MACHINE LEARNING

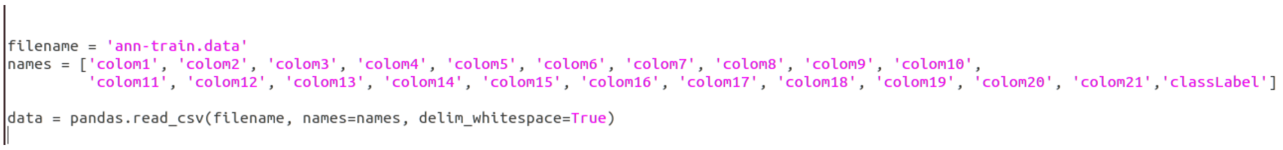
HOMEWORK1 REPORT

Name: Hüseyin Eryılmaz ***ID: 20160808058***

1- Parameters Which Used In Implementation

1.1. Reading Data

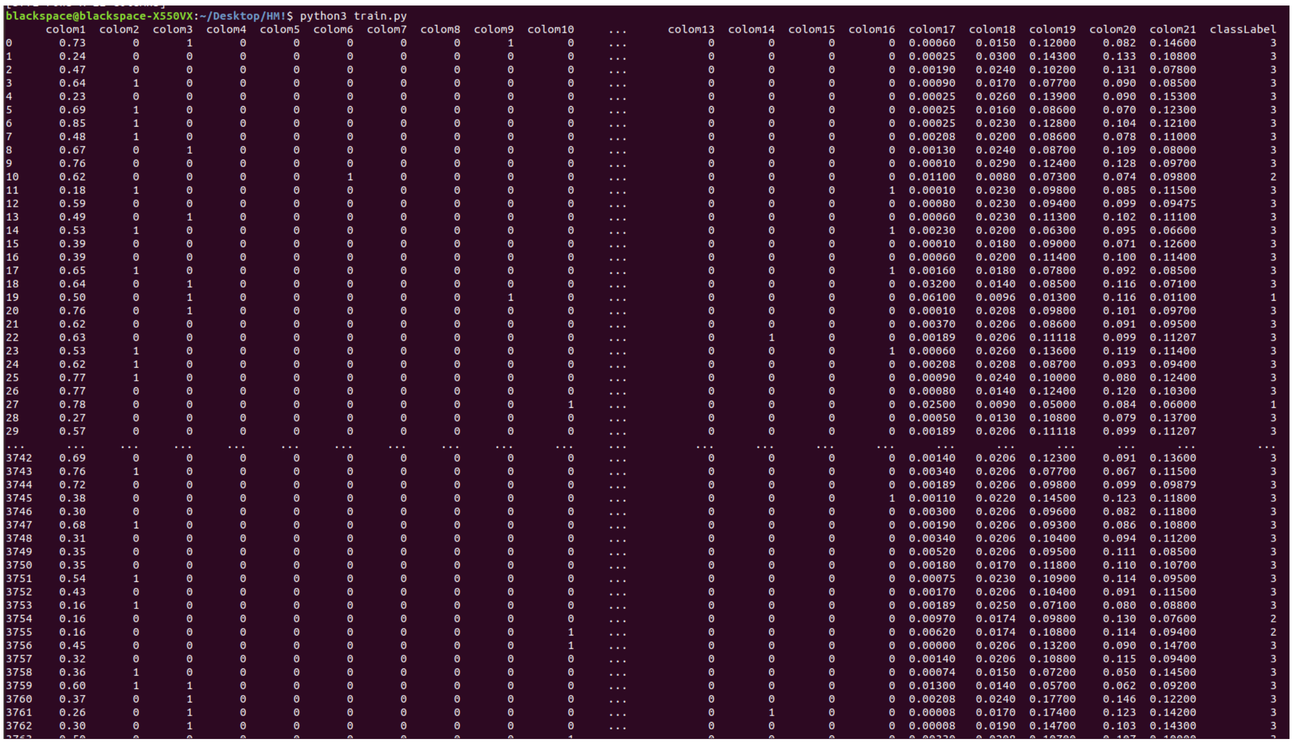
First part of code reads the data from dataset file and split the data recording to columns. Figure 1.1 shows to codes used reading.



```
filename = 'ann-train.data'
names = ['colom1', 'colom2', 'colom3', 'colom4', 'colom5', 'colom6', 'colom7', 'colom8', 'colom9', 'colom10',
         'colom11', 'colom12', 'colom13', 'colom14', 'colom15', 'colom16', 'colom17', 'colom18', 'colom19', 'colom20', 'colom21','classLabel']

data = pandas.read_csv(filename, names=names, delim_whitespace=True)
```

