AKDENIZ UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

Machine Learning – Homework 3

Temmuz Burak Yavuzer - 20160808026

General Explanation of Multilayer Perceptrons*[1]*

Artificial Neural Networks is often just called ann or multilayer perceptrons after probably one of the most useful type of neural network.Basicly a perceptron is a single neuron model that was a precursor to larger neural networks.

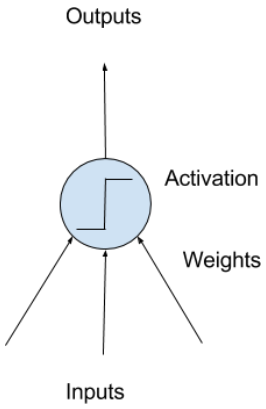
The goal is not to create realistic models of the brain, but instead to develop robust algorithms and data structures that we can use to model difficult problems.

The power of neural networks comes from their ability to learn the representation in your training data and how to best relate it to the output variable that you want to predict. This means that it has kind of sense to neural networks learn a mapping.

Capability of NN comes from the multilayered structure of the networks

Neuron

These are simple computational units that have weighted input signals and produce an output signal using an activation function.



Each neuron has a bias which can be explained as an input that always has the value 1.0 and it too must be weighted. Weights are often initialized to small random values, such as values in the range 0 to 0.3, although more complex initialization schemes can be used.

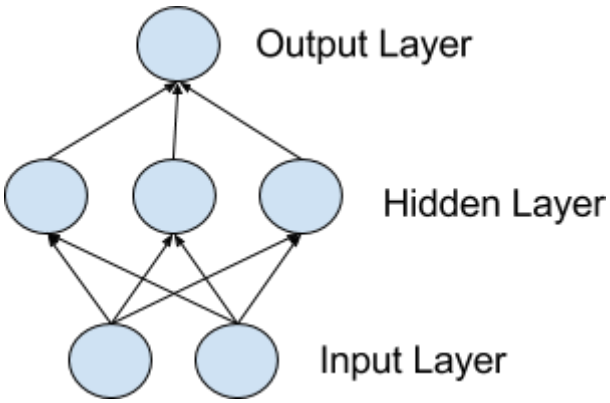
Like linear regression, larger weights indicate increased complexity and fragility. It is desirable to keep weights in the network small and regularization techniques can be used.

Activation

The weighted inputs are summed and passed through an activation function, sometimes called a transfer function. An activation function is a simple mapping of summed weighted input to the output of the neuron

If the summed input was above a threshold, for example 0.5, then the neuron would output a value of 1.0, otherwise it would output a 0.0.

Networks and Neurons



A row of neurons is called a layer and one network can have multiple layers. The architecture of the neurons in the network is often called the network topology.

The bottom layer that takes input from your dataset is called the ***input layer***.These are not neurons as described above, but simply pass the input value though to the next layer.

Layers after the input layer are called ***hidden*** **layers** because that are not directly exposed to the input. The simplest network structure is to have a single neuron in the hidden layer that directly outputs the value. Given increases in computing power and efficient libraries, very deep neural networks can be constructed.

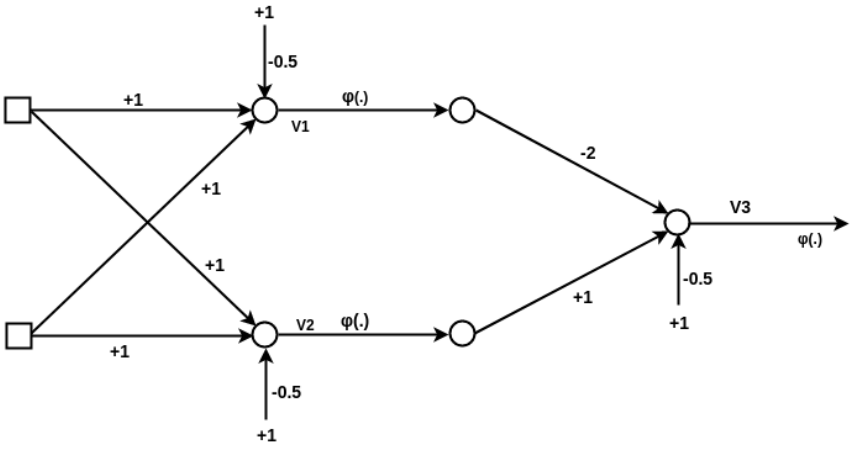
The final hidden layer is called the ***output layer*** and it is responsible for outputting a value or vector of values that correspond to the format required for the problem.

