Car Evaluation

02/March/2017

1 Group members

Name	Roll number	email-id
Shivam Porwal	1611CS14	shivam.mtcs16@iitp.ac.in
Debanjan Sarkar	1611CS17	debanjan.mtcs16@iitp.ac.in

2 Abstract of the project

Car Evaluation Dataset evaluate the target concept(CAR) with 3 other intermediate concept. PRICE(overall price),TECH(technical characteristic) and COMFORT(comfort). Now totally we have 6 attribute, each attribute is a part of one of the intermediate concept as described above. These attributes are :
1. Buying(buying price)
2. Maint(price of maintenance)
3. Doors(number of doors)
4. Persons(capacity in terms of persons to carry)
5. *Lug_Boot*(the size of luggage bot)
6. Safety(estimated safety of the car)

The number of the instances in the training data are 1728 , and there are 6 number of attributes as mentioned above. This is basically a multi-class classification problem. we will classify the instance into 4 classes:

Here are the attribute values:

Table 1: **Attribute values**

buying	{v-high, high, med, low}
maint	{v-high, high, med, low}
doors	{2, 3, 4, 5- more}
persons	{2, 4, more}
lug_boot	{small, med, big}
safety	{low, med, high}

Table 2: **Class Distribution (number of instance per class)**

unacc	1210
acc	384
good	69
v-good	65

3 Introduction

To implement such type of model we will use the concept of Convolution Neural Network.To implement it we will use the python library "keras" in which we are using backend as Theano. To find the highest

accuracy we will consider various scenarios or cases which are described in the upcoming topics.

3.1 Data sources

https://archive.ics.uci.edu/ml/datasets/Car+Evaluation

3.2 Literature survey

The following tests were performed by Tijana Jovanovic, Faculty of Organisation Sciences, University of Belgrade.
In this example they will be using 80% of data for training the network and 20% of data for testing it.
Training attempt 1:

- Network Type: Multi Layer Perceptron
- Training Algorithm: Backpropagation with Momentum
- Number of inputs: 21
- Number of outputs: 4 (unacc,acc,good,vgood)
- Hidden neurons: 14

Training Parameters:

- Learning Rate: 0.2
- Momentum: 0.7
- Max. Error: 0.01

Training Results:
For this training, they used Sigmoid transfer function.

Following results were generated in the First Attempt

	Network Inputs						Real Outputs			
Instance number	Buying	Maint	Doors	Persons	Lug boot	Safety	Unacc	Acc	Good	VGood
1.	0,0,0,1(vhigh)	0,0,0,1(vhigh)	1,0,0,0(2)	1,0,0(2)	0,1,0,(med)	0,1,0(med)	1	0	0	0
2.	1,0,0,0 (low)	1,0,0,0 (low)	0,0,0,1 (5more)	0,1,0 (4)	0,0,1 (big)	0,0,1 (high)	0	0	0	1
3.	1,0,0,0 (low)	1,0,0,0 (low)	0,0,0,1 (5more)	0,1,0 (4)	1,0,0 (small)	1,0,0 (low)	1	0	0	0
4.	1,0,0,0 (low)	1,0,0,0 (low)	0,0,0,1 (5more)	0,0,1 (more)	1,0,0 (small)	0,1,0 (med)	0	1	0	0
5.	1,0,0,0 (low)	0,1,0,0 (med)	0,0,0,1 (5more)	0,1,0 (4)	0,0,1 (big)	0,1,0 (med)	0	0	1	0

The **output neural network produced** for this input is, respectively:

	Network Inputs						Outputs neural network produced			
Instance number	Buying	Maint	Doors	Persons	Lug boot	Safety	Unacc	Acc	Good	VGood
1.	0,0,0,1(vhigh)	0,0,0,1(vhigh)	1,0,0,0(2)	1,0,0(2)	0,1,0,(med)	0,1,0(med)	1	0	0	0
2.	1,0,0,0 (low)	1,0,0,0 (low)	0,0,0,1 (5more)	0,1,0 (4)	0,0,1 (big)	0,0,1 (high)	0,0009	0,0002	0,0053	0,9931
3.	1,0,0,0 (low)	1,0,0,0 (low)	0,0,0,1 (5more)	0,1,0 (4)	1,0,0 (small)	1,0,0 (low)	1	0	0,0001	0
4.	1,0,0,0 (low)	1,0,0,0 (low)	0,0,0,1 (5more)	0,0,1 (more)	1,0,0 (small)	0,1,0 (med)	0,0033	0,9965	0,0025	0
5.	1,0,0,0 (low)	0,1,0,0 (med)	0,0,0,1 (5more)	0,1,0 (4)	0,0,1 (big)	0,1,0 (med)	0,0002	0,0006	0,9973	0,0016