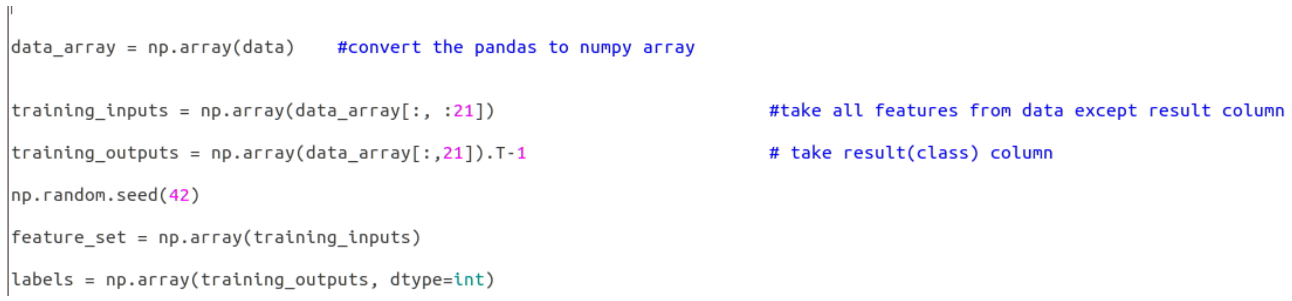
Figure 1.1.

