backprop_example

May 4, 2022

0.1 EXAMPLES

0.1.1 How to use this?

It's strongly recommend to use the ***backprop_example.ipynb*** file to check the correctness of two examples..

I provide all solutions from the txt file in the #comment, and my code include the print the solutions.

It's also okay to just run the ***example.py*** file, the output might be slightly messy, but it contains all the information needed.

Even though I include most function import from utils, neuralnetwork, etc, it's still recommend to have download all files.

0.1.2 Back Propagation Example 1

[ ]:
```
import numpy as np
from utils import *
from stratified import *
from neuralnetwork import *
```

Forward Propagate

[ ]:
```
def g(x): # sigmoid function
    return 1/(1 + np.exp(-x))
```

[ ]:
```
# Theta 1
# 0.40000  0.10000
# 0.30000  0.20000
theta1 = np.array([[0.4, 0.1],[0.3,0.2]])
# Theta 2
#   0.70000  0.50000  0.60000
theta2 = np.array([0.7,0.5,0.6])
weightlist1= [theta1,theta2]
```

[ ]:
```
# Training set
#         Training instance 1
#                 x: [0.13000]
#                 y: [0.90000]
```

```
#         Training instance 2
#                 x: [0.42000]
#                 y: [0.23000]
# Training instance 1
trainingcategory = {'x1':'numerical', 'y':'class_numerical'}
trainingdata1 = np.array([0.13,0.9])
trainingdata2 = np.array([0.42,0.23])
inputdata1 = np.append(1,trainingdata1[0])
inputdata2 = np.append(1,trainingdata2[0])
exceptout1 = trainingdata1[1]
exceptout2 = trainingdata2[1]
lambda1 = 0
```

[ ]:
```
def costfunction(expected_output, actual_output):
    j = -np.multiply(expected_output,np.log(actual_output)) - np.multiply((1 -␣
 ↪expected_output),np.log(1 - actual_output))
    return np.sum(j)
```

[ ]:
```
def forwardtest(inputdata,weightl,expectedout):
    current_layer_a = inputdata
    print('current_a at 1 is',current_layer_a)
    current_layer_index = 0
    alist = []
    alist.append(current_layer_a)
    for theta in weightl:
        z = np.dot(theta,current_layer_a)
        a = g(z)
        current_layer_a = np.append(1,a) if (current_layer_index+1 !=␣
 ↪len(weightl)) else a
        print('current_a at',current_layer_index+2,'is',current_layer_a)
        alist.append(current_layer_a)
        current_layer_index += 1
    result = current_layer_a
    print('prediction is', result)
    print('exceptout is', expectedout)
    print('cost is', costfunction(expectedout,result))
    return result, costfunction(expectedout,result), alist
```

[ ]:
```
r1,j1,a1 = forwardtest(inputdata1,weightlist1,exceptout1)
# Computing the error/cost, J, of the network
#         Processing training instance 1
#         Forward propagating the input [0.13000]
#                 a1: [1.00000   0.13000]

#                 z2: [0.41300   0.32600]
#                 a2: [1.00000   0.60181   0.58079]
```

```
#                 z3: [1.34937]
#                 a3: [0.79403]

#                 f(x): [0.79403]
#         Predicted output for instance 1: [0.79403]
#         Expected output for instance 1: [0.90000]
#         Cost, J, associated with instance 1: 0.366
```

```
current_a at 1 is [1.   0.13]
current_a at 2 is [1.        0.601807  0.5807858]
current_a at 3 is 0.7940274264318581
prediction is 0.7940274264318581
exceptout is 0.9
cost is 0.36557477431084995
```

[ ]:
```
r2,j2,a2 = forwardtest(inputdata2,weightlist1,exceptout2)
        # Processing training instance 2
        # Forward propagating the input [0.42000]
        #         a1: [1.00000   0.42000]

        #         z2: [0.44200   0.38400]
        #         a2: [1.00000   0.60874   0.59484]

        #         z3: [1.36127]
        #         a3: [0.79597]

        #         f(x): [0.79597]
        # Predicted output for instance 2: [0.79597]
        # Expected output for instance 2: [0.23000]
        # Cost, J, associated with instance 2: 1.276
```

```
current_a at 1 is [1.   0.42]
current_a at 2 is [1.         0.60873549 0.59483749]
current_a at 3 is 0.7959660671522611
prediction is 0.7959660671522611
exceptout is 0.23
cost is 1.2763768066887786
```