Three Steps in the Training of the Model *[2]*

1. *Forward pass*
2. *Calculate error or loss*
3. *Backward pass*

1)Forward Pass

In this step of training the model, we just pass the input to model and multiply with weights and add bias at every layer and find the calculated output of the model.



2)Loss Calculate

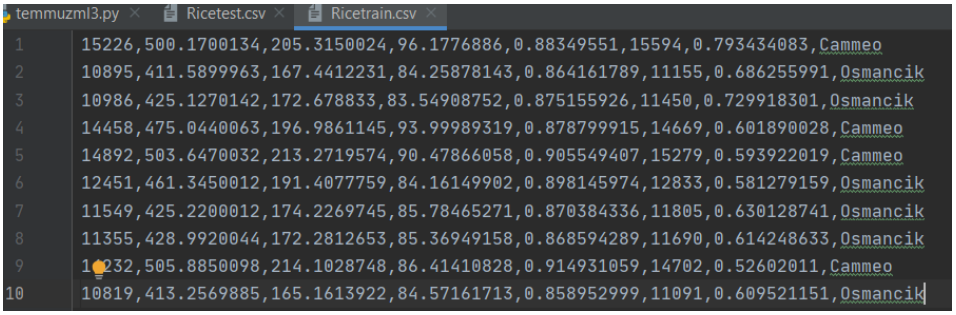
When we pass the data,we will get some output from the model that is called Predicted output and we have the label with the data that is real output or expected output. Based upon these both we calculate the loss that we have to backpropagate.

3)Backward Pass

After calculating the loss, we backpropagate the loss and updates the weights of the model by using gradient. This is the main step in the training of the model. In this step, weights will adjust according to the gradient flow in that direction.

Implementation of Multilayer Perceptron

Before all that we have to understand out ricetest and ricetrain csv file. As you can see we have 7 output for each rice data. We want to get the rice type depends on the values to understand it is Osmancık or Cammeo. Which means that our output type will be 2 for rice dataset.



We can move on the basic libraries that I have used during the process. Basically those libraries used for mathematical operations and data management.

```
from csv import reader
from math import exp
import numpy as np
from sklearn import metrics
from random import random
```

As you can see I have created a function to read csv file. It is same as the previous homework. Just typical csv read function

```
def _csv(csv):
    rices = list()
    with open(csv, 'r') as file:
        csv_r = reader(file)
        for r in csv_r:
            if not r:continue
            rices.append(r)
    return rices
```

As we talked about in the second homework single layers difference between the multilayer perceptron is hidden layers between inputs and outputs. Those hidden layer help us to get better results. In this part, I created a start to network function which takes several parameters that I written above.Such as number of input, number layer of hidden layers or number of output layers

```
def start_net(number_inpt, number_h, number_outpt):
    netwrk = list()
    h_layer = [{'weights': [random() for i in range(number_inpt + 1)]} for i in range(number_h)]
    netwrk.append(h_layer)
    o_layer = [{'weights': [random() for i in range(number_hidden + 1)]} for i in range(number_outpt)]
    netwrk.append(o_layer)
    return netwrk
```

So we created out network but it is time to create function to neuron active and neuron transfer for calculating activation for neuron with a given input. After that which means neuron is activated, this activation is need to transfer.

For the activation, basically we are calculating the bias and sum of the multiplication of weight and input

For the transfer, we are calculating value of 1 divided by 1 plus power of minus activation value of euler number . For the Sigmoid function calculations is

$$y(v_i) = \tanh(v_i) \text{ and } y(v_i) = (1 + e^{-v_i})^{-1}.$$

For the derivative, basically calculating the output multiplied by 1 minus output.Which will be used in Error Backpropagation function You can see the functions below

```
def active(weight, inpt):
    active = weight[-1]
    for i in range(len(weight) - 1):
        active += weight[i] * inpt[i]
    return active
def trnsfer(active):return 1.0 / (1.0 + exp(-active))
def trnsferderive(outpt):return outpt * (1.0 - outpt)
```

Now, It is time to calculate the error for neuron.We are calculating the multiplication of transfer derivative and multiplication of weight and error

```
def bckward_propgate_err(network, expcted):
    for i in reversed(range(len(network))):
        layer = network[i]
        errs = list()
        if i != len(network) - 1:
            for j in range(len(layer)):
                err = 0.0
                for neuron in network[i + 1]:
                    err += (neuron['weights'][j] * neuron['delta'])
                errs.append(err)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errs.append(expcted[j] - neuron['output'])
        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errs[j] * trnsferderive(neuron['output'])
```

