob_okay=predprob_max) then pred_class='okay';
if (predprob_good=predprob_max) then pred_class='good';
keep rating pred_class;
run;

data all_data;
set all_data;
match=(rating=pred_class);
run;

proc sql;
select mean(match) as accuracy
from all_data;
quit;
```



accuracy
0.302817

Setting the activation function to the default value of tanh, we run the diagram and the SAS code again to output the accuracy of



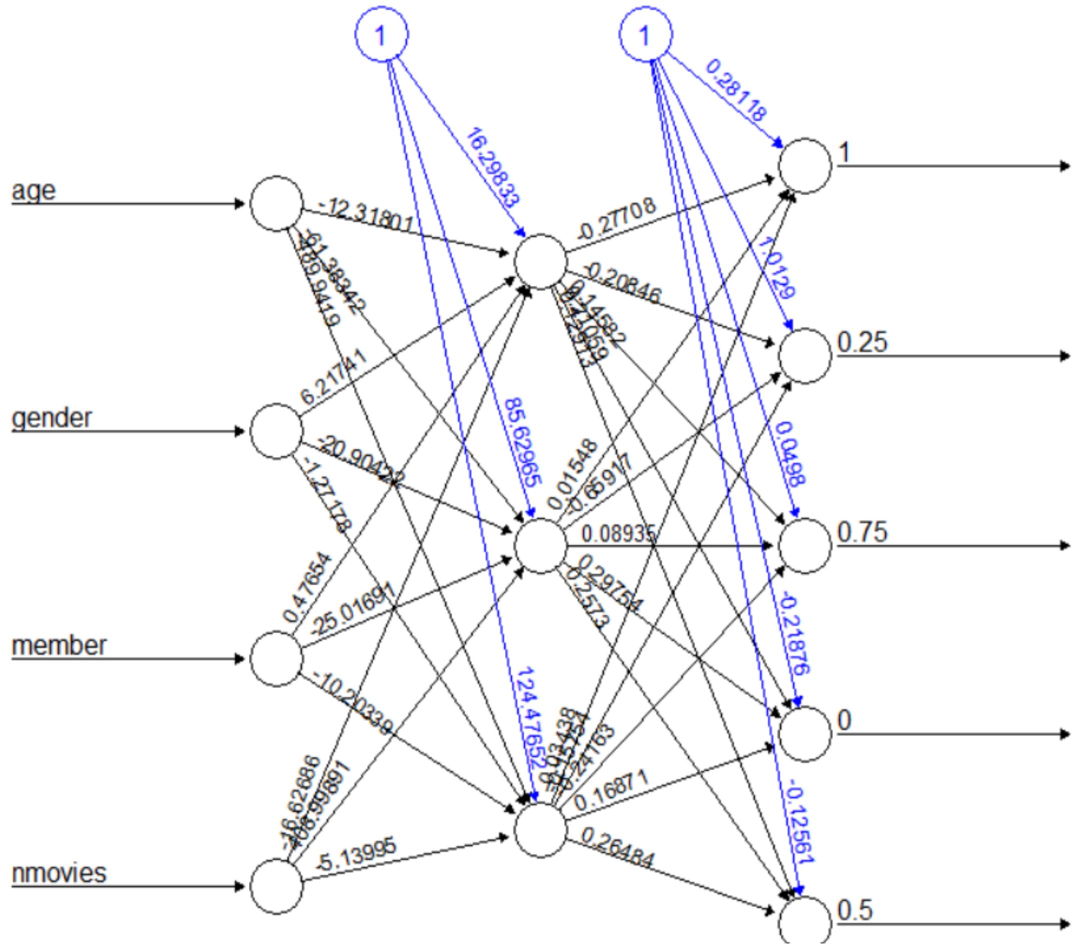
accuracy
0.246479

The model with the logistic activation function has a higher accuracy.

In R:

```
movie.data<- read.csv(file="./movie_data.csv", header=TRUE, sep=",")

movie.data$gender<- ifelse(movie.data$gender=='M',1,0)
movie.data$member<- ifelse(movie.data$member=='yes',1,0)

movie.data$rating<- ifelse(movie.data$rating=='very bad',1, ifelse(movie.data$rating=='bad',2,ifelse(movie.data
ifelse(movie.data$rating=='good',4,5))))

#SCALING VARIABLES TO FALL IN [0,1]
library(dplyr)

scale01 <- function(x){
   (x-min(x))/(max(x)-min(x))
}

movie.data<- movie.data %>% mutate_all(scale01)

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
set.seed(100001)
sample <- sample(c(TRUE, FALSE), nrow(movie.data), replace=TRUE, prob=c(0.8,0.2))
train<- movie.data[sample,]
test<- movie.data[!sample,]

train.x<- data.matrix(train[-5])
train.y<- data.matrix(train[5])
test.x<- data.matrix(test[-5])
test.y<- data.matrix(test[5])
```

```
#FITTING ANN WITH LOGISTIC ACTIVATION FUNCTION
library(neuralnet)
set.seed(4544446)
ann.mclass<- neuralnet(as.factor(rating) ~ age + gender + member + nmovies, data=train, hid-
den=3, act.fct="logistic")

