Homework 04

Deepak Sharma
CSCI 630-01 Foundations of Intelligent Systems

February 28, 2018

1

Suppose we generate a training set from a decision tree and then apply decision-tree learning to that training set. Will the learning algorithm eventually produce the original decision tree used to generate the sample, if the training set size approaches infinity? Why or why not?
Solution:
Since all possible samples of the dataset were available while constructing two decision trees, hence both decision trees know the true distribution of the data and always give same output for any valid input sample. But these two decision trees generated using same dataset can have different configuration they will be logically equivalent .
A Decision tree perform sequentially multiple logical operations. Here parameters of these operation are values of attributes in the dataset. Since for a input sample the end result may be indepenent of the order of applicable operations, hence, each decision tree with different configuration might be executing such possible logically equivalent sequence of operations for an input sample. In the end each decision tree logically equivalent set of operations and produced the same results. Though, decision tree constructor algorithm use entropy heuristic while deciding the next attribute, but in large dataset with high number of attributes, two attributes can provide same information gain after expansion. Hence the algorithm can pick different attributes for expansion during the two independent executions.
For example, we have a dataset with 2 attributes: grades in FIS and grade in algorithms. All students who work in DPRL lab have scored A grade in both courses. Now one decision tree may first check grade in Algorithm course and then in FIS course for deciding whether he/she will get a chance to work in the lab or not. On the other hand, second decision tree many first check algorithm course's grade and then FIS grade for deciding whether the applicant will get a chance or not.

2

Below is a data set for a binary class i cation problem (classes 0 and 1), for which each input has three binary attributes (with values inf0,1g). Each row in the table represents a training instance of the form ((A1;A2;A3),y), where (A1;A2;A3) are the inputs to the decision tree classier, and y is the correct output (i.e. the output for the 'true' function the learned tree

should represent). Use the decision tree learning algorithm (see text, p. 702, or the lecture slides) to learn a decision tree using Information Gain to select the best attribute from available attributes(i.e. attributes that are unused along the current path from the root to the current node in the tree at each recursive call).Show the computations that determine how attributes are selected for the tree.

Solution:

Sample	A1	A2	A3	y
x1	1	0	0	0
x2	1	0	1	0
x3	0	1	0	0
x4	1	1	1	1
x5	1	1	0	1

Table 1: data table

LEVEL=1

Let entropy of the random binary variable y for the provided data in table 2 is $\mathbf{E(A)}$.
Let $\mathbf{n}$ and $\mathbf{p}$ represent the number of samples with $\mathbf{0}$ label and $\mathbf{1}$ label respectively.
Given $\mathbf{n = 3}$, $\mathbf{p = 2}$
Let:$\mathbf{k = p/(n + p) = 2/5}$
Using the definition of entropy for a binary distribution [3]:

$$\mathbf{E(A) = B(k) = -1 * ((k) \log_2(k) + (1 - k) \log_2(k))}$$

$$\mathbf{E(A) = B(2/5) = -1 * ((2/5) \log_2(2/5) + (3/5) \log_2(3/5)) = 0.972}$$

CASE 1

If we choose attribute $\mathbf{A1}$ for expansion of the decision tree then
For distribution of $\mathbf{A1 = 1}$, $\mathbf{k = p_{A1=1}/(nA1 = 1 + pA1 = 1) = 2/4}$

$$\mathbf{E(A1 = 1) = B(1/2) = -1 * ((2/4) \log_2(2/4) + (2/4) \log_2(2/4)) = 1} \quad (1)$$

For the distribution of $\mathbf{A1 = 0}$, $\mathbf{k = p/(n + p)0}$

$$\mathbf{E(A1 = 0) = B(0) = -1 * ((0) \log_2(0) + (1) \log_2(1))}$$

$$\mathbf{B(0) = 0} \quad (2)$$

Total Entropy for the resultant distribution will be a weighted average of the entropy for $\mathbf{A1 = 1}$ and $\mathbf{A1 = 0}$ distribution:

$$\mathbf{E(A1) = ((n_{A1=1} + p_{A1=1})/(n + p)) * E(A1 = 1) + ((n_{A1=0} + p_{A1=0})/(n + p)) * E(A1 = 0)}$$

using equation 1 and 2:

$$\mathbf{E(A1) = (4/5) * E(A1 = 1) + (1/5) * E(A1 = 0) = (4/5) * 1 + (1/5) * 0 = .80}$$

information gain after the expansion $\mathbf{I_{A1}}$:

$$\mathbf{I_{A1} = E(A) - E(A1) = 0.972 - 0.80 = .172} \tag{3}$$

CASE 2
If we decide attribute **A2** for expansion of the decision tree then for the distribution of $\mathbf{A2 = 1}$, $\mathbf{k = p_{A2=1}/(p_{A2=1} + n_{A2=1}) = 2/3}$

$$\mathbf{E(A2 = 1) = B(2/3) = -1 * ((2/3)\log_2(2/3) + (1/3)\log_2(1/3)) = .918} \tag{4}$$

