

---

**1: The First Problem**


---

**(a) Algorithm:**

```
import numpy as np
```

```
def tanh(x):
    return np.tanh(x)
```

```
def tanh_deriv(x):
    return 1.0 - np.tanh(x) * np.tanh(x)
```

```
class MLP:
```

```
    def __init__(self, layers):
        self.layers = layers
        self.activation = tanh
        self.activation_deriv = tanh_deriv
        self.num_layers = len(layers)
        self.bias = [np.random.randn(x) for x in layers[1:]]
        self.weights = [np.random.randn(y, x) for x, y in zip(layers[:-1], layers[1:])]
```

```
    def fit(self, X, y, learning_rate, epochs):
        for k in range(epochs):
            for i in range(len(X)):
                #store to use
                results = [X[i]]
                for b, w in zip(self.bias, self.weights):
                    z = np.dot(w, results[-1]) + b
                    output = self.activation(z)
                    results.append(output)

                error = y[i] - results[-1]
                deltas = [error * self.activation_deriv(results[-1])]

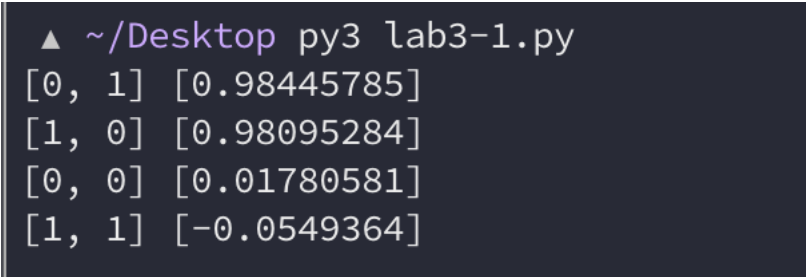
                for l in range(self.num_layers-2, 0, -1):
                    deltas.append(self.activation_deriv(results[l]) * np.dot(deltas[-1], self.weights[l+1].T))

                deltas.reverse()

                for j in range(self.num_layers-1):
                    layers = np.array(results[j])
                    delta_w = learning_rate * ((np.atleast_2d(deltas[j]).T).dot(np.atleast_2d(layers)))
                    self.weights[j] += delta_w
                    delta_t = learning_rate * deltas[j]
                    self.bias[j] += delta_t
```

```
def predict(self, x):
    for b, w in zip(self.bias, self.weights):
        z = np.dot(w, x) + b
        x = self.activation(z)
    return x
```

```
mlp = MLP([2,4,3,1])
X = np.array([[0, 1], [1, 0], [0, 0], [1, 1]])
y = np.array([1, 1, 0, 0])
mlp.fit(X, y, 0.05, epochs=1000)
for i in [[0, 1], [1, 0], [0, 0], [1, 1]]:
    print(i, mlp.predict(i))
```

**(b) Output:**


```
▲ ~/Desktop py3 lab3-1.py
[0, 1] [0.98445785]
[1, 0] [0.98095284]
[0, 0] [0.01780581]
[1, 1] [-0.0549364]
```