#PLOTTING THE DIAGRAM
plot(ann.mclass)
```

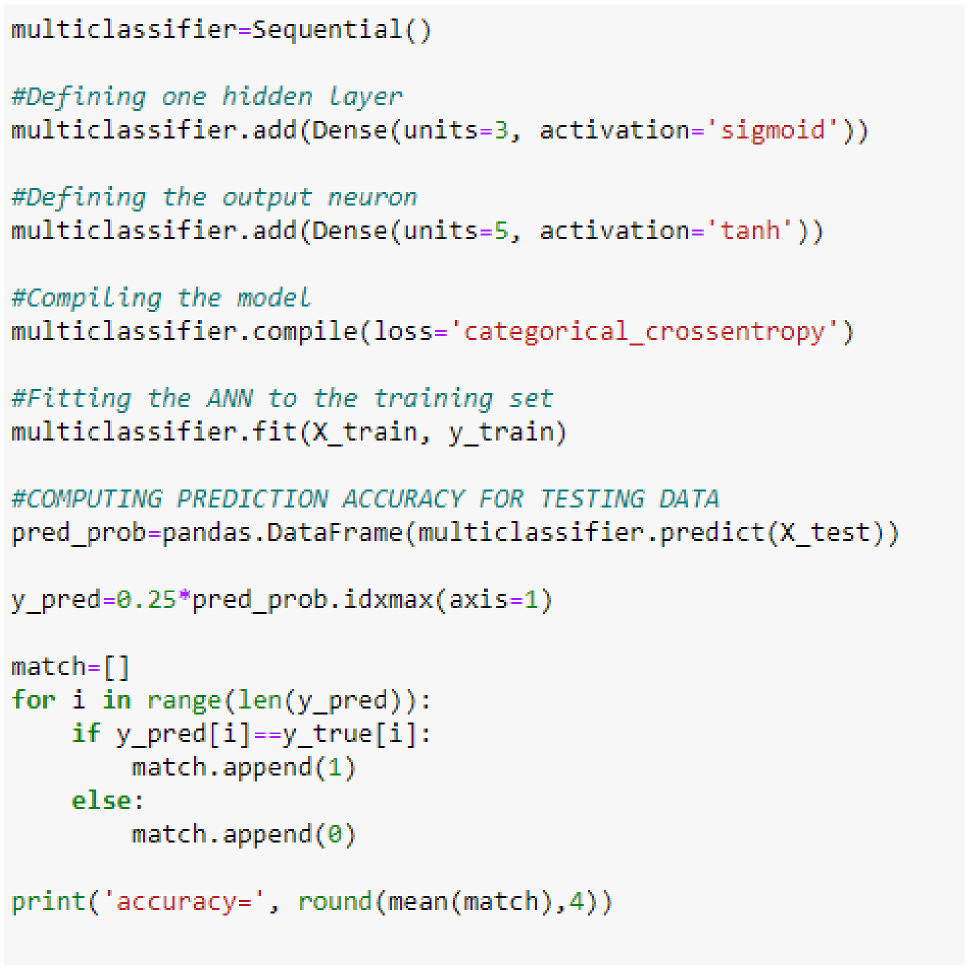


```
#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred.prob <- predict(ann.mclass, test.x)
pred.prob<- as.data.frame(pred.prob)

colnames(pred.prob)<- c(0, 0.25, 0.5, 0.75, 1)
```

```
pred.class<- apply(pred.prob, 1, function(x) colnames(pred.prob)[which.max(x)])

match<- c()
for (i in 1:length(test.y)) {
    match[i]<- ifelse(pred.class[i]==as.character(test.y[i]),1,0)
}

print(accuracy<- mean(match))
0.3611111
```

In Python:

```
import numpy
import pandas
from sklearn.model_selection import train_test_split
from statistics import mean

movie_data=pandas.read_csv('./movie_data_ind.csv')

code_gender={'M':1,'F':0}
code_member={'yes':1,'no':0}
code_rating={'very bad':1,'bad':2,'okay':3,'good':4,'very good':5}

movie_data['gender']=movie_data['gender'].map(code_gender)
movie_data['member']=movie_data['member'].map(code_member)
movie_data['rating']=movie_data['rating'].map(code_rating)

#SCALING VARIABLES TO FALL IN [0,1]
from sklearn import preprocessing
scaler=preprocessing.MinMaxScaler()
scaler_fit=scaler.fit_transform(movie_data)
scaled_movie_data=pandas.DataFrame(scaler_fit, columns=movie_data.columns)

X=scaled_movie_data.iloc[:,0:4].values
y=scaled_movie_data.iloc[:,4:10].values

#SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.20, random_state=116008)

y_train=y_train[:,1:6]
y_true=y_test[:,0]
y_test=y_test[:,1:6]

#FITTING AN ARTIFICIAL NEURAL NETWORK
import keras
from keras.models import Sequential
from keras.layers import Dense
import tensorflow
tensorflow.random.set_seed(454545)
```



```python
multiclassifier=Sequential()

#Defining one hidden layer
multiclassifier.add(Dense(units=3, activation='sigmoid'))

#Defining the output neuron
multiclassifier.add(Dense(units=5, activation='tanh'))

#Compiling the model
multiclassifier.compile(loss='categorical_crossentropy')

#Fitting the ANN to the training set
multiclassifier.fit(X_train, y_train)

#COMPUTING PREDICTION ACCURACY FOR TESTING DATA
pred_prob=pandas.DataFrame(multiclassifier.predict(X_test))

y_pred=0.25*pred_prob.idxmax(axis=1)

match=[]
for i in range(len(y_pred)):
    if y_pred[i]==y_true[i]:
        match.append(1)
    else:
        match.append(0)

print('accuracy=', round(mean(match),4))
```

```
accuracy= 0.3158
```

□