For distribution of $\mathbf{A2 = 0}$, $\mathbf{k = p_{A2=0}/(p_{A2=0} + n_{A2=0} = 0}$
Using equation 2:

$$\mathbf{E(A2 = 0) = B(0) = 0}$$

Total Entropy for the resultant distribution will be a weighted average of the entropy for $\mathbf{A2 = 1}$ and $\mathbf{A2 = 0}$ distribution:

$$\mathbf{E(A2) = ((n_{A2=1} + p_{A2=1})/(n + p)) * E(A2 = 1) + ((n_{A2=0} + p_{A2=0})/(n + p)) * E(A2 = 0)}$$

using equation 4 and 1:

$$\mathbf{E(A2) = (3/5) * E(A2 = 1) + (2/5) * E(A2 = 0) = .546}$$

Information gain after the expansion $\mathbf{I_{A2}}$:

$$\mathbf{I_{A2} = E(A) - E(A2) = 0.972 - 0.546 = .426} \tag{5}$$

CASE 3
If we decide attribute **A3** for expansion of the decision tree then for distribution of $\mathbf{A3 = 1}$:
$\mathbf{k = p_{A3=1}/(p_{A3=1} + n_{A3=1}) = 1/2}$
Using equation 1:

$$\mathbf{E(A3 = 1) = B(1/2) = 1}$$

For distribution of $\mathbf{A3 = 0}$: $\mathbf{k = 1/3}$
Using equation 4:

$$\mathbf{E(A3 = 0) = B(1/3) = B(2/3) = .918} \tag{6}$$

Total Entropy for the resultant distribution will be a weighted average of the entropy for $\mathbf{A3 = 1}$ and $\mathbf{A3 = 0}$ distribution:

$$\mathbf{E(A3) = ((n_{A3=1} + p_{A3=1})/(n + p)) * E(A3 = 1) + ((n_{A3=0} + p_{A3=0})/(n + p)) * E(A3 = 0)}$$

$$\mathbf{E(A3) = (2/5) * B(1/2) + (3/5) * B(1/3) = (2/5) * 1 + (3/5) * .918 = .946}$$

information gain after the expansion $\mathbf{I_{A3}}$:

$$\mathbf{I_{A3} = E(A) - E(A3) = 0.972 - 0.946 = .026} \tag{7}$$

Using equation 3, 5 and 7 we can say:

$$\mathbf{I_{A2} > I_{A1} > I_{A3}}$$

Sample	A1	A3	y
x3	0	0	0
x4	1	1	1
x5	1	0	1

Table 2: data distribution for A2=1

Sample	A1	A3	y
x1	1	0	0
x2	1	1	0

Table 3: data distribution for A2=0

Therefore, we will decide to expend decision tree with A2 attribute. In the resultant distribution has been shown in table 2 and 3 for A2=1 and A2=0 values respectively. Since the resultant distribution for A2=0 (table 3) belongs to only y=0, hence this distribution is a base case for decision tree algorithm and it can not be expended any further.

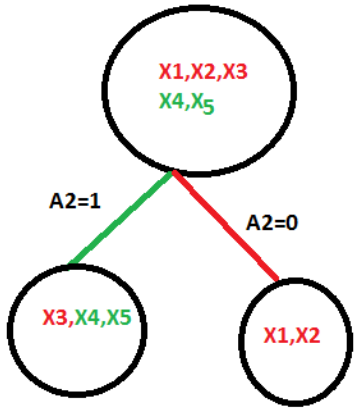

Figure 1: After expending the decision tree with attribute A2

LEVEL=2

For the data distribution of table 2 entropy is:

$$\mathbf{E(A) = B(2/3) = .91}$$

CASE=1

Expending the decision tree for the attribute A1:

For distribution of $\mathbf{A1 = 1}$: $\mathbf{k = (p_{A1=1}/(n_{A1=1} + p_{A1=1})) = 2/2}$

Using equation 2 :

$$\mathbf{E(A1 = 1) = B(1) = B(0) = 0}$$

For distribution of $\mathbf{A1 = 0}$: $\mathbf{k(p_{A1=0}/(n_{A1=0} + p_{A1=0})) = 0}$
using equation 2

$$\mathbf{E(A1 = 0) = B(0) = 0}$$

Total Entropy for the resultant distribution will be a weighted average of the entropy for $\mathbf{A1 = 1}$ and $\mathbf{A1 = 0}$ distribution:

$$\mathbf{E(A1) = ((n_{A1=1} + p_{A1=1})/(n + p)) * E(A1 = 1) + ((n_{A1=0} + p_{A1=0})/(n + p)) * E(A1 = 0)}$$

$$\mathbf{E(A1) = (2/3) * E(A1 = 1) + (1/3) * E(A1 = 0) = 0}$$

Information gain after the expansion $\mathbf{I_{A1}}$:

$$\mathbf{I_{A1} = E(A) - E(A1) = .91 - 0.0 = .91} \tag{8}$$

CASE 2
If we select to expend binary tree using A3 attribute then for distribution of $\mathbf{A3 = 1}$: $\mathbf{k = (p/(n + p)) = 1/1}$
Using equation 2 :

$$\mathbf{E(A3 = 1) = B(1) = 0}$$

For distribution of $\mathbf{A3 = 0}$: $\mathbf{k = p/(n + p) = 1/2}$
using equation 1

$$\mathbf{E(A3 = 0) = B(1/2) = 1}$$

Total Entropy for the resultant distribution will be a weighted average of the entropy for $\mathbf{A3 = 1}$ and $\mathbf{A3 = 0}$ distribution:

$$\mathbf{E(A1) = ((n_{A3=1} + p_{A3=1})/(n + p)) * E(A3 = 1) + ((n_{A3=0} + p_{A3=0})/(n + p)) * E(A3 = 0)}$$

$$\mathbf{E(A3) = (1/3) * E(A3 = 1) + (2/3) * E(A3 = 0) = 2/3}$$

Information gain after the expansion $\mathbf{I_{A1}}$:

$$\mathbf{I_{A3} = E(A) - E(A3).91 - 0.67 = .24} \tag{9}$$

Using equation 8 and 9 we can say:

$$\mathbf{I_{A1} > I_{A3}}$$

Hence we will select attribute A1 for the expansion of the binary tree.
After expansion, in the resultant distribution for A1=1, all the samples have value 1 for target variable y. Similarly, for A1=0 all the samples have value 0 for target variable y. Therefore we can not expend the tree any further as both these distribution are the base case of the algorithm.

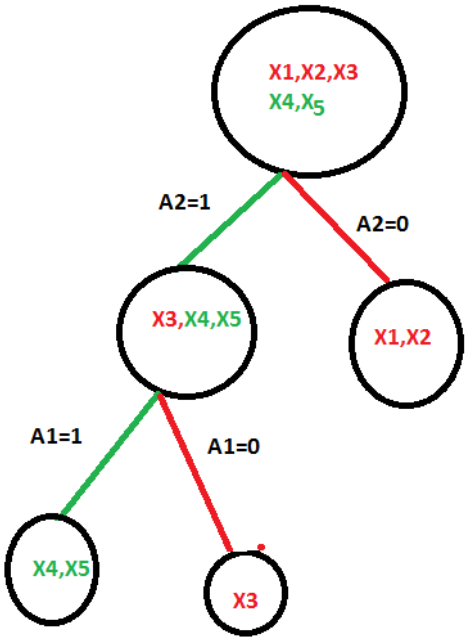

Figure 2: After expending the decision tree with attribute A2 and A1

3

Provide a neural network that computes the XOR function for two binary inputs (i.e.with values inf0,1g). Make sure to provide the explicit network architecture (including activation functions), and all weights in the network.Note: you should not train your network using a program to answer this question set the weights of your net-work manually.
Solution

Weight	Value	Weight	Value	Weight	Value
$\mathbf{W_{14}}$	1	$\mathbf{W_{15}}$	1	$\mathbf{W_{24}}$	1
$\mathbf{W_{25}}$	1	$\mathbf{W_{34}}$	-.5	$\mathbf{W_{35}}$	-1.5
$\mathbf{W_{47}}$	1	$\mathbf{W_{57}}$	-1	$\mathbf{W_{67}}$	0