We are able to find the change weight because I already had our back propagation function. We can use this function for how to change the weights of the inputs for network. With the help of the for loop we can continue for each row.When we got our change weight function, we can focus on creating the training of the network because all the necessary function already completed.Below function will train the dataset for every iteration for each epoch number with the help of the learning rate. For example when we choose epoch as a 10 , we will get different outputs for each 1,2,3…..10.

```
def change_weight(netwrk, r, learning_rate):
    for i in range(len(netwrk)):
        inpts = r[:-1]
        if i != 0:
            inpts = [neuron['output'] for neuron in netwrk[i - 1]]
        for neuron in netwrk[i]:
            for j in range(len(inpts)):
                neuron['weights'][j] += learning_rate * neuron['delta'] * inpts[j]
            neuron['weights'][-1] += learning_rate * neuron['delta']
def train_net(netwrk, train, learning_rate, number_epch, number_outp):
    errs = []
    for epch in range(number_epch):
        sum_err = 0
        for r in train:
            outp = frward_propgate(netwrk, r)
            expectd = [0 for i in range(number_outp)]
            expectd[r[-1]] = 1
            bckward_propgate_err(netwrk, expectd)
            sum_err += sum([(expectd[i] - outp[i]) ** 2 for i in range(len(expectd))])
            change_weight(netwrk, r, learning_rate)
            errs.append(sum_err)
        print('epoch number=%d, learning rate=%.1f, error=%.1f' % (epch, learning_rate, sum_err))
```

We can make guessing right now. Guess function basically returns the network output that has the highest guessing number for guess. After that we can create a function for back propagation multilayer perceptron part which will return the predicted type of the rice, as you remember osmancık or cammeo

```
def guess(network, r):
    outpt = frward_propgate(network, r)
    return outpt.index(max(outpt))
def back_prop_multi_percep(train, test, learning_rate, number_epoch, number_hidden):
    n_i = len(train[0]) - 1
    n_o = len(set([row[-1] for row in train]))
    netwrk = start_net(n_i, number_hidden, n_o)
    train_net(netwrk, train, learning_rate, number_epoch, n_o)
    guessings = list()
    for r in test:
        guessing = guess(netwrk, r)
        guessings.append(guessing)
    return (guessings)
```

Conclusion

I had tested result for same learning rate which is 0.3 and same hidden layer which 3. For the first example as you can see the epoch number is 3. For the second example epoch number is 10.For the third example only hidden layer value changed to 10.

When we look at the errors, 0 epochs are always terrible. When we look at the second or the third epochs, results are getting better and better for each examples. But as you can see the error difference between epochs are not as much as it used to be. After a some point changes of the errors are too little. This means that we have to choose good values for different examples because then it doesn't make sense for training the 10 hour if we are able to get very similar result with only 10 minute training.

```
epoch number=0, learning rate=0.3, error=927.5
epoch number=1, learning rate=0.3, error=360.5
epoch number=2, learning rate=0.3, error=330.3
Accuracy: 0.06911636045494313
```

Example 1

```
epoch number=0, learning rate=0.3, error=875.2
epoch number=1, learning rate=0.3, error=358.7
epoch number=2, learning rate=0.3, error=330.2
epoch number=3, learning rate=0.3, error=324.6
epoch number=4, learning rate=0.3, error=322.5
epoch number=5, learning rate=0.3, error=321.2
epoch number=6, learning rate=0.3, error=320.3
epoch number=7, learning rate=0.3, error=319.5
epoch number=8, learning rate=0.3, error=318.7
epoch number=9, learning rate=0.3, error=318.0
Accuracy: 0.9291338582677166
```

Example 2

```
epoch number=0, learning rate=0.3, error=941.6
epoch number=1, learning rate=0.3, error=354.9
epoch number=2, learning rate=0.3, error=330.6
epoch number=3, learning rate=0.3, error=325.5
epoch number=4, learning rate=0.3, error=323.4
epoch number=5, learning rate=0.3, error=322.0
epoch number=6, learning rate=0.3, error=320.8
epoch number=7, learning rate=0.3, error=319.8
epoch number=8, learning rate=0.3, error=318.8
epoch number=9, learning rate=0.3, error=317.9
Accuracy: 0.9300087489063867
```

Example 3

SOURCES

[1] https://machinelearningmastery.com/neural-networks-crash-course/

[2] https://medium.com/@AI_with_Kain/understanding-of-multilayer-perceptron-mlp-8f179c4a135f

[3] https://en.wikipedia.org/wiki/Multilayer_perceptron

[4] https://www.allaboutcircuits.com/technical-articles/how-to-create-a-multilayer-perceptron-neural-network-in-python/

[5] https://medium.com/engineer-quant/multilayer-perceptron-4453615c4337