---

**2: The Second Problem**

---

**(a) Algorithm:**

```
import csv
import random
import math
random.seed(250)

with open('iris.csv') as csvfile:
    csvreader = csv.reader(csvfile)
    next(csvreader, None)
    dataset = list(csvreader)

for row in dataset:
    row[4] = ["Iris-setosa", "Iris-versicolor", "Iris-virginica"].index(row[4])
    row[:4] = [float(row[j]) for j in range(len(row))]

random.shuffle(dataset)
datatrain = dataset[:int(len(dataset) * 0.8)]
datatest = dataset[int(len(dataset) * 0.8):]
train_X = [data[:4] for data in datatrain]
```

```

train_y = [data[4] for data in datatrain]
test_X = [data[:4] for data in datatest]
test_y = [data[4] for data in datatest]

def matrix_mul_bias(A, B, bias):
    C = [[0 for i in range(len(B[0]))] for i in range(len(A))]
    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                C[i][j] += A[i][k] * B[k][j]
            C[i][j] += bias[j]
    return C

def vec_mat_bias(A, B, bias):
    C = [0 for i in range(len(B[0]))]
    for j in range(len(B[0])):
        for k in range(len(B)):
            C[j] += A[k] * B[k][j]
        C[j] += bias[j]
    return C

def mat_vec(A, B):
    C = [0 for i in range(len(A))]
    for i in range(len(A)):
        for j in range(len(B)):
            C[i] += A[i][j] * B[j]
    return C

def sigmoid(A, deriv=False):
    if deriv:
        for i in range(len(A)):
            A[i] = A[i] * (1 - A[i])
    else:
        for i in range(len(A)):
            A[i] = 1 / (1 + math.exp(-A[i]))
    return A

alfa = 0.005
epoch = 2000
# [input layer:feature of Iris, hidden layer, output layer:class of Iris]
neuron = [4, 4, 3]
weight = [[0 for j in range(neuron[1])] for i in range(neuron[0])]
weight_2 = [[0 for j in range(neuron[2])] for i in range(neuron[1])]
bias = [0 for i in range(neuron[1])]
bias_2 = [0 for i in range(neuron[2])]

```

```

for i in range(neuron[0]):
    for j in range(neuron[1]):
        weight[i][j] = 2 * random.random() - 1

for i in range(neuron[1]):
    for j in range(neuron[2]):
        weight_2[i][j] = 2 * random.random() - 1

for e in range(epoch):
    cost_total = 0
    for idx, x in enumerate(train_X):

        h_1 = vec_mat_bias(x, weight, bias)
        X_1 = sigmoid(h_1)
        h_2 = vec_mat_bias(X_1, weight_2, bias_2)
        X_2 = sigmoid(h_2)

        target = [0, 0, 0]
        target[int(train_y[idx])] = 1

        error = 0
        for i in range(neuron[2]):
            error += (target[i] - X_2[i]) ** 2
        cost_total += error * 1 / neuron[2]

        delta_2 = []
        for j in range(neuron[2]):
            delta_2.append(-1 * 2. / neuron[2] * (target[j] - X_2[j]) * X_2[j] * (1 - X_2[j]))

        for i in range(neuron[1]):
            for j in range(neuron[2]):
                weight_2[i][j] -= alfa * (delta_2[j] * X_1[i])
                bias_2[j] -= alfa * delta_2[j]

        delta_1 = mat_vec(weight_2, delta_2)
        for j in range(neuron[1]):
            delta_1[j] = delta_1[j] * (X_1[j] * (1 - X_1[j]))

        for i in range(neuron[0]):
            for j in range(neuron[1]):
                weight[i][j] -= alfa * (delta_1[j] * x[i])
                bias[j] -= alfa * delta_1[j]

    cost_total /= len(train_X)
    if(e % 100 == 0):
        print(cost_total)

```

```

#Test
result = matrix_mul_bias(test_X, weight, bias)
output = matrix_mul_bias(result, weight_2, bias)

predict = []
for r in output:
    predict.append(max(enumerate(r), key=lambda x:x[1])[0])
print(predict)

acc = 0.0
for i in range(len(predict)):
    if predict[i] == int(test_y[i]):
        acc += 1
print(acc / len(predict) * 100, "%")

print("weight_2: ", weight_2)
print("bias_2: ", bias_2)
print("weight: ", weight)
print("bias: ", bias)