Table 4: Weights for the Neural network executing XOR function

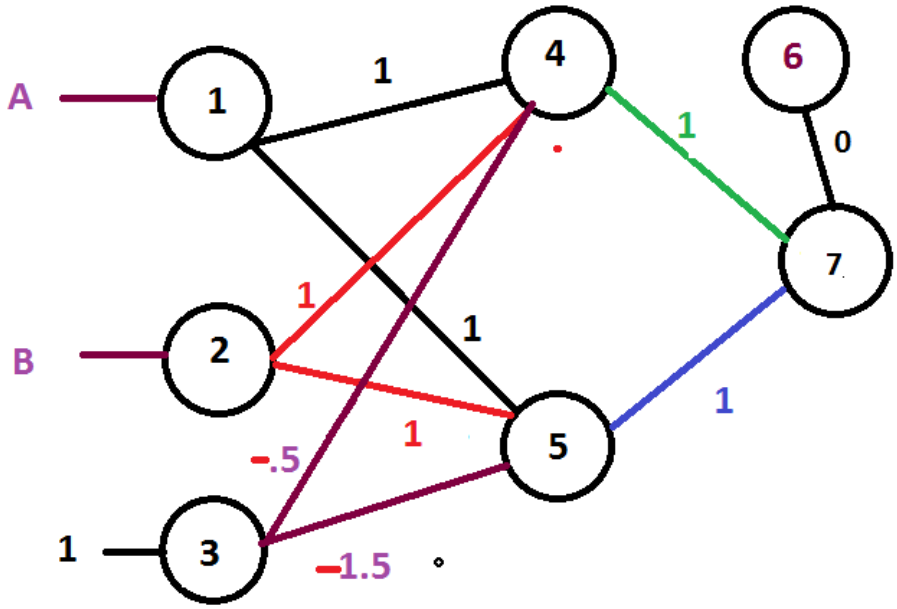

Figure 3: Neural Network with network weights for performing XOR operation. A and B are binary inputs where A and B $\epsilon\{0,1\}$

Activation function

$$g(X) = \begin{cases} 0, & \text{if } X < 0 \\ 1, & \text{otherwise} \end{cases} \tag{10}$$

Activation value of neuron, bias and input units:

$\mathbf{a_1}(input)$	$\mathbf{a_2}(input)$	$\mathbf{a_4}$	$\mathbf{a_5}$	$\mathbf{a_7}(output)$
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Table 5: Inputs and Activation of the neurons in neural network exhibiting XOR functionality

Calculation

For input $\mathbf{a_1}$=0,$\mathbf{a_2}$=0

$$\mathbf{a_4 = g(a_1 * W_{14} + a_2 * W_{24} + a_3 * W_{34}) = g(.1 + .1 + 1 * (-.5)) = g(-.5) = 0}$$

$$\mathbf{a_5 = g(a_1 * W_{15} + a_2 * W_{25} + a_3 * W_{35}) = g(0 * 1 + 0 * 1 + 1 * (-1.5)) = g(-1.5) = 0}$$

For input $\mathbf{a_1}$=0,$\mathbf{a_2}$=1

$$\mathbf{a_4 = g(1 * 1 + .1 + 1 * (-.5)) = g(.5) = 1}$$

$$\mathbf{a_5 = g(0 * 1 + 1 * 1 + 1 * (-1.5)) = g(-0.5) = 0}$$

For input $\mathbf{a_1}$=1,$\mathbf{a_2}$=0

$$\mathbf{a_4 = g(1 * 1 + .1 + 1 * (-.5)) = g(.5) = 1}$$

$$\mathbf{a_5 = g(1 * 1 + 0 * 1 + 1 * (-1.5)) = g(-0.5) = 0}$$

For input $\mathbf{a_1}$=1,$\mathbf{a_2}$=1

$$\mathbf{a_4 = g(1 * 1 + .1 * 1 + 1 * (-.5)) = g(1.5) = 1}$$

$$\mathbf{a_5 = g(1 * 1 + 1 * 1 + 1 * (-1.5)) = g(0.5) = 1}$$

For input $\mathbf{a_4}$=0,$\mathbf{a_5}$=0

$$\mathbf{a_7 = g(a_4 * W_{45} + a_5 * W_{57} + a_6 * W_{67}) = g(0 * 1 + 0 * 1 + 1 * (0)) = g(0) = 0}$$