Training attempt 2:

- Network Type: Multi Layer Perceptron
- Training Algorithm: Backpropagation with Momentum

- Number of inputs: 21
- Number of outputs: 4 (unacc,acc,good,vgood)
- Hidden neurons: 14

Training Parameters:

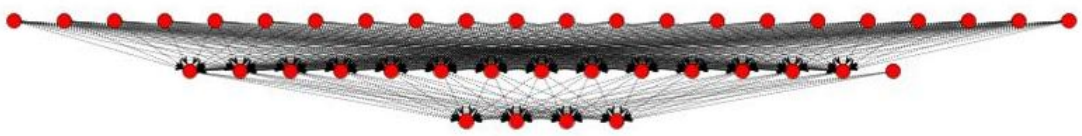
- Learning Rate: 0.3
- Momentum: 0.6
- Max. Error: 0.01

Training Results:
For this training, they used Sigmoid transfer function.

Following results were generated in the First Attempt

	Network Inputs						Outputs neural network produced			
Instance number	Buying	Maint	Doors	Persons	Lug boot	Safety	Unacc	Acc	Good	VGood
1.	0,0,0,1(vhigh)	0,0,0,1(vhigh)	1,0,0,0(2)	1,0,0(2)	0,1,0,(med)	0,1,0(med)	1	0	0	0
2.	1,0,0,0 (low)	1,0,0,0 (low)	0,0,0,1 (5more)	0,1,0 (4)	0,0,1 (big)	0,0,1 (high)	0	0	0	0,9996
3.	1,0,0,0 (low)	1,0,0,0 (low)	0,0,0,1 (5more)	0,1,0 (4)	1,0,0 (small)	1,0,0 (low)	1	0	0	0
4.	1,0,0,0 (low)	1,0,0,0 (low)	0,0,0,1 (5more)	0,0,1 (more)	1,0,0 (small)	0,1,0 (med)	0	1	0	0
5.	1,0,0,0 (low)	0,1,0,0 (med)	0,0,0,1 (5more)	0,1,0 (4)	0,0,1 (big)	0,1,0 (med)	0	0	1	0

3.3 Network Architecture :



fig(1) : Convolution Neural Network

3.4 Our Results:

Case 1:

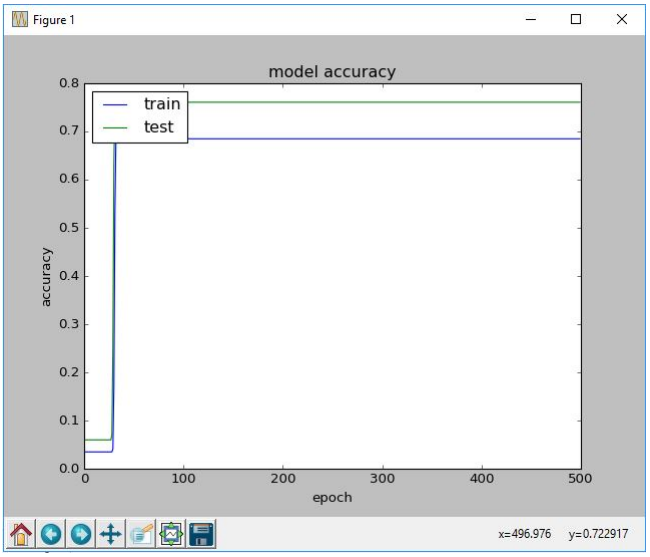
- Input Neuron: 21
- Number of Hidden Layers: 1
- Hidden Neurons : 14
- Output Neurons : 4
- Activation function : Sigmoid function (both in hidden layer and output layer)
- Learning rate : 0.1
- Momentum : 0.7
- Number of Iterations : 500

Result of Case 1:

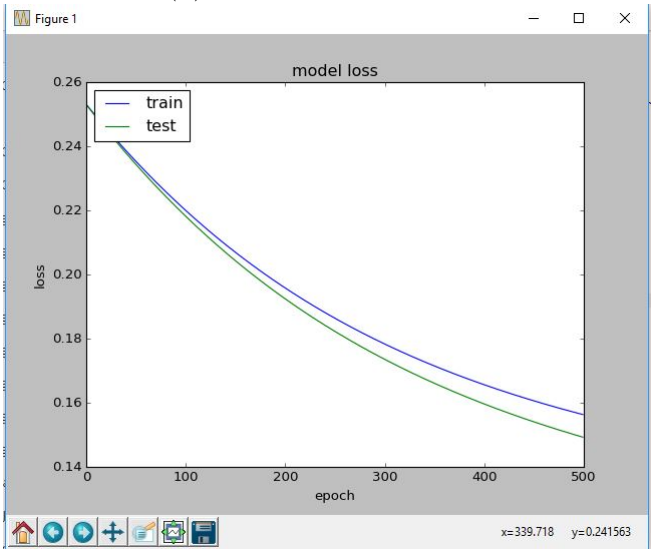
```
Epoch 497/500
1377/1377 [==============================] - 0s - loss: 0.1566 - acc: 0.6848 - val_loss: 0.1495 - val_acc: 0.7607
Epoch 498/500
1377/1377 [==============================] - 0s - loss: 0.1565 - acc: 0.6848 - val_loss: 0.1494 - val_acc: 0.7607
Epoch 499/500
1377/1377 [==============================] - 0s - loss: 0.1564 - acc: 0.6848 - val_loss: 0.1494 - val_acc: 0.7607
Epoch 500/500
1377/1377 [==============================] - 0s - loss: 0.1563 - acc: 0.6848 - val_loss: 0.1493 - val_acc: 0.7607
('mean squared error :', 0.14926735564344629)
('PREDICTED', array([[ 0.58279097,  0.37154564,  0.29080477,  0.28780967],
       [ 0.5833267 ,  0.37154239,  0.29027137,  0.28745881],
       [ 0.5828824 ,  0.37183481,  0.29136008,  0.28830126],
       ...,
       [ 0.58359236,  0.37079629,  0.29020908,  0.28789011],
       [ 0.58321053,  0.37072924,  0.29023898,  0.28795239],
       [ 0.58374566,  0.37072596,  0.28970632,  0.28760129]], dtype=float32))
('ORIGINAL', array([[1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       ...,
       [1, 0, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1]]))
```

fig(a): Output of Case1

Graphical Representation:



fig(b): Accuracy plot of Case1



fig(c): Error plot of Case1

Case 2:

- Input Neuron: 21
- Number of Hidden Layers: 2
- Hidden Neurons at Layer 1: 14

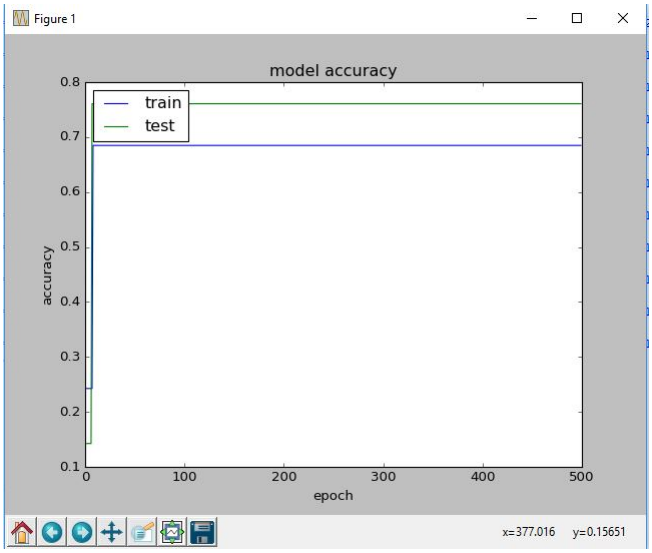
- Hidden Neurons at Layer2 : 7
- Output Neurons : 4
- Activation function : Sigmoid function (both in hidden layer and output layer)
- Learning rate : 0.1
- Momentum : 0.7
- Number of Iterations : 500

Result of Case 2:

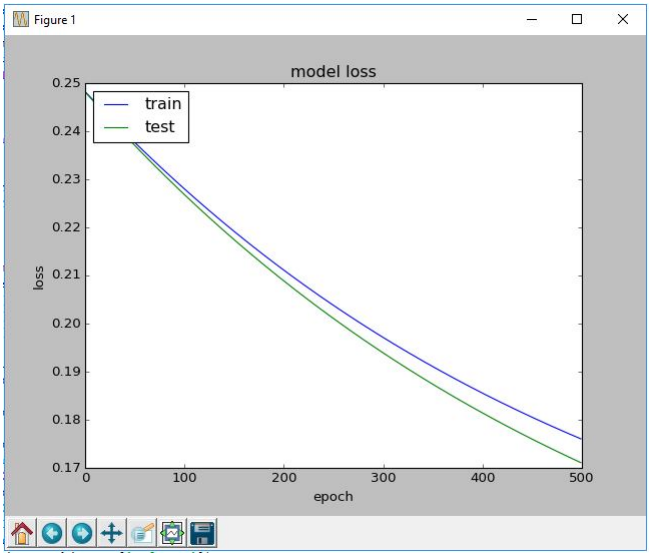
```
Epoch 497/500
1377/1377 [==============================] - 1s - loss: 0.1763 - acc: 0.6848 - val_loss: 0.1714 - val_acc: 0.7607
Epoch 498/500
1377/1377 [==============================] - 1s - loss: 0.1762 - acc: 0.6848 - val_loss: 0.1713 - val_acc: 0.7607
Epoch 499/500
1377/1377 [==============================] - 1s - loss: 0.1761 - acc: 0.6848 - val_loss: 0.1712 - val_acc: 0.7607
Epoch 500/500
1377/1377 [==============================] - 1s - loss: 0.1760 - acc: 0.6848 - val_loss: 0.1711 - val_acc: 0.7607
('mean squared error :', 0.17107827783140361)
('PREDICTED', array([[ 0.56394356,  0.41390133,  0.3456955 ,  0.34379134],
       [ 0.56394118,  0.41389966,  0.34568802,  0.3437905 ],
       [ 0.56394327,  0.4139027 ,  0.34569395,  0.34379551],
       ...,
       [ 0.56394738,  0.41388243,  0.34565979,  0.34375069],
       [ 0.56395102,  0.41387793,  0.34565696,  0.34374544],
       [ 0.56394863,  0.4138763 ,  0.34564945,  0.34374461]], dtype=float32))
('ORIGINAL', array([[1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       ...,
       [1, 0, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1]]))
```

fig(d): Output of Case2

Graphical Representation:



fig(e): Accuracy plot of Case2



fig(f): Error plot of Case2

Case 3:

- Input Neuron: 21
- Number of Hidden Layers: 3
- Hidden Neurons at Layer 1: 500
- Hidden Neurons at Layer2 : 400
- Hidden Neurons at Layer 3 : 300
- Output Neurons : 4
- Activation function : Sigmoid function (both in hidden layer and output layer)
- Learning rate : 0.1
- Momentum : 0.7
- Number of Iterations : 20

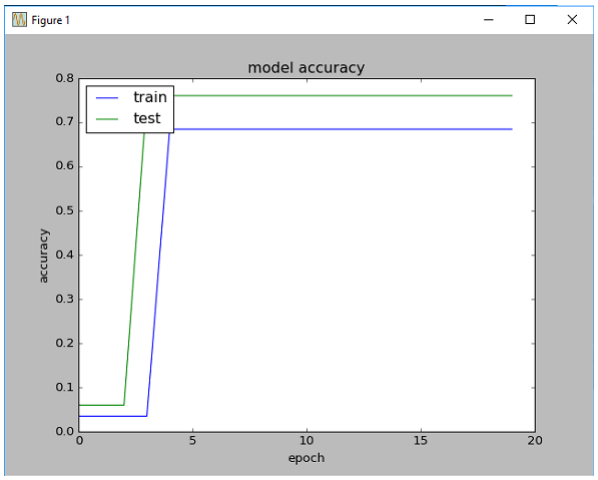
Result of Case 3:

```
1377/1377 [==============================] - 37s - loss: 0.2230 - acc: 0.6848 - val_loss: 0.2124 - val_acc: 0.7607
Epoch 10/20
1377/1377 [==============================] - 38s - loss: 0.2162 - acc: 0.6848 - val_loss: 0.2058 - val_acc: 0.7607
Epoch 11/20
1377/1377 [==============================] - 37s - loss: 0.2099 - acc: 0.6848 - val_loss: 0.1996 - val_acc: 0.7607
Epoch 12/20
1377/1377 [==============================] - 36s - loss: 0.2040 - acc: 0.6848 - val_loss: 0.1939 - val_acc: 0.7607
Epoch 13/20
1377/1377 [==============================] - 36s - loss: 0.1986 - acc: 0.6848 - val_loss: 0.1886 - val_acc: 0.7607
Epoch 14/20
1377/1377 [==============================] - 37s - loss: 0.1936 - acc: 0.6848 - val_loss: 0.1836 - val_acc: 0.7607
Epoch 15/20
1377/1377 [==============================] - 36s - loss: 0.1890 - acc: 0.6848 - val_loss: 0.1791 - val_acc: 0.7607
Epoch 16/20
1377/1377 [==============================] - 37s - loss: 0.1848 - acc: 0.6848 - val_loss: 0.1749 - val_acc: 0.7607
Epoch 17/20
1377/1377 [==============================] - 37s - loss: 0.1808 - acc: 0.6848 - val_loss: 0.1710 - val_acc: 0.7607
Epoch 18/20
1377/1377 [==============================] - 37s - loss: 0.1772 - acc: 0.6848 - val_loss: 0.1673 - val_acc: 0.7607
Epoch 19/20
1377/1377 [==============================] - 37s - loss: 0.1738 - acc: 0.6848 - val_loss: 0.1640 - val_acc: 0.7607
Epoch 20/20
1377/1377 [==============================] - 37s - loss: 0.1707 - acc: 0.6848 - val_loss: 0.1608 - val_acc: 0.7607
```