[ ]:
```
jlist1 = np.array([j1,j2])
numberofinstance1 = 2
```

[ ]:
```
def overallcost(jlist,n,weightl,lambda_reg):
    s = sumofweights(weightl,bias=0)*lambda_reg/(2*n)
    jsum = np.sum(jlist)
    return jsum/n + s
```

[ ]:
```
overallcost(jlist1,numberofinstance1,weightlist1,lambda1)
# Final (regularized) cost, J, based on the complete training set: 0.82098
```

[ ]: 0.8209757904998143

Back Propagate

[ ]:
```
def delta(weightl,alist,expect,actual):
    delta_layer_n = actual-expect
    deltalist = []
    deltalist.append(delta_layer_n)
    i = len(weightl)-1
    current_delta = delta_layer_n
    while i > 0:
        delta_layer_now = np.multiply(np.multiply(np.dot(weightl[i].
 ↪T,current_delta),alist[i]),(1-alist[i]))
        current_delta = delta_layer_now[1:]
        deltalist.append(current_delta)
        i-=1
    deltalist.reverse()
    return deltalist
```

[ ]:
```
def gradientD(weights_list,delta_list,a_list,biasterm=True):
    gradlist = []
    for i in range(len(weights_list)):
        anow = a_list[i]
        deltanow = np.array([delta_list[i]]).T
        dotproduct = deltanow*anow
        # print('dotshape',dotproduct.shape)
        gradlist.append(dotproduct)
    return gradlist
```

[ ]:
```
delta1_1 = delta(weightlist1,a1,exceptout1,r1)
        # Computing gradients based on training instance 1
        #          delta3: [-0.10597]
        #          delta2: [-0.01270   -0.01548]
print(delta1_1)
```

```
[array([-0.01269739, -0.01548092]), -0.10597257356814194]
```

[ ]:
```
                # Gradients of Theta2 based on training instance 1:
                #          -0.10597  -0.06378  -0.06155

                # Gradients of Theta1 based on training instance 1:
                #          -0.01270  -0.00165
                #          -0.01548  -0.00201
gradd1_1 = gradientD(weightlist1,delta1_1,a1)
print(gradd1_1)
```

```
[array([[-0.01269739, -0.00165066],
       [-0.01548092, -0.00201252]]), array([-0.10597257, -0.06377504,
-0.06154737])]
```

[ ]:
```python
delta1_2 = delta(weightlist1,a2,exceptout2,r2)
        # Computing gradients based on training instance 2
        #         delta3: [0.56597]
        #         delta2: [0.06740   0.08184]
print(delta1_2)
```

```
[array([0.06739994, 0.08184068]), 0.5659660671522612]
```

[ ]:
```python
                # Gradients of Theta2 based on training instance 2:
                #         0.56597  0.34452  0.33666

                # Gradients of Theta1 based on training instance 2:
                #         0.06740  0.02831
                #         0.08184  0.03437
gradd1_2 = gradientD(weightlist1,delta1_2,a2)
print(gradd1_2)
```

```
[array([[0.06739994, 0.02830797],
       [0.08184068, 0.03437309]]), array([0.56596607, 0.34452363, 0.33665784])]
```

[ ]:
```python
def transposelistoflist(l):
    newlistoflist = []
    for i in range(len(l[0])):
        newlist = []
        for j in range(len(l)):
            newlist.append(l[j][i])
        newlistoflist.append(newlist)
    return newlistoflist
```

[ ]:
```python
listofgradient = [gradd1_1,gradd1_2]
gradientP1 = [lambda1*t for t in weightlist1]
grad_D_transpose = transposelistoflist(listofgradient)
grad_D_sum = [np.sum(t,axis=0) for t in grad_D_transpose]
update_gradients = []
for i in range(len(grad_D_sum)):
    update_gradients.append((grad_D_sum[i] + gradientP1[i])*(1/
 ↪numberofinstance1))
```