For input $\mathbf{a_4}$=1,$\mathbf{a_5}$=0

$$\mathbf{a_7 = g(1 * 1 + 0 * (-1) + 1 * (0)) = g(1) = 1}$$

For input $\mathbf{a_4}$=1,$\mathbf{a_5}$=1

$$\mathbf{a_7 = g(1 * 1 + 1 * (-1) + 1 * (0)) = g(0) = 0}$$

4

Hyper Parameters during experiment:

Learning Rate:.1
Epochs:1000

Outputs:

Linear dataset final SSE: 011176

Non Linear dataset final SSE 0.14259

	correct	incorrect
Class 0	20	0
Class 1	16	0

Table 6: Number of samples predicted Correct and Incorrect using the weight gained after training on iris dataset(Linearly separable) [2]

	correct	incorrect
Class 0	59	41
Class 1	93	7

Table 7: Number of samples predicted Correct and Incorrect using the weight gained after training with artificially generated dataset(not linearly separable) [1]

Results:

After performing logistic regression using batch gradient decent optimizer on linearly separable and non-separable dataset, we have found that after each epoch the Sum of Square Error has reduced. Initially rate of SSE reduction was fast but latter the process become slow as shown in figure 4 and 5. After a number of epochs, the SSE was reducing very less with each epoch. The rate of reduction for SSE also depends on the learning rate, with high learning rate SSE reduces early. But the final value of SSE after 10000 epoch was grater than value(.1425) achieved by a low learning rates(.001, value=.1420 non linear dataset). Also, if we perform experiment with very high learning rate (grater than 2 for non linear dataset) than the SSE was not reducing with each epoch. Running the algorithm for high number of epochs might results in overfitting on the test dataset [3].
The suggested one neuron network was not able to separate non-linear distribution as the underneath hypothesis which gradient decent algorithm was optimizing, was a linear hypothesis of form.

$$\mathbf{y = w_1 * x_1 + w_2 * x_2 + bias}$$

by finding a right combination of $\mathbf{w_1}$, $\mathbf{w_2}$ and bias. Since the given data distribution can not be defined by any linear equation the gradient decent algorithm was not able to reduce the loss under certain value.

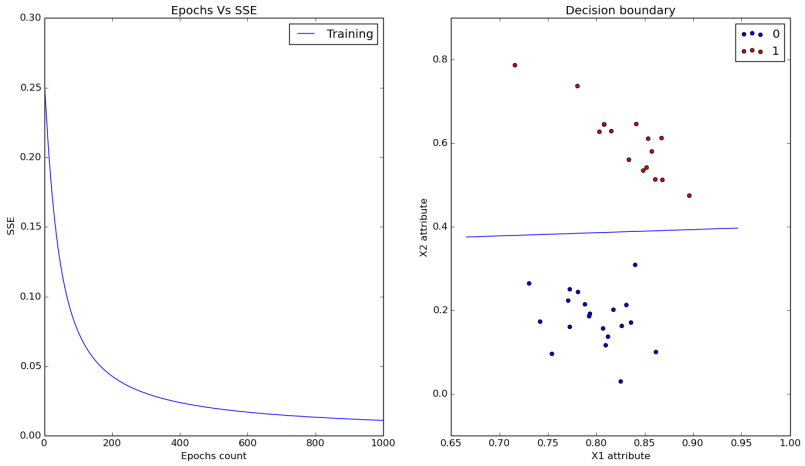

Figure 4: SSE vs Epoch curve and Decision Boundary for a linearly separable dataset [2]

