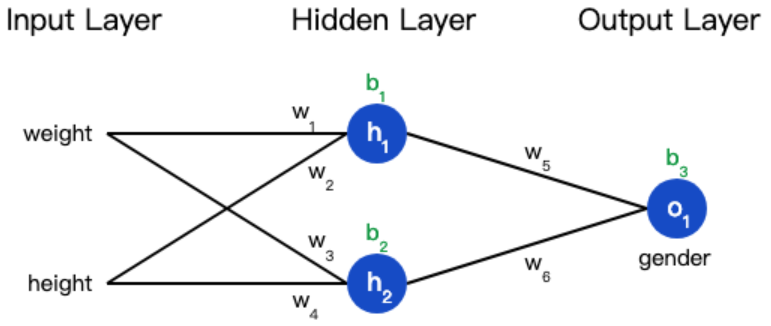
<https://www.zhihu.com/question/314879954>





import numpy as np

def sigmoid(x):

# Sigmoid activation function: f(x) = 1 / (1 + e^(-x))

return 1 / (1 + np.exp(-x))

def deriv\_sigmoid(x):

# Derivative of sigmoid: f'(x) = f(x) \* (1 - f(x))

fx = sigmoid(x)

return fx \* (1 - fx)

def mse\_loss(y\_true, y\_pred):

# y\_true and y\_pred are numpy arrays of the same length.

return ((y\_true - y\_pred) \*\* 2).mean()

class OurNeuralNetwork:

'''

A neural network with:

- 2 inputs

- a hidden layer with 2 neurons (h1, h2)

- an output layer with 1 neuron (o1)

\*\*\* DISCLAIMER \*\*\*:

The code below is intended to be simple and educational, NOT optimal.

Real neural net code looks nothing like this. DO NOT use this code.

Instead, read/run it to understand how this specific network works.

'''

def \_\_init\_\_(self):

# Weights

self.w1 = np.random.normal()

self.w2 = np.random.normal()

self.w3 = np.random.normal()

self.w4 = np.random.normal()

self.w5 = np.random.normal()

self.w6 = np.random.normal()

# Biases

self.b1 = np.random.normal()

self.b2 = np.random.normal()

self.b3 = np.random.normal()

def feedforward(self, x):

# x is a numpy array with 2 elements.

h1 = sigmoid(self.w1 \* x[0] + self.w2 \* x[1] + self.b1)

h2 = sigmoid(self.w3 \* x[0] + self.w4 \* x[1] + self.b2)

o1 = sigmoid(self.w5 \* h1 + self.w6 \* h2 + self.b3)

return o1

def train(self, data, all\_y\_trues):

'''

- data is a (n x 2) numpy array, n = # of samples in the dataset.

- all\_y\_trues is a numpy array with n elements.

Elements in all\_y\_trues correspond to those in data.

'''

learn\_rate = 0.1

epochs = 1000 # number of times to loop through the entire dataset

for epoch in range(epochs):

for x, y\_true in zip(data, all\_y\_trues):

# --- Do a feedforward (we'll need these values later)

sum\_h1 = self.w1 \* x[0] + self.w2 \* x[1] + self.b1

h1 = sigmoid(sum\_h1)

sum\_h2 = self.w3 \* x[0] + self.w4 \* x[1] + self.b2

h2 = sigmoid(sum\_h2)

sum\_o1 = self.w5 \* h1 + self.w6 \* h2 + self.b3

o1 = sigmoid(sum\_o1)

y\_pred = o1

# --- Calculate partial derivatives.

# --- Naming: d\_L\_d\_w1 represents "partial L / partial w1"

d\_L\_d\_ypred = -2 \* (y\_true - y\_pred)

# Neuron o1

d\_ypred\_d\_w5 = h1 \* deriv\_sigmoid(sum\_o1)

d\_ypred\_d\_w6 = h2 \* deriv\_sigmoid(sum\_o1)

d\_ypred\_d\_b3 = deriv\_sigmoid(sum\_o1)

d\_ypred\_d\_h1 = self.w5 \* deriv\_sigmoid(sum\_o1)

d\_ypred\_d\_h2 = self.w6 \* deriv\_sigmoid(sum\_o1)

# Neuron h1

d\_h1\_d\_w1 = x[0] \* deriv\_sigmoid(sum\_h1)

d\_h1\_d\_w2 = x[1] \* deriv\_sigmoid(sum\_h1)

d\_h1\_d\_b1 = deriv\_sigmoid(sum\_h1)

# Neuron h2

d\_h2\_d\_w3 = x[0] \* deriv\_sigmoid(sum\_h2)

d\_h2\_d\_w4 = x[1] \* deriv\_sigmoid(sum\_h2)

d\_h2\_d\_b2 = deriv\_sigmoid(sum\_h2)

# --- Update weights and biases

# Neuron h1

self.w1 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h1 \* d\_h1\_d\_w1

self.w2 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h1 \* d\_h1\_d\_w2

self.b1 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h1 \* d\_h1\_d\_b1

# Neuron h2

self.w3 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h2 \* d\_h2\_d\_w3

self.w4 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h2 \* d\_h2\_d\_w4

self.b2 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h2 \* d\_h2\_d\_b2

# Neuron o1

self.w5 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_w5

self.w6 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_w6

self.b3 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_b3

# --- Calculate total loss at the end of each epoch

if epoch % 10 == 0:

y\_preds = np.apply\_along\_axis(self.feedforward, 1, data)

loss = mse\_loss(all\_y\_trues, y\_preds)

print("Epoch %d loss: %.3f" % (epoch, loss))

# Define dataset

data = np.array([

[-2, -1], # Alice

[25, 6], # Bob

[17, 4], # Charlie

[-15, -6], # Diana

])

all\_y\_trues = np.array([

1, # Alice

0, # Bob

0, # Charlie

1, # Diana

])

# Train our neural network!

network = OurNeuralNetwork()

network.train(data, all\_y\_trues)