[ ]:
```python
print(update_gradients)
        # The entire training set has been processes. Computing the average␣
 ↪(regularized) gradients:
        #         Final regularized gradients of Theta1:
        #                 0.02735  0.01333
        #                 0.03318  0.01618

        #         Final regularized gradients of Theta2:
        #                 0.23000  0.14037  0.13756
```

```
[array([[0.02735127, 0.01332866],
```

```
[0.03317988, 0.01618028]]), array([0.22999675, 0.1403743 , 0.13755523])]
```

0.1.3 Back Propagation Example 2

Forward Propagrate

```
[ ]: # Initial Theta1 (the weights of each neuron, including the bias weight, are
     ↪stored in the rows):
     #         0.42000  0.15000  0.40000
     #         0.72000  0.10000  0.54000
     #         0.01000  0.19000  0.42000
     #         0.30000  0.35000  0.68000

     # Initial Theta2 (the weights of each neuron, including the bias weight, are
     ↪stored in the rows):
     #         0.21000  0.67000  0.14000  0.96000  0.87000
     #         0.87000  0.42000  0.20000  0.32000  0.89000
     #         0.03000  0.56000  0.80000  0.69000  0.09000

     # Initial Theta3 (the weights of each neuron, including the bias weight, are
     ↪stored in the rows):
     #         0.04000  0.87000  0.42000  0.53000
     #         0.17000  0.10000  0.95000  0.69000
     e2theta1 = np.array([[0.42,0.15,0.4],[0.72,0.1,0.54],[0.01,0.19,0.42],[0.3,0.
     ↪35,0.68]])
     e2theta2 = np.array([[0.21,0.67,0.14,0.96,0.87],[0.87,0.42,0.2,0.32,0.89],[0.
     ↪03,0.56,0.8,0.69,0.09]])
     e2theta3 = np.array([[0.04,0.87,0.42,0.53],[0.17,0.1,0.95,0.69]])
     e2weightlist = [e2theta1,e2theta2,e2theta3]
```

```
[ ]: # Training set
     #         Training instance 1
     #                 x: [0.32000   0.68000]
     #                 y: [0.75000   0.98000]
     #         Training instance 2
     #                 x: [0.83000   0.02000]
     #                 y: [0.75000   0.28000]

     e2input1 = np.array([0.32,0.68])
     e2input2 = np.array([0.83,0.02])
     e2exceptout1 = np.array([0.75,0.98])
     e2exceptout2 = np.array([0.75,0.28])

     e2input1 = np.append(1,e2input1)
     e2input2 = np.append(1,e2input2)
     e2lambda0 = 0.25
```

[ ]:
```
e2r1,e2j1,e2a1 = forwardtest(e2input1,e2weightlist,e2exceptout1)
        # Processing training instance 1
        # Forward propagating the input [0.32000   0.68000]
        #         a1: [1.00000   0.32000   0.68000]

        #         z2: [0.74000   1.11920   0.35640   0.87440]
        #         a2: [1.00000   0.67700   0.75384   0.58817   0.70566]

        #         z3: [1.94769   2.12136   1.48154]
        #         a3: [1.00000   0.87519   0.89296   0.81480]

        #         z4: [1.60831   1.66805]
        #         a4: [0.83318   0.84132]

        #         f(x): [0.83318   0.84132]
        # Predicted output for instance 1: [0.83318   0.84132]
        # Expected output for instance 1: [0.75000   0.98000]
        # Cost, J, associated with instance 1: 0.791
```

```
current_a at 1 is [1.   0.32 0.68]
current_a at 2 is [1.         0.67699586 0.75384029 0.5881687  0.70566042]
current_a at 3 is [1.         0.87519469 0.89296181 0.81480444]
current_a at 4 is [0.83317658 0.84131543]
prediction is [0.83317658 0.84131543]
exceptout is [0.75 0.98]
cost is 0.7907366961135718
```