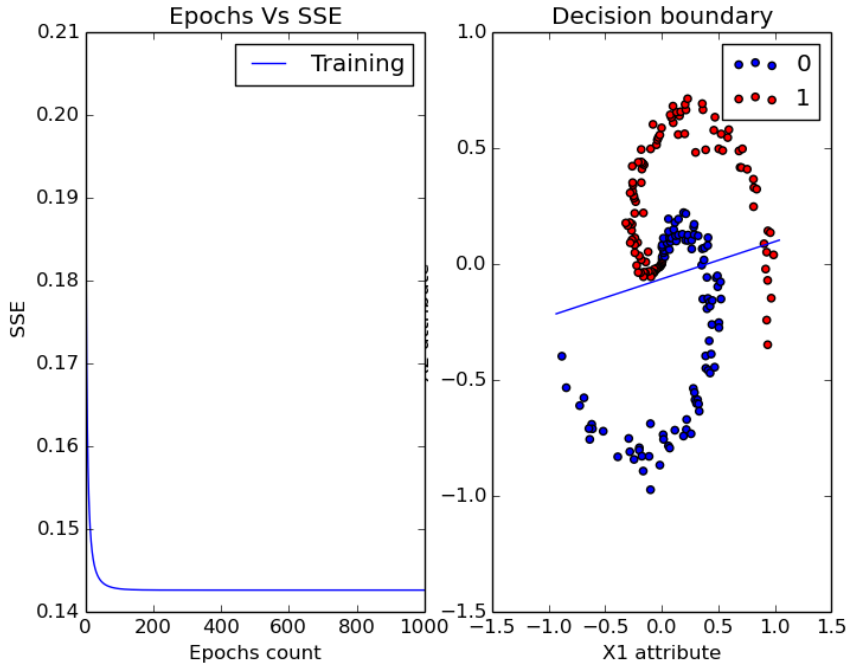

Figure 5: SSE vs Epoch curve and Decision Boundary for a not linearly separable dataset (artificially generated [1])

5

For the neural network shown in figure 6, compute the updated weights for the remaining edges in the network. Then compute the output values at nodes (v1;v2) using the updated weights for the same training sample.The network weights are provided in table .

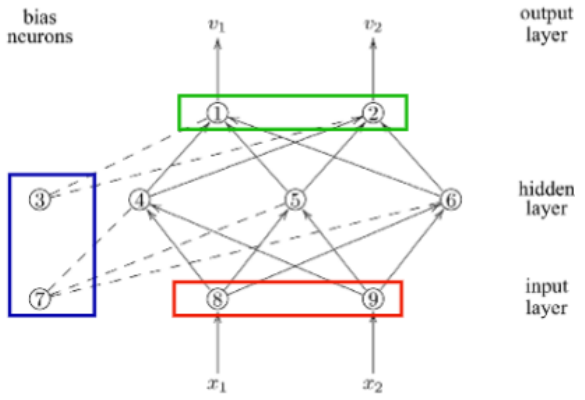

Figure 6: Given MLP

Weight	Value	Weight	Value	Weight	Value	Weight	Value
$\mathbf{W_{31}}$	.43	$\mathbf{W_{41}}$	.05	$\mathbf{W_{51}}$	.71	$\mathbf{W_{61}}$	.75
$\mathbf{W_{32}}$	.63	$\mathbf{W_{42}}$	.57	$\mathbf{W_{52}}$	.96	$\mathbf{W_{62}}$	.74
$\mathbf{W_{74}}$	.55	$\mathbf{W_{84}}$	.82	$\mathbf{W_{94}}$	.96		
$\mathbf{W_{75}}$	.26	$\mathbf{W_{85}}$	.67	$\mathbf{W_{95}}$	.06		
$\mathbf{W_{76}}$	.60	$\mathbf{W_{86}}$	1.0	$\mathbf{W_{96}}$	.36		

Table 8: Weights for the given Neural network

Node	Activation	Node	Activation	Node	Activation
$\mathbf{v_1}$	.8488	$\mathbf{v_2}$	.9267	$\mathbf{v_3}$	1
$\mathbf{v_4}$	.7738	$\mathbf{v_5}$	.8235	$\mathbf{v_6}$	.9038
$\mathbf{v_7}$	1	$\mathbf{v_8}$	2	$\mathbf{v_9}$	-1

Table 9: Activation of neurons and bias units in given Neural Network

Updated Weights After Backpropagation:
Activation values generated using updated weights:

Weight	Value	Weight	Value	Weight	Value	Weight	Value
$\mathbf{W_{31}}$	.4191	$\mathbf{W_{41}}$	.0416	$\mathbf{W_{51}}$	.6910	$\mathbf{W_{61}}$	.7402
$\mathbf{W_{32}}$	.6305	$\mathbf{W_{42}}$	.5704	$\mathbf{W_{52}}$	.9604	$\mathbf{W_{62}}$	.7405
$\mathbf{W_{74}}$	.54995	$\mathbf{W_{84}}$	.8199	$\mathbf{W_{94}}$	.9601		
$\mathbf{W_{75}}$	.25896	$\mathbf{W_{85}}$	.6679	$\mathbf{W_{95}}$	.0610		
$\mathbf{W_{76}}$	.5993	$\mathbf{W_{86}}$	.9986	$\mathbf{W_{96}}$	.3607		

