NN_MarkusWiktorin_131117

November 18, 2017

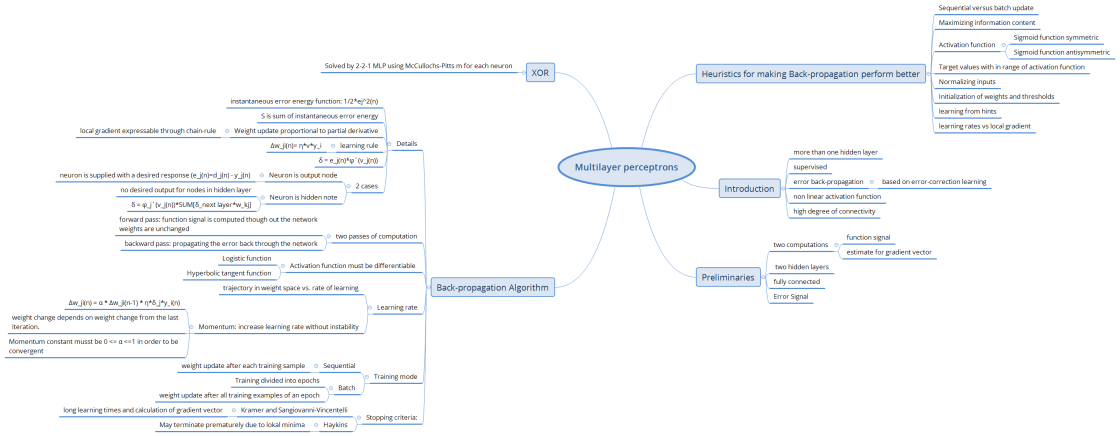
1 Assignment 6

Linda Koine, Jens Weimann, Markus Wiktorin

1.1 1

```
In [31]: from IPython.display import Image
         Image("Multilayer_perceptrons.png")
```

Out[31]:



1.2 2

```
In [32]: import numpy as np
         import sympy as sp
         import matplotlib.pyplot as plt
         from sympy.plotting import plot
```

```
In [33]: alpha=1

         # high alpha makes derivative zero.. no weight update possible
```

```
def sigmoidFunction(x):
    return 1/(1+np.exp(-alpha*x))

def sigmoidFunction2(x):
    return  np.tanh((alpha*x)/2)

def derivationSigmoidFunction(x):
    return alpha*sp.exp(-alpha*x)/(1 + sp.exp(-alpha*x))**2

def derivationSigmoidFunction2(x):
    return -(alpha/2)*np.tanh((alpha/2)*x)**2 + (alpha/2)


def setWeightMatrix(MLVector):
    weightMatrix=np.zeros((np.max(MLVector)**2, np.shape(MLVector)[0]-1))
    for i in range(np.shape(MLVector)[0]):
        if(i+1<np.shape(MLVector)[0]):
            b=0;
            for j in range(MLVector.item(i)):
                for z in range(MLVector.item(i+1)):
                    a=initialWeight(1)
                    weightMatrix[b,i]= a
                    b=b+1
    return weightMatrix


def initialWeight(setting):
    if(setting==1):
        return np.random.randint(1,5)
    return 0

def getLocalField(MLVector, weightMatrix, inputX, outputY,bias):

    if(MLVector.item(0)<np.shape(inputX)[0]):
        print(" input vector x to big vor NN")
        print(inputX)
        return 0
    if(MLVector.item(0)>np.shape(inputX)[0]):
        print(" input vector x to small vor NN")
        return 0
    else:

        localField=np.zeros((np.max(MLVector), np.shape(MLVector)[0]-1))

        for i in range(MLVector.item(0)):
            outputY[i,0]=inputX.item(i)
```

```
        for i in range(1, np.shape(MLVector)[0]):
            a=0;
            for j in range(MLVector.item(i-1)):

                for z in range(MLVector.item(i)):
                    localField[z,i-1]= localField[z,i-1]\
                        + ((outputY[j,i-1])* weightMatrix[a,i-1])
                    a=a+1


            for j in range(MLVector.item(i)):
                localField[j,i-1]= localField[j,i-1] +bias[j,i-1]
                outputY[j,i]=sigmoidFunction2(localField[j,i-1])
        return localField


    def backProp(MLVector, localField, weightMatrix, outputY, desiredOutput, learningRate

        delta=np.zeros((np.max(MLVector), np.shape(MLVector)[0]-1))
        a=0;
        for i in range(MLVector.item(np.shape(MLVector)[0]-1)):

            for j in range(MLVector.item(np.shape(MLVector)[0]-2)):
                #print("i",i)
                y=outputY[i,np.shape(MLVector)[0]-1]
                #print("output:",y)
                delta[j,np.shape(MLVector)[0]-2]=\
                    -((alpha/2)*(1-y**2))*(desiredOutput[i]-y)
                #Wikipedia : y-d
                #print("delta matrix :")
                #print(delta)
                #print("j",j)
                #print("input in node j",outputY[j,np.shape(MLVector)[0]-2])
                #print("delta j",delta[i,np.shape(MLVector)[0]-2])
                #print("weight vor änderung",weightMatrix[a,np.shape(MLVector)[0]-2])
                weightMatrix[a,np.shape(MLVector)[0]-2]=\
                    weightMatrix[a,np.shape(MLVector)[0]-2]\
                    - (learningRate*delta[i,np.shape(MLVector)[0]-2]\
                    * outputY[j,np.shape(MLVector)[0]-2])
                #print("weight nach änderung",weightMatrix[a,np.shape(MLVector)[0]-2])
                a=a+1

        for i in reversed(range(np.shape(MLVector)[0]-2)):
            a=0
            pos=0
            for j in range(MLVector.item(i)):
                mulDeltaWeights=0;
```

```
                    for y in range(MLVector.item(i+1)):
                        #print("delta:",delta[y,i+1])
                        mulDeltaWeights=mulDeltaWeights+ (delta[y,i+1]* weightMatrix[a,i])
                        a=a+1
                    delta[j,i]=-((alpha/2)*(1-outputY[j,i+1]**2))*mulDeltaWeights


                    for y in range(MLVector.item(i+1)):
                        weightMatrix[pos,i]=weightMatrix[pos,i]\
                            - (learningRate*delta[j,i]*outputY[y,i])
                        #print("ÄNDERUNG hidden layer",(learningRate*delta[j,i]*outputY[y,i]))
                        pos=pos+1
```