[ ]:
```
e2r2,e2j2,e2a2 = forwardtest(e2input2,e2weightlist,e2exceptout2)
        # Processing training instance 2
        # Forward propagating the input [0.83000   0.02000]
        #         a1: [1.00000   0.83000   0.02000]

        #         z2: [0.55250   0.81380   0.17610   0.60410]
        #         a2: [1.00000   0.63472   0.69292   0.54391   0.64659]

        #         z3: [1.81696   2.02468   1.37327]
        #         a3: [1.00000   0.86020   0.88336   0.79791]

        #         z4: [1.58228   1.64577]
        #         a4: [0.82953   0.83832]

        #         f(x): [0.82953   0.83832]
        # Predicted output for instance 2: [0.82953   0.83832]
        # Expected output for instance 2: [0.75000   0.28000]
        # Cost, J, associated with instance 2: 1.944
```

```
current_a at 1 is [1.   0.83 0.02]
current_a at 2 is [1.         0.63471542 0.69291867 0.54391158 0.64659376]
```

```
current_a at 3 is [1.         0.86020091 0.88336451 0.79790763]
current_a at 4 is [0.82952703 0.83831889]
prediction is [0.82952703 0.83831889]
exceptout is [0.75 0.28]
cost is 1.9437823352945296
```

[ ]:
```python
e2jlist = np.array([e2j1,e2j2])
e2numberofinstance = 2
```

[ ]:
```python
def sumofweights(listofweights,bias=True): # computes the square of all weights␣
 ↪of the network and sum them up
    sum = 0
    for weight in listofweights:
        if bias:
            w = weight.copy()
            w[:, 0] = 0
            sum += np.sum(np.square(w))
        else:
            sum += np.sum(np.square(weight))
    return sum
```

[ ]:
```python
def overallcost(jlist,n,weightl,lambda_reg):
    s = sumofweights(weightl,bias=1)*lambda_reg/(2*n)
    jsum = np.sum(jlist)
    return jsum/n + s
```

[ ]:
```python
overallcost(e2jlist,e2numberofinstance,e2weightlist,e2lambda0)
# Final (regularized) cost, J, based on the complete training set: 1.90351
```

[ ]:
```
1.9035095157040507
```

Back Propagation for E2

[ ]:
```python
e2delta1 = delta(e2weightlist,e2a1,e2exceptout1,e2r1)
# Running backpropagation
#         Computing gradients based on training instance 1
#                 delta4: [0.08318   -0.13868]
#                 delta3: [0.00639   -0.00925   -0.00779]
#                 delta2: [-0.00087   -0.00133   -0.00053   -0.00070]
print(e2delta1)
```

```
[array([-0.00086743, -0.00133354, -0.00053312, -0.00070163]), array([
0.00638937, -0.00925379, -0.00778767]), array([ 0.08317658, -0.13868457])]
```

[ ]:
```python
                 # Gradients of Theta3 based on training instance 1:
                 #         0.08318  0.07280  0.07427  0.06777
                 #         -0.13868  -0.12138  -0.12384  -0.11300

                 # Gradients of Theta2 based on training instance 1:
```

```
                    #         0.00639  0.00433  0.00482  0.00376  0.00451
                    #         -0.00925  -0.00626  -0.00698  -0.00544  -0.00653
                    #         -0.00779  -0.00527  -0.00587  -0.00458  -0.00550

                    # Gradients of Theta1 based on training instance 1:
                    #         -0.00087  -0.00028  -0.00059
                    #         -0.00133  -0.00043  -0.00091
                    #         -0.00053  -0.00017  -0.00036
                    #         -0.00070  -0.00022  -0.00048
e2grad1 = gradientD(e2weightlist,e2delta1,e2a1)
print(e2grad1)
```

```
[array([[-0.00086743, -0.00027758, -0.00058985],
       [-0.00133354, -0.00042673, -0.00090681],
       [-0.00053312, -0.0001706 , -0.00036252],
       [-0.00070163, -0.00022452, -0.00047711]]), array([[ 0.00638937,
0.00432557,  0.00481656,  0.00375802,  0.00450872],
       [-0.00925379, -0.00626478, -0.00697588, -0.00544279, -0.00653003],
       [-0.00778767, -0.00527222, -0.00587066, -0.00458046, -0.00549545]]),
array([[ 0.08317658,  0.0727957 ,  0.07427351,  0.06777264],
       [-0.13868457, -0.121376  , -0.12384003, -0.1130008 ]])]
```