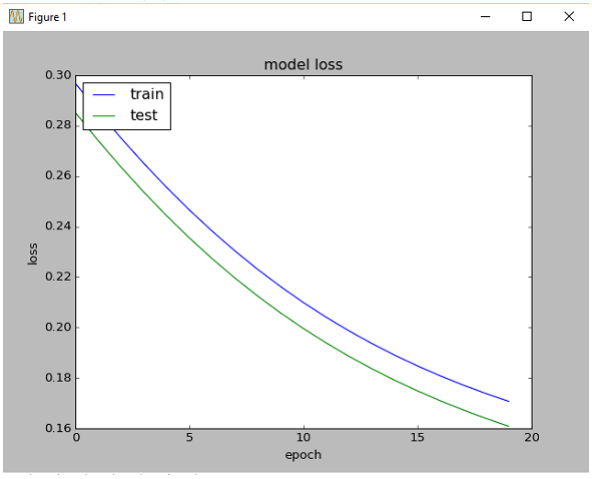
```
('mean squared error :', 0.16084803144137064)
('PREDICTED', array([[ 0.61050826,  0.40708581,  0.34769079,  0.30529341],
       [ 0.61058283,  0.40731785,  0.34763551,  0.30501154],
       [ 0.61047441,  0.40688914,  0.34723935,  0.30520773],
       ...,
       [ 0.61059737,  0.40751094,  0.34674102,  0.3050831 ],
       [ 0.61060041,  0.40754312,  0.34709203,  0.30536228],
       [ 0.61067557,  0.40777513,  0.34703651,  0.30507955]], dtype=float32))
('ORIGINAL', array([[1, 0, 0, 0],
       [1, 0, 0, 0],
       [1, 0, 0, 0],
       ...,
       [1, 0, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1]]))
```

fig(g): Output of Case3

Graphical Representation:



fig(h): Accuracy plot of Case3



fig(i): Error plot of Case3

Case 4:

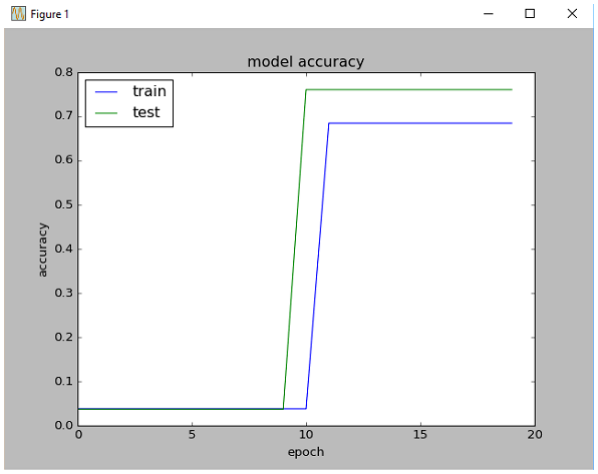
- Input Neuron: 21
- Number of Hidden Layers: 4
- Hidden Neurons at Layer 1: 500
- Hidden Neurons at Layer2 : 400
- Hidden Neurons at Layer 3 : 300
- Hidden Neurons at Layer 4 : 200
- Output Neurons : 4
- Activation function : Sigmoid function (both in hidden layer and output layer)
- Learning rate : 0.1
- Momentum : 0.7
- Number of Iterations : 20

Result of Case 4:

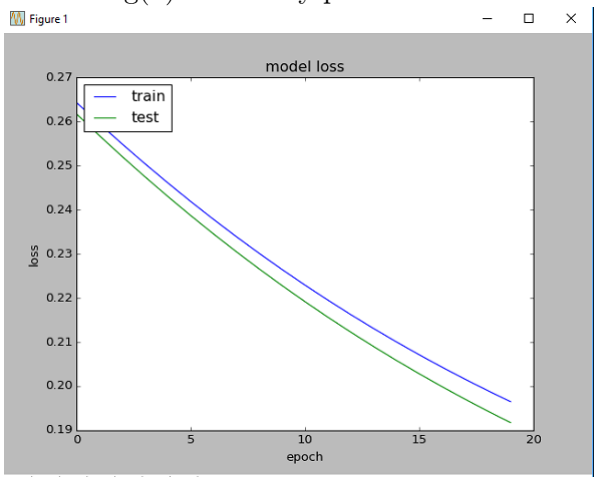
```
Epoch 11/20
1377/1377 [==============================] - 58s - loss: 0.2229 - acc: 0.0378 - val_loss: 0.2192 - val_acc: 0.7607
Epoch 12/20
1377/1377 [==============================] - 59s - loss: 0.2195 - acc: 0.6848 - val_loss: 0.2157 - val_acc: 0.7607
Epoch 13/20
1377/1377 [==============================]  - 58s - loss: 0.2162 - acc: 0.6848 - val_loss: 0.2123 - val_acc: 0.7607
Epoch 14/20
1377/1377 [==============================]  - 60s - loss: 0.2131 - acc: 0.6848 - val_loss: 0.2090 - val_acc: 0.7607
Epoch 15/20
1377/1377 [==============================] - 58s - loss: 0.2101 - acc: 0.6848 - val_loss: 0.2059 - val_acc: 0.7607
Epoch 16/20
1377/1377 [==============================] - 58s - loss: 0.2071 - acc: 0.6848 - val_loss: 0.2028 - val_acc: 0.7607
Epoch 17/20
1377/1377 [==============================] - 58s - loss: 0.2043 - acc: 0.6848 - val_loss: 0.1999 - val_acc: 0.7607
Epoch 18/20
1377/1377 [==============================] - 59s - loss: 0.2016 - acc: 0.6848 - val_loss: 0.1971 - val_acc: 0.7607
Epoch 19/20
1377/1377 [==============================] - 58s - loss: 0.1990 - acc: 0.6848 - val_loss: 0.1944 - val_acc: 0.7607
Epoch 20/20
1377/1377 [==============================] - 58s - loss: 0.1965 - acc: 0.6848 - val_loss: 0.1918 - val_acc: 0.7607
('mean squared error :', 0.19175669389572578)
('PREDICTED', array([[ 0.50512058,  0.42121321,  0.35301095,  0.413773  ],
       [ 0.5051195 ,  0.42121616,  0.35300726,  0.41376883],
       [ 0.5051198 ,  0.42120591,  0.35301316,  0.41377863],
       ...,
       [ 0.50511622,  0.42120776,  0.3530184 ,  0.41377339],
       [ 0.50511646,  0.42121091,  0.35301864,  0.41377029],
       [ 0.50511533,  0.42121387,  0.35301504,  0.41376612]], dtype=float32))
('ORIGINAL', array([[1, 0, 0, 0],
       [1, 0, 0, 0],
       ...,
       [1, 0, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1]]))
```

fig(j): output of Case4

Graphical Representation:



fig(k): Accuracy plot of Case4



fig(l): Error plot of Case4

3.5 Implementation :

The implementation of following models can be found on following link :
https://github.com/shiv8989/carevaluation

4 Future work :

In this program we are training the model based on some hidden layer. if the size of hidden layer is increased then model can be trained more accurately but complexity of the network may increase. Due to which it is also possible for reduction in accuracy. To solve this problem we can add more data, due to increase in data-size. Model can be trained more accurately and accuracy will increase. Due to the limited amount of data the accuracy can not cross the certain threshold. If we train the network by large number of data items performance will increase.

References

[1] Knowledge acquisition and explanation for multi-attribute decision making by M Bohanec, V Rajkovic

[2] Machine Learning by Function Decomposition Blaz Zupan, Marko Bohanec, Ivan Bratko, Janez Demsar

[3] https://archive.ics.uci.edu/ml/datasets/Car+Evaluation

[4] http://neuroph.sourceforge.net/tutorials/car$_e$*valuation*1/*car*$_e$*valuation*1.*html*