Figure 1.2. shows the data after reading,



```
blackspace@blackspace-X550VX:~/Desktop/HM!$ python3 train.py
      colom1  colom2  colom3  colom4  colom5  colom6  colom7  colom8  colom9  colom10     ...       colom13  colom14  colom15  colom16  colom17  colom18  colom19  colom20  colom21  classLabel
0       0.73       0       1       0       0       0       0       0       1        0     ...             0        0        0        0  0.00060   0.0150  0.12000    0.082  0.14600           3
1       0.24       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00025   0.0300  0.14300    0.133  0.10800           3
2       0.47       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00190   0.0240  0.10200    0.131  0.07800           3
3       0.64       1       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00090   0.0170  0.07700    0.090  0.08500           3
4       0.23       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00025   0.0260  0.13900    0.090  0.15300           3
5       0.69       1       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00025   0.0160  0.08600    0.070  0.12300           3
6       0.85       1       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00025   0.0230  0.12800    0.104  0.12100           3
7       0.48       1       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00208   0.0200  0.08600    0.078  0.11000           3
8       0.67       0       1       0       0       0       0       0       0        0     ...             0        0        0        0  0.00130   0.0240  0.08700    0.109  0.08000           3
9       0.76       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00010   0.0290  0.12400    0.128  0.09700           3
10      0.62       0       0       0       0       1       0       0       0        0     ...             0        0        0        0  0.01100   0.0080  0.07300    0.074  0.09800           2
11      0.18       1       0       0       0       0       0       0       0        0     ...             0        0        0        1  0.00010   0.0230  0.09800    0.085  0.11500           3
12      0.59       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00080   0.0230  0.09400    0.099  0.09475           3
13      0.49       0       1       0       0       0       0       0       0        0     ...             0        0        0        0  0.00060   0.0230  0.11300    0.102  0.11100           3
14      0.53       1       0       0       0       0       0       0       0        0     ...             0        0        0        1  0.00230   0.0200  0.06300    0.095  0.06600           3
15      0.39       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00010   0.0180  0.09000    0.071  0.12600           3
16      0.39       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00060   0.0200  0.11400    0.100  0.11400           3
17      0.65       1       0       0       0       0       0       0       0        0     ...             0        0        0        1  0.00160   0.0180  0.07800    0.092  0.08500           3
18      0.64       0       1       0       0       0       0       0       0        0     ...             0        0        0        0  0.03200   0.0140  0.08500    0.116  0.07100           3
19      0.50       0       1       0       0       0       0       0       1        0     ...             0        0        0        0  0.06100   0.0096  0.01300    0.116  0.01100           1
20      0.76       0       1       0       0       0       0       0       0        0     ...             0        0        0        0  0.00010   0.0208  0.09800    0.101  0.09700           3
21      0.62       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00370   0.0206  0.08600    0.091  0.09500           3
22      0.63       0       0       0       0       0       0       0       0        0     ...             0        1        0        0  0.00189   0.0206  0.11118    0.099  0.11207           3
23      0.53       1       0       0       0       0       0       0       0        0     ...             0        0        0        1  0.00060   0.0260  0.13600    0.119  0.11400           3
24      0.62       1       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00208   0.0208  0.08700    0.093  0.09400           3
25      0.77       1       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00090   0.0240  0.10000    0.080  0.12400           3
26      0.77       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00080   0.0140  0.12400    0.120  0.10300           3
27      0.78       0       0       0       0       0       0       0       0        1     ...             0        0        0        0  0.02500   0.0090  0.05000    0.084  0.06000           1
28      0.27       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00050   0.0130  0.10800    0.079  0.13700           3
29      0.57       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00189   0.0206  0.11118    0.099  0.11207           3
...      ...     ...     ...     ...     ...     ...     ...     ...     ...      ...     ...           ...      ...      ...      ...      ...      ...      ...      ...      ...         ...
3742    0.69       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00140   0.0206  0.12300    0.091  0.13600           3
3743    0.76       1       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00340   0.0206  0.07700    0.067  0.11500           3
3744    0.72       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00189   0.0206  0.09800    0.099  0.09879           3
3745    0.38       0       0       0       0       0       0       0       0        0     ...             0        0        0        1  0.00110   0.0220  0.14500    0.123  0.11800           3
3746    0.30       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00300   0.0206  0.09600    0.082  0.11800           3
3747    0.68       1       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00190   0.0206  0.09300    0.086  0.10800           3
3748    0.31       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00340   0.0206  0.10400    0.094  0.11200           3
3749    0.35       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00520   0.0206  0.09500    0.111  0.08500           3
3750    0.35       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00180   0.0170  0.11800    0.110  0.10700           3
3751    0.54       1       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00075   0.0230  0.10900    0.114  0.09500           3
3752    0.43       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00170   0.0206  0.10400    0.091  0.11500           3
3753    0.16       1       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00189   0.0250  0.07100    0.080  0.08800           3
3754    0.16       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00970   0.0174  0.09800    0.130  0.07600           2
3755    0.16       0       0       0       0       0       0       0       0        1     ...             0        0        0        0  0.00620   0.0174  0.10800    0.114  0.09400           2
3756    0.45       0       0       0       0       0       0       0       0        1     ...             0        0        0        0  0.00000   0.0206  0.13200    0.090  0.14700           3
3757    0.32       0       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00140   0.0206  0.10800    0.115  0.09400           3
3758    0.36       1       0       0       0       0       0       0       0        0     ...             0        0        0        0  0.00074   0.0150  0.07200    0.050  0.14500           3
3759    0.60       1       1       0       0       0       0       0       0        0     ...             0        0        0        0  0.01300   0.0140  0.05700    0.062  0.09200           3
3760    0.37       0       1       0       0       0       0       0       0        0     ...             0        0        0        0  0.00208   0.0240  0.17700    0.146  0.12200           3
3761    0.26       0       1       0       0       0       0       0       0        0     ...             0        1        0        0  0.00008   0.0170  0.17400    0.123  0.14200           3
3762    0.30       0       1       0       0       0       0       0       0        0     ...             0        0        0        0  0.00008   0.0190  0.14700    0.103  0.14300           3
```

Figure 1.2.



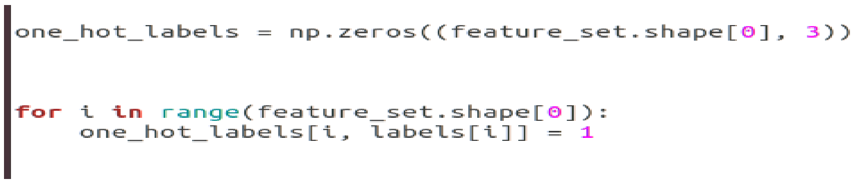
```
data_array = np.array(data)    #convert the pandas to numpy array

training_inputs = np.array(data_array[:, :21])                       #take all features from data except result column

training_outputs = np.array(data_array[:,21]).T-1                    # take result(class) column

np.random.seed(42)

feature_set = np.array(training_inputs)

labels = np.array(training_outputs, dtype=int)
```

Figure 1.3.