```

**(b) Output:**

```

▲ ~/Desktop py3 lab3-2.py
0.3011447188963846
0.15459838905028134
0.13636664166400167
0.1281939032283599
0.12386403775714513
0.12127217524429423
0.11957698780766658
0.11839177337838645
0.11751838215837747
0.11684607364656774
[2, 2, 2, 0, 2, 2, 2, 2, 0, 0, 0, 2, 2, 0, 2, 0, 0, 0, 2, 2, 2, 2, 2, 0,
 2, 0, 2, 2]
70.0 %
weight_2: [[-0.6948748902761284, -0.6720292802532452, 0.5995607419626166], [0
.6604386246887691, -0.4210452788143647, 0.6452858628401181], [0.31163386575300
317, -0.738625612405973, 0.17631938523420462], [4.871569596318834, -2.09298676
55020957, -3.120304750906864]]
bias_2: [-0.582112424283751, 0.3589134223113549, -0.18818201374626414]
weight: [[0.8570150137394552, -0.391521725047482, 0.5415733216146104, 0.22756
457848728703], [0.28352101759922155, -1.0916363472527504, 0.6993620492897716,
1.4141183300030058], [0.6967326082698332, 0.583603750976725, 0.452234448325052
4, -2.493579536436302], [0.15145670620879348, -1.0659068471879765, 0.475381678
60499275, -0.001992258333983404]]
bias: [-0.02184434462834539, -0.4270801696625457, 0.019949458876588153, 0.513
6168900913907]

```

```
▲ ~/Desktop py3 lab3-2.py
0.26362056277453416
0.16934280990291867
0.13134967964529792
0.12041730869127258
0.11342091570184704
0.10414362893074827
0.09184705025727877
0.0779954590516714
0.06477373474514017
0.05365503640757382
[0, 2, 2, 2, 0, 2, 2, 0, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 2, 2, 2, 2, 0,
 0, 0, 2, 2]
73.33333333333333 %
weight_2: [[2.7169569567425764, 1.9957049628370467, -3.8820190030698845], [-0
.8831709991401138, 1.2824925417267878, -1.1053919790928848], [-0.8194524399278
739, -0.8106032121975791, 0.7718787074472471], [3.756667872557584, -2.97929484
1920487, -1.3060446013797082]]
bias_2: [-0.7208766172051841, -0.24647664317424312, 0.4252771311627085]
weight: [[0.8890044365378295, -0.4040524273936041, 0.62259550404742, 0.054695
545792393684], [1.3878968778154201, -0.2210059962561583, 0.8974078963000592, 1
.1530001106289594], [-2.7653274879504885, 0.32532853174406334, 0.6751183800416
117, -2.276203572577283], [-1.2312200141784995, -1.4756102313506867, 0.6278385
202128228, -1.4316454117700472]]
bias: [1.499507216186553, 1.0862814177097753, -0.008277724465635822, 0.774707
7803709734]
```

```
▲ Assignments/Computational Intelligence/Lab3 py3 lab3-2.py
0.33323926691797706
0.22278767622335363
0.2222918120788896
0.20230851923233567
0.15658643236041564
0.13459163168931054
0.12519927913421952
0.12041874922787428
0.11762407272747341
0.11582472235005171
0.11458291095016235
0.11368004411855402
0.11299639364232893
0.11246148512317435
0.11203136515571795
0.11167726996452075
0.11137957416070851
0.11112436430252418
0.11090139316036048
0.11070279100551038
[2, 0, 0, 1, 1, 1, 2, 2, 1, 2, 2, 0, 2, 0, 2, 1, 2, 2, 1, 2, 1, 0, 1, 2, 0, 0,
 2, 0, 2, 1]
93.33333333333333 %
weight_2: [[0.7508078650705086, 0.5149404370863823, 0.2971437832030401], [0.2
79500074062341, -0.7020831280524199, -0.4029562658319158], [-0.711297550726282
5, -0.6322244576888507, -0.36867713188491325], [-5.516102226464258, 2.41451137
64851813, 3.470371545467875]]
bias_2: [0.44033916274827756, -0.731081061923275, -0.9270890172963717]
weight: [[0.9133710250519014, -0.5544481867878253, -1.0860546324653628, -0.76
45096788012778], [-0.23675667184444535, -0.36374678004217337, 0.17857086888674
584, -0.6745190985374291], [1.0115974358841704, -0.5643490346979001, -0.420033
2006677712, 2.8800541115326364], [0.7236096849974896, -0.20057071046276223, 0.
026552486308061198, 0.3893844359507134]]
bias: [0.05142651932379132, 0.14227307172811007, -0.08281903037732208, -0.669
1211269217752]
```