Table 10: Updated Weights after backpropagation

Node	Activation	Node	Activation	Node	Activation
$\mathbf{v_1}$	.8440	$\mathbf{v_2}$	.9263	$\mathbf{v_3}$	1
$\mathbf{v_4}$	.7738	$\mathbf{v_5}$	.8226	$\mathbf{v_6}$	.9034
$\mathbf{v_7}$	1	$\mathbf{v_8}$	2	$\mathbf{v_9}$	-1

Table 11: New Activation values produced by neurons and bias units using updated weights in forward propagation

Calculations:

During the backpropogation using the error value in prediction, we can calculate δ for all layers for updating the previous weights. [3]
For output layer the value of δ will be:

$$\delta_1 = (\mathbf{v_1} - \mathbf{label_1}) * \mathbf{v_1} * (\mathbf{1} - \mathbf{v_1}) = (\mathbf{.8488 - 0}) * \mathbf{.8488} * (\mathbf{1} - \mathbf{.8488}) = \mathbf{.1089}$$

$$\delta_2 = (\mathbf{v_2} - \mathbf{label_2}) * \mathbf{v_2} * (\mathbf{1} - \mathbf{v_2}) = (\mathbf{.9267 - 1}) * \mathbf{.9267} * (\mathbf{1} - \mathbf{.9276}) = \mathbf{-.0050}$$

Using the δ values of the output layer, we can calculate δ for previous hidden layer.

$$\delta_3 = (\delta_1 * \mathbf{W_{31}} + \delta_2 * \mathbf{W_{32}}) * \mathbf{v_3} * (\mathbf{1} - \mathbf{v3}) = \mathbf{0}$$

$$\delta_4 = (\delta_1 * \mathbf{W_{41}} + \delta_2 * \mathbf{W_{42}}) * \mathbf{v_4} * (\mathbf{1} - \mathbf{v4}) = \mathbf{.0005}$$

$$\delta_5 = (\delta_1 * \mathbf{W_{51}} + \delta_2 * \mathbf{W_{52}}) * \mathbf{v_5} * (\mathbf{1} - \mathbf{v5}) = \mathbf{.01038218978}$$

$$\delta_6 = (\delta_1 * \mathbf{W_{61}} + \delta_2 * \mathbf{W_{62}}) * \mathbf{v_6} * (\mathbf{1} - \mathbf{v6}) = \mathbf{.00677958004}$$

Using the δ values of the second hidden layer, we can calculate δ for the first hidden layer.

$$\delta_7 = (\delta_4 * \mathbf{W_{74}} + \delta_5 * \mathbf{W_{75}} + \delta_6 * \mathbf{W_{76}}) * \mathbf{v_7} * (\mathbf{1} - \mathbf{v7}) = \mathbf{0}$$

$$\delta_8 = (\delta_4 * \mathbf{W_{84}} + \delta_5 * \mathbf{W_{85}} + \delta_6 * \mathbf{W_{86}}) * \mathbf{v_8} * (\mathbf{1} - \mathbf{v8}) = \mathbf{-.028356}$$

$$\delta_9 = (\delta_4 * \mathbf{W_{94}} + \delta_5 * \mathbf{W_{95}} + \delta_6 * \mathbf{W_{96}}) * \mathbf{v_9} * (\mathbf{1} - \mathbf{v9}) = \mathbf{-.007104}$$

Using learning rate η=.1, respective δ values and inputs to the neuron we can update respective weights.

$$\mathbf{W_{31}} = \mathbf{W_{31}} - \eta * \delta_1 * \mathbf{v_3} = \mathbf{.43} - \mathbf{.1} * \mathbf{.1089} = \mathbf{0.41911}$$