[ ]:
```
e2delta2 = delta(e2weightlist,e2a2,e2exceptout2,e2r2)
        # Computing gradients based on training instance 2
        #         delta4: [0.07953   0.55832]
        #         delta3: [0.01503   0.05809   0.06892]
        #         delta2: [0.01694   0.01465   0.01999   0.01622]
print(e2delta2)
```

```
[array([0.01694006, 0.01465141, 0.01998824, 0.01622017]), array([0.01503437,
0.05808969, 0.06891698]), array([0.07952703, 0.55831889])]
```

[ ]:
```
                    # Gradients of Theta3 based on training instance 2:
                    #         0.07953  0.06841  0.07025  0.06346
                    #         0.55832  0.48027  0.49320  0.44549

                    # Gradients of Theta2 based on training instance 2:
                    #         0.01503  0.00954  0.01042  0.00818  0.00972
                    #         0.05809  0.03687  0.04025  0.03160  0.03756
                    #         0.06892  0.04374  0.04775  0.03748  0.04456

                    # Gradients of Theta1 based on training instance 2:
                    #         0.01694  0.01406  0.00034
                    #         0.01465  0.01216  0.00029
                    #         0.01999  0.01659  0.00040
                    #         0.01622  0.01346  0.00032
e2grad2 = gradientD(e2weightlist,e2delta2,e2a2)
print(e2grad2)
```

```
[array([[0.01694006, 0.01406025, 0.0003388 ],
       [0.01465141, 0.01216067, 0.00029303],
       [0.01998824, 0.01659024, 0.00039976],
       [0.01622017, 0.01346274, 0.0003244 ]]), array([[0.01503437, 0.00954254,
0.01041759, 0.00817737, 0.00972113],
       [0.05808969, 0.03687042, 0.04025143, 0.03159565, 0.03756043],
       [0.06891698, 0.04374267, 0.04775386, 0.03748474, 0.04456129]]),
array([[0.07952703, 0.06840922, 0.07025135, 0.06345522],
       [0.55831889, 0.48026642, 0.4931991 , 0.44548691]])]
```

[ ]:
```
e2listofgradient = [e2grad1,e2grad2]
gradientP2 = [e2lambda0*t for t in e2weightlist]
for singleP in gradientP2:
    singleP[:, 0] = 0
e2_grad_D_transpose = transposelistoflist(e2listofgradient)
e2_grad_D_sum = [np.sum(t,axis=0) for t in e2_grad_D_transpose]
e2_update_gradients = []
for i in range(len(grad_D_sum)):
    e2_update_gradients.append((e2_grad_D_sum[i] + gradientP2[i])*(1/
 ↪e2numberofinstance))
```

[ ]:
```
print(e2_update_gradients)
        # The entire training set has been processes. Computing the average␣
 ↪(regularized) gradients:
        #         Final regularized gradients of Theta1:
        #                 0.00804  0.02564  0.04987
        #                 0.00666  0.01837  0.06719
        #                 0.00973  0.03196  0.05252
        #                 0.00776  0.05037  0.08492

        #         Final regularized gradients of Theta2:
        #                 0.01071  0.09068  0.02512  0.12597  0.11586
        #                 0.02442  0.06780  0.04164  0.05308  0.12677
        #                 0.03056  0.08924  0.12094  0.10270  0.03078

        #         Final regularized gradients of Theta3:
        #                 0.08135  0.17935  0.12476  0.13186
        #                 0.20982  0.19195  0.30343  0.25249
```

```
[array([[0.00803632, 0.02564134, 0.04987447],
       [0.00665894, 0.01836697, 0.06719311],
       [0.00972756, 0.03195982, 0.05251862],
       [0.00775927, 0.05036911, 0.08492365]]), array([[0.01071187, 0.09068406,
0.02511708, 0.1259677 , 0.11586492],
       [0.02441795, 0.06780282, 0.04163777, 0.05307643, 0.1267652 ],
       [0.03056466, 0.08923522, 0.1209416 , 0.10270214, 0.03078292]])]
```