After reading it converts this data to an numpy array. It converts the data to numpy array because it really supports fast matrix operations. We have a numpy array and it has all features and classlabel(result) on it. I split the data to take features and and classLabels. Figure 1.3. show the code block which used this step.

1.2. Preparing to Neural Network

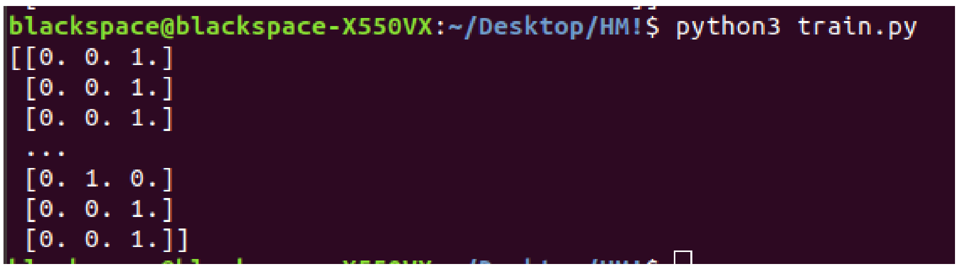
We will use “one hot label”, “sigmoid function”, “softmax function”, “accuracy function”.



```
one_hot_labels = np.zeros((feature_set.shape[0], 3))

for i in range(feature_set.shape[0]):
    one_hot_labels[i, labels[i]] = 1
```

Figure 1.4.

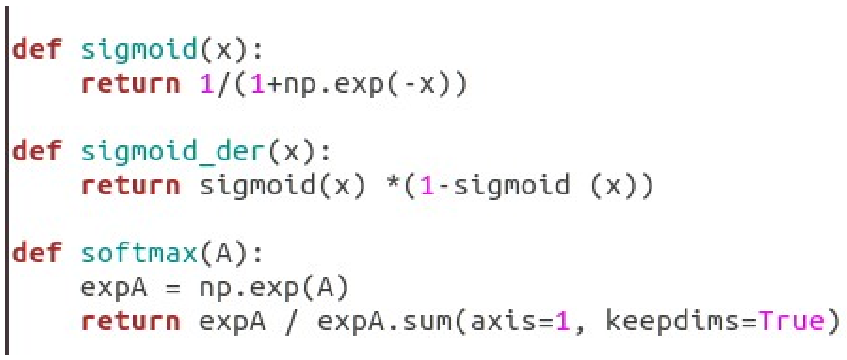
If your classification problem is a multi-class classification problem then the “one hot label” is a good approach. Figure 1.4. shows the “one hot label” implementation. It creates a 3772x3 matrix for here and fill the matrix with zeros. It shows that we have 3772 elements and 3 classLabel for each element. Then we put 1 the corresponding class label on matrix. It show like:



```
blackspace@blackspace-X550VX:~/Desktop/HM!$ python3 train.py
[[0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 ...
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]]
```

Figure 1.5.

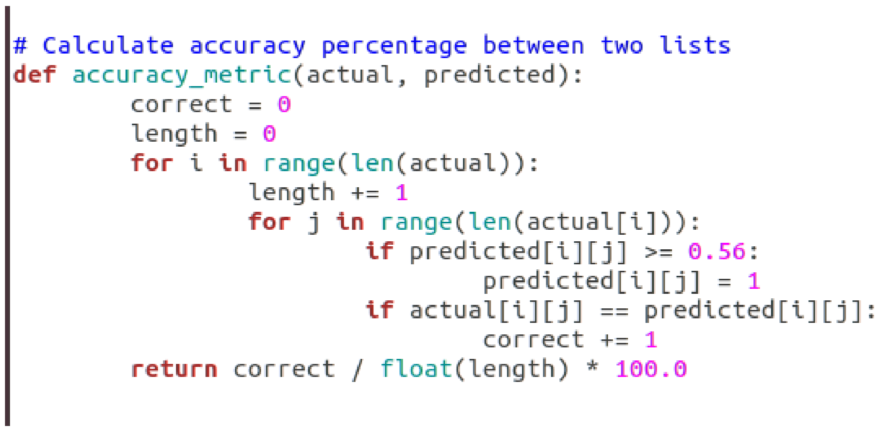
If 1st row 3rd column is equal to 1 then our first element belongs to 3rd class. “One hot label” will used on weight calculation on next steps.



```python
def sigmoid(x):
    return 1/(1+np.exp(-x))

def sigmoid_der(x):
    return sigmoid(x) *(1-sigmoid (x))

def softmax(A):
    expA = np.exp(A)
    return expA / expA.sum(axis=1, keepdims=True)
```

Figure 1.6.