$$\mathbf{W_{32} = W_{32} - \eta * \delta_1 * v_3 = .63 - .1 * (-.0050) * 1 = 0.6305}$$
$$\mathbf{W_{41} = W_{41} - \eta * \delta_1 * v_4 = .0500 - .1 * .1089 * .7738 = .041573318}$$
$$\mathbf{W_{42} = W_{42} - \eta * \delta_2 * v_4 = .5700 + .1 * .005 * .7738 = .5703869}$$
$$\mathbf{W_{51} = W_{51} - \eta * \delta_1 * v_5 = .70 - .1 * .1089 * .8235 = .691032085}$$
$$\mathbf{W_{52} = W_{52} - \eta * \delta_2 * v_5 = .96 + .1 * .005 * .8235 = .96041175}$$
$$\mathbf{W_{61} = W_{61} - \eta * \delta_1 * v_6 = .75 - .1 * .1089 * .9038 = .740157618}$$
$$\mathbf{W_{62} = W_{62} - \eta * \delta_2 * v_6 = .74 + .1 * .005 * .9038 = .7404519}$$
$$\mathbf{W_{74} = W_{74} - \eta * \delta_4 * v_7 = .55 - .1 * .0005 * 1 = .54995}$$
$$\mathbf{W_{75} = W_{75} - \eta * \delta_5 * v_7 = .26 - .1 * .0103822 = .2589617}$$
$$\mathbf{W_{76} = W_{76} - \eta * \delta_6 * v_7 = .60 - .1 * .00677958 * 1 = .599322042}$$
$$\mathbf{W_{84} = W_{84} - \eta * \delta_4 * v_8 = .82 - .1 * .0005 * 2 = 0.8199}$$
$$\mathbf{W_{85} = W_{85} - \eta * \delta_5 * v_8 = .67 - .1 * .01038 * 2 = .667924}$$
$$\mathbf{W_{86} = W_{86} - \eta * \delta_6 * v_8 = 1 - .1 * .006779 * 2 = .9986442}$$

$$\mathbf{W_{94} = W_{94} - \eta * \delta_4 * v_9 = .96 - .1 * .0005 * (-1) = .96005}$$
$$\mathbf{W_{95} = W_{95} - \eta * \delta_5 * v_9 = .06 - .1 * .01038 * (-1) = .061038}$$
$$\mathbf{W_{96} = W_{96} - \eta * \delta_6 * v_9 = .36 - .1 * .006779 * (-1) = .3606779}$$

Forward propagation with updated weights:

$$\mathbf{v_4 = g(v_8 * W_{84} + v_9 * W_{94} + v_7 * W_{74}) = g(2 * .8199 - 1 * .96005 + 1 * .54995)}$$
$$\mathbf{v_4 = g(1.2297) = .77376606294}$$

$$\mathbf{v_5 = g(v_8 * W_{85} + v_9 * W_{95} + v_7 * W_{75}) = g(2 * .667924 - 1 * .061038 + 1 * .2589617)}$$
$$\mathbf{v_5 = g(1.5337717) = .82255748887}$$

$$\mathbf{v_6 = g(v_8 * W_{86} + v_9 * W_{96} + v_7 * W_{76}) = g(2 * .9986442 - 1 * .36106779 + 1 * .599322042)}$$
$$\mathbf{v_6 = g(2.235542652) = .9033961575}$$

$$\mathbf{v_1 = g(v_4 * W_{41} + v_5 * W_{51} + v_6 * W_{61} + v_3 * W_{31})}$$

$$\mathbf{v_1} = \mathbf{g}(\mathbf{.77376606294} * \mathbf{.041573318} + \mathbf{.82255748887} * \mathbf{.691032085}$$
$$+\mathbf{.90345530776} * \mathbf{.740157618} + \mathbf{1} * \mathbf{0.41911})$$
$$\mathbf{v_1} = \mathbf{g}(\mathbf{1.68839096772}) = \mathbf{.84401243926}$$

$$\mathbf{v_2} = \mathbf{g}(\mathbf{v_4} * \mathbf{W_{42}} + \mathbf{v_5} * \mathbf{W_{52}} + \mathbf{v_6} * \mathbf{W_{62}} + \mathbf{v_3} * \mathbf{W_{32}})$$
$$\mathbf{v_2} = \mathbf{g}(\mathbf{.77376606294} * \mathbf{.5703869} + \mathbf{.82255748887} * \mathbf{.961032085}$$
$$+\mathbf{.90345530776} * \mathbf{.7404519} + \mathbf{1} * \mathbf{.6305})$$
$$\mathbf{v_2} = \mathbf{g}(\mathbf{2.53131536372}) = \mathbf{.92630819223}$$

References

[1] karpathy. cs231 training a softmax linear classifier, 2016. URL `http://cs231n.github.io/neural-networks-case-study/`.

[2] M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml/datasets/Iris`.

[3] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003. ISBN 0137903952.