In [34]:
```
MLVector=np.array([[2],[2],[1]])
learningRate=0.2
weightMatrix=setWeightMatrix(MLVector)

#weightMatrix=np.array([[ 1, 1],[ -1 , 0.9],[ -1 , 0.],[ 1 , 0.]])
bias=np.array([[ -1, 0.1 ],[   -1  , 0 ],[   -1  , 0 ]])
#bias=np.array([[ 0, 0 ,0,0,0],[  0, 0 ,0,0,0 ],[  0, 0 ,0,0,0 ]])

print("initial weight matrix:")
print(weightMatrix)
inputX= np.array([[-1,1],[1,-1],[-1,-1],[1,1]])
desiredOutput= np.array([[1],[1],[-1],[-1]])

error=np.zeros(( 100))

j=0
for i in range(0,100):

    y=np.random.randint(0,4)
    outputY=np.zeros((np.max(MLVector), np.shape(MLVector)[0]))
    localField = getLocalField(MLVector, weightMatrix, inputX[y], outputY,bias)
    #print("ITERATION", i)
    #print("outp: ")
    #print(outputY)
    #print("local Field: ")
    #print(localField)
    backProp(MLVector, localField, weightMatrix,  outputY,\
             desiredOutput[y], learningRate)
    #print("weightMatrix")
    #print(weightMatrix)

    for j in range(0,4):
        outputY=np.zeros((np.max(MLVector), np.shape(MLVector)[0]))
        localField = getLocalField(MLVector, weightMatrix, inputX[j], outputY,bias)
        if(abs(desiredOutput[j] - outputY[0,2]) > 0.1):
```