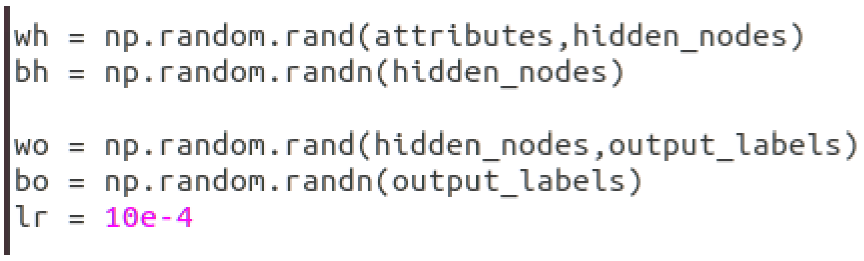
You can see the implementation of sigmoid funtion and softmax function on Figure 1.6. We will use these functions forward propagation and back propagations. Sigmoid function makes a normalization on forward propagation step. Softmax function makes a normalization on outputs. For example, we have three class label and softmax function gives us a percentage to each 3 class label like [0.2   0.6   0.2].



```python
# Calculate accuracy percentage between two lists
def accuracy_metric(actual, predicted):
        correct = 0
        length = 0
        for i in range(len(actual)):
                length += 1
                for j in range(len(actual[i])):
                        if predicted[i][j] >= 0.56:
                                predicted[i][j] = 1
                        if actual[i][j] == predicted[i][j]:
                                correct += 1
        return correct / float(length) * 100.0
```

Figure 1.7.

We have a huge dataset and 21 features so we cant take a big percentage from softmax function. Its predict range changing between 20s% and 60s%. So I applied a threshold when I calculate the accuracy. If predict percentage over and equal to 56% then it accepts the class label and calculate accuracy. I mean if output is [0.30 0.56 0.14] then program accept that it belongs to 2nd class because our threshold is 56%.



```python
wh = np.random.rand(attributes,hidden_nodes)
bh = np.random.randn(hidden_nodes)

wo = np.random.rand(hidden_nodes,output_labels)
bo = np.random.randn(output_labels)
lr = 10e-4
```

Figure 1.8.

Lets define the code block at Figure 1.8.

wh = weight of hidden layer (creates a matrix regarding to feature number and hidden Node number and fill it with random numbers)

bh = bias of hidden

wo = weight of output layer (creates a matrix regarding to output number and hidden Node number and fill it with random numbers)

bo = bias of output

lr = learning rate, it is important in gradient descent. 0.001 is gives best result so I checked and choose it.

