1   
if len(x.shape) == 1:

x = x.reshape(-1, 1)

# Step 1. Initialize activation on initial layer to x

self.a[0] = x

# Step 2-4. Loop over layers and compute activities and activations

for l in range(1, self.L):

self.z[l] = np.dot(self.W[l-1], self.a[l-1]) + self.b[l-1]

self.a[l] = self.g(self.z[l])  
  
2  
if len(y.shape) == 1:

y = y.reshape(-1, 1)

# Step 1. forward prop training example to fill in activities and activations

self.forward\_prop(x)

# Step 2. compute deltas on output layer

self.delta[-1] = self.grad\_loss(self.a[-1], y) \* self.g\_prime(self.z[-1])

# Step 3-6. loop backward through layers, backprop deltas, compute dWs and dbs

for l in range(self.L-2, -1, -1):

self.delta[l] = np.dot(self.W[l].T, self.delta[l+1]) \* self.g\_prime(self.z[l])

self.dW[l] = np.dot(self.delta[l+1], self.a[l].T)

self.db[l] = self.delta[l+1]  
  
3  
shuffled\_inds = list(range(X\_train.shape[0]))

for ep in range(num\_epochs):

np.random.shuffle(shuffled\_inds)

for ind in shuffled\_inds:

# back prop to get derivatives

self.back\_prop(X\_train[ind], y\_train[ind])

# update all weights and biases for all layers

for l in range(self.L - 1):

self.W[l] -= eta \* self.dW[l]

self.b[l] -= eta \* self.db[l]