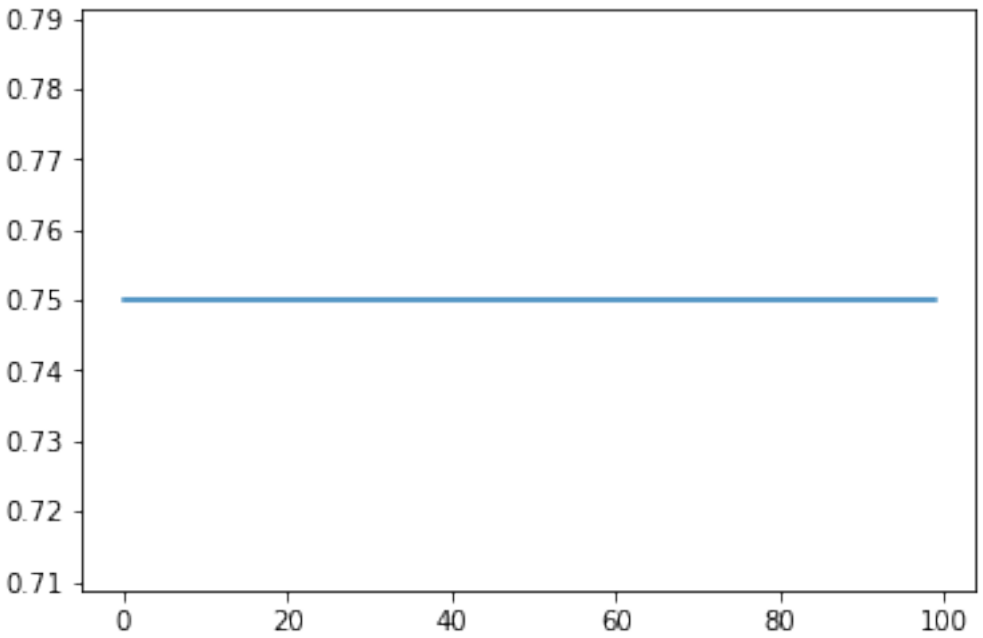
```
                    error[i]= error[i]+1


            error[i]= error[i]/4


        print(" weight matrix after training 10000 times with learning rate 0.2:")
        print(weightMatrix)

        plt.plot(error)
        plt.show()
```

```
initial weight matrix:
[[ 4.  4.]
 [ 2.  4.]
 [ 1.  0.]
 [ 3.  0.]]
 weight matrix after training 10000 times with learning rate 0.2:
[[ 1.8718315   4.64895213]
 [ 5.27464595  2.9340456 ]
 [-0.50606572  0.        ]
 [ 4.50744243  0.        ]]
```



1.3 3

In [35]:
```
import neurolab as nl
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [36]:
```
def func1(x):
    return 1/x

def func2(x):
    return np.log10(x)

def func3(x):
    return np.exp(-x)

def func4(x):
    return np.sin(x)
```

In [37]:
```
class mapping:
    def __init__(self, function, range_, str_):
        self.function = function
        self.range_ = range_
        self.str = str_

    def generate_set(self, size):
        result = []
        for i in range(size):
            x = np.random.random() * (self.range_[1] - self.range_[0]) + self.range_[
            result.append(np.array([x, self.function(x)]))
        return np.array((result))
```

In [38]:
```
def get_error(net, test_set):
    error = 0
    for test in test_set:
        error = error + abs(test[1] - net.sim(np.array([[test[0]]])))[0][0]
    return error / np.size(test_set, 0)
```

In [39]:
```
training_size = 100
test_size = 20
hidden_neurons = [1, 2, 3, 4, 5, 10, 20]

mappings = []
mappings.append(mapping(func1, [1,100], "1/x"))
mappings.append(mapping(func2, [1,10], "log_10(x)"))
mappings.append(mapping(func3, [1,10], "exp(-x)"))
mappings.append(mapping(func4, [1, np.pi / 2], "sin(x)"))

for m in mappings:
```

```
        training_set = m.generate_set(training_size)
        test_set = m.generate_set(test_size)
        input = training_set[:,0].reshape(training_size, 1)
        target = training_set[:,1].reshape(training_size, 1)

        errors = np.zeros((len(hidden_neurons), 1))

        hn_idx = 0
        for num_hidden_neurons in hidden_neurons:
            net = nl.net.newff([m.range_], [num_hidden_neurons,1])
            net.train(input, target, epochs=training_size, show=0)
            errors[hn_idx] = get_error(net, test_set)
            hn_idx = hn_idx + 1

        _ = plt.plot(errors)

    _ = plt.legend([m.str for m in mappings], loc=1)
    _ = plt.xticks(range(len(hidden_neurons)), hidden_neurons)
    _ = plt.xlabel("Number of hidden neurons")
    _ = plt.ylabel("Error")
```