1.3. Forward Propagation

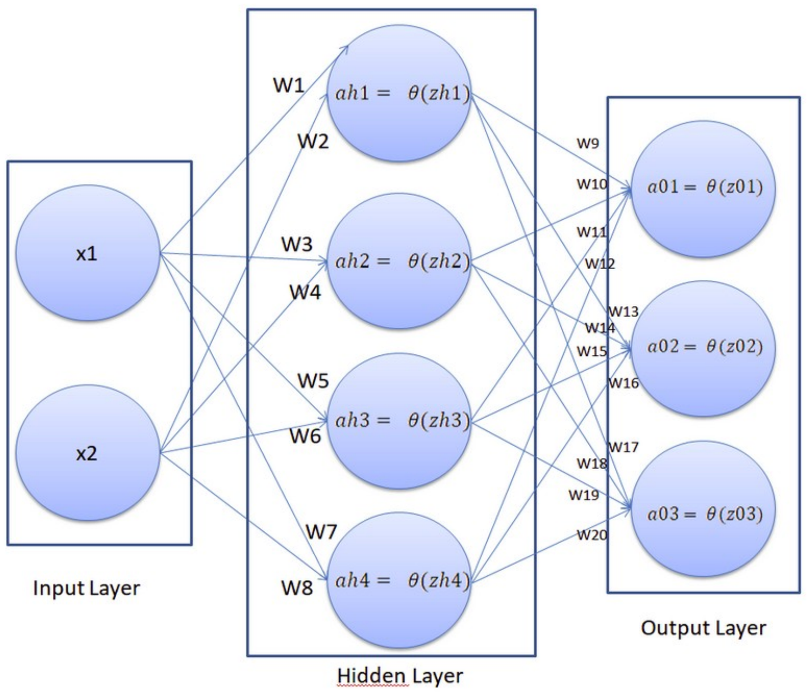


Figure 1.9.

Our Neural Network model will similar like this.

First step of forward propagation is calculate the output of weights and features so we can formulate it like here:

$$zh1 = x1w1 + x2w2 + b$$

$$ah1 = \frac{1}{1 + e^{-zh1}}$$

Figure 1.10.

In above Figure we calculate of a dot product between features and weights and add bias to result so we obtain the hidden layers results. Second formula make a normalization with makes sigmoid function. Our code block is here for this operation:

```
zh = np.dot(feature_set, wh) + bh  #output of hidden layer after multiplication
ah = sigmoid(zh)  #take sigmoid of result
```

Figure 1.11.

Now, we have the hidden layer values. We will use this values to calculate output value now.

$$zo1 = ah1w9 + ah2w10 + ah3w11 + ah4w12$$

$$zo2 = ah1w13 + ah2w14 + ah3w15 + ah4w16$$

$$zo3 = ah1w17 + ah2w18 + ah3w19 + ah4w20$$

$$ao1(zo) = \frac{e^{zo1}}{\sum_{k=1}^{k} e^{zok}}$$

Figure 1.12.

Here we calculate the output values for each node. Then take softmax of this operation. Its code block just like here:

```
zo = np.dot(ah, wo) + bo    #hidden layers weihgt*a0 (values of aoutput layer)
ao = softmax(zo)
```

Figure 1.13.

1.4. Back Propagation

We make forward propagation at last step. We will make back propagation now. Back propagation is the operation that update weights. We will use gradient descent algorithm here:

$$repeat\ until\ convergence : \left\{ w_j := w_j - \alpha \frac{\partial}{\partial w_j} J(w_0, w_1 \ldots\ldots w_n) \right\} \ldots\ldots\ldots\ldots (1)$$

Figure 1.14.

Above algorithm shows the gradient descent. Briefly, we will update the weights until minimize the cost function. Instead of normal cost function we will use cross-entropy function(loss function) here because we have a multi-class classification problem.

$$cost(y, ao) = -\sum_i y_i \log ao_i$$

Figure 1.15.

Figure 1.15. shows the our cost function.

First step of back propagation is update to output weights(wo). We can formulate this step like here:

$$\frac{dcost}{dwo} = \frac{dcost}{dao} *, \frac{dao}{dzo} * \frac{dzo}{dwo} \ldots\ldots (1)$$

$$\frac{dcost}{dao} * \frac{dao}{dzo} \ldots\ldots\ldots (2)$$

$$\frac{dcost}{dao} * \frac{dao}{dzo} = ao - y \ldots\ldots\ldots (3)$$

$$\frac{dzo}{dwo} = ah$$

$$\frac{dcost}{dbo} = \frac{dcost}{dao} * \frac{dao}{dzo} * \frac{dzo}{dbo} \ldots\ldots (4)$$

$$\frac{dcost}{dbo} = ao - y \ldots\ldots\ldots\ldots (5)$$

Figure 1.16.

Here is our code block for Figure 1.16.:

```
dcost_dzo = ao - one_hot_labels
dzo_dwo = ah

dcost_wo = np.dot(dzo_dwo.T, dcost_dzo)

dcost_bo = dcost_dzo
```

Figure 1.17.

We find the cost function value at Figure 1.17. now we can update output weight(wo) and output bias(bo). We use gradient descent algorithm like code block just like here:

```
wo -= lr * dcost_wo
bo -= lr * dcost_bo.sum(axis=0)
```

Figure 1.18.

Second step is the back propagation is update to hidden layer weights(wh). We can formulate it like here:

$$\frac{dcost}{dwh} = \frac{dcost}{dah} *, \frac{dah}{dzh} * \frac{dzh}{dwh} \ldots\ldots (6)$$

$$\frac{dcost}{dah} = \frac{dcost}{dzo} * \frac{dzo}{dah} \ldots\ldots (7)$$

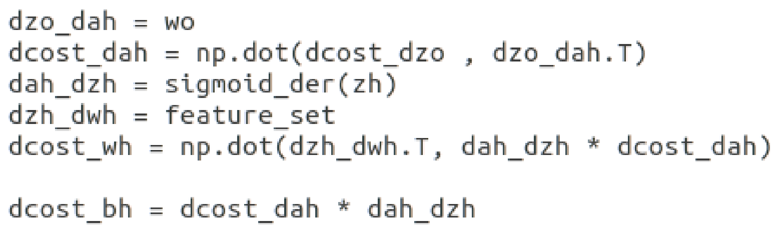
$$\frac{dcost}{dao} * \frac{dao}{dzo} = \frac{dcost}{dzo} == ao - y \ldots\ldots\ldots (8)$$

$$\frac{dzo}{dah} = wo \ldots\ldots (9)$$

$$\frac{dah}{dzh} = sigmoid(zh) * (1 - sigmoid(zh)) \ldots\ldots\ldots (10)$$

Figure 1.19.

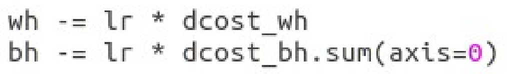
Here is our code block for Figure 1.19. :



```
dzo_dah = wo
dcost_dah = np.dot(dcost_dzo , dzo_dah.T)
dah_dzh = sigmoid_der(zh)
dzh_dwh = feature_set
dcost_wh = np.dot(dzh_dwh.T, dah_dzh * dcost_dah)

dcost_bh = dcost_dah * dah_dzh
```

Figure 1.20.

We find the cost value for the update hidden layer weights(wh) and hidden layer bias(bh) at Figure 1.20. Now we can update the hidden layer weights like here:



```
wh -= lr * dcost_wh
bh -= lr * dcost_bh.sum(axis=0)
```

Figure 1.21.

2- Accuracy Table

Hidden Layer Node number	Train Accuracy	Train Time	Test Accuracy	Class 1 Test Accuracy	Class 2 Test Accuracy	Class 2 Test Accuracy
10	78.34	6 min. and 50 sec.	93.64	61.64	15.81	98.7
25	94.59	18 min. and 40 sec.	74.59	67.12	40.67	76.65
40	79.05	23 min. and 08 sec.	93.61	64.38	16.38	98.58
50	94.77	29 min. and 40 sec.	74.64	64.38	36.15	77.02
60	79.34	41 min. and 01 sec.	93.93	71.23	18.07	98.67
75	79.77	41 min. and 47 sec.	93.61	69.89	16.94	98.42

Table 1. Note: Threshold was 0.56 at Train and Test Accuracy.

3- Observations

Normalization

We have 21 features at this data set. We need to normalize this features to get better outputs and save time. If you work with 21 features then it really takes too much time while data processing. We have 3772 elements on data set but it takes really too much time. You can check the Table 1 and you can see the time complexity with 21 features cause of this time complexity is matrix operations. If we choose 50 nodes in hidden layer then we need to (21x30) size matrix to hidden layer weight.

Unbalanced Class Distributions

If we check the class distribution on training dataset then we can see that data set is unbalanced. Numbers like here:

Class 1 = 93

Class 2 = 191

Class 3 = 3488

If we check the Table 1 then we can see the affect of this unbalanced distribution. So lets get select 10 Nodes hidden layer. Test accuracy is 93.64 but if we check the class based accuracy it 61.64 for class 1, 15.81 for class 2 and 98.7 for class 3. As you see, it seems like good if we just look the general accuracy but it is too low especially for class 2.

Stochastic, Batch, and Mini-Batch Gradient Descent

Main difference is update times these 3 gradient descent type. If we make a table it will see just here:

	Stochastic	Batch	Mini-Batch
Update	Every Time	Just once	Every mini-batch
Fast	Faster	Slower	Fast
Accuracy	Low accuracy	High accuracy	Medium accuracy

Table 2.

Table 2 explain the differences of gradient descent types. Mini-batch method most effective one I think. Because if we huge data it can be good about time and accuracy.