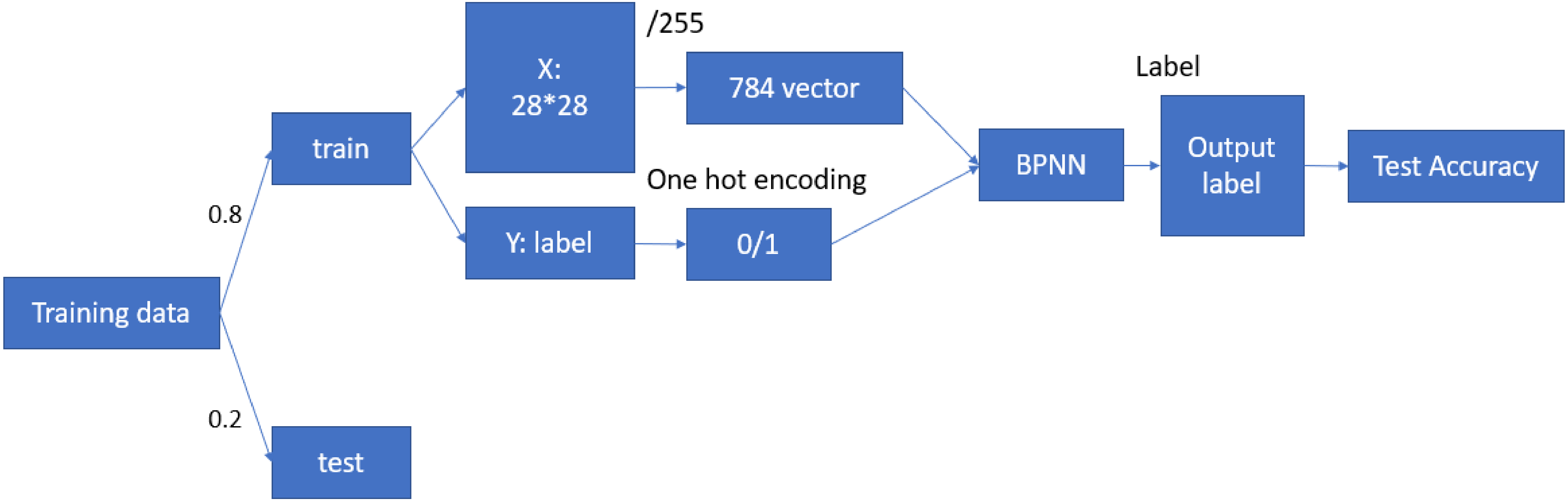
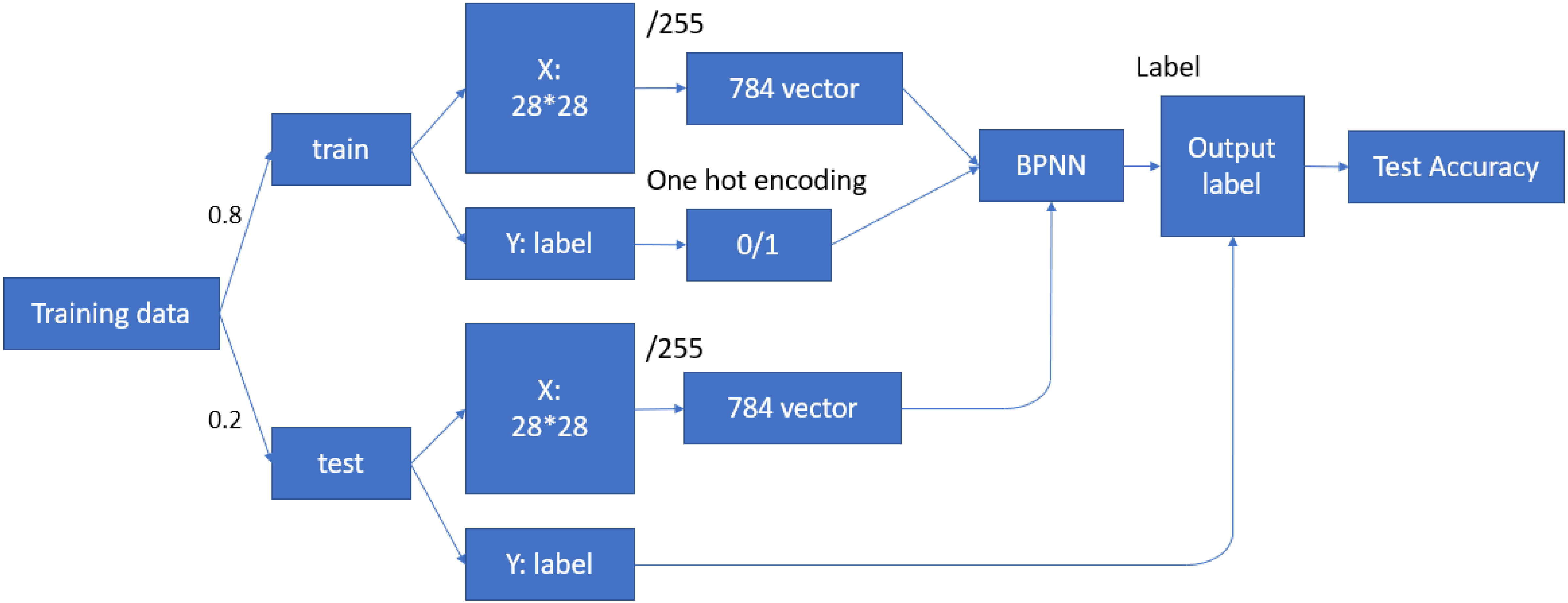
Implement of 2 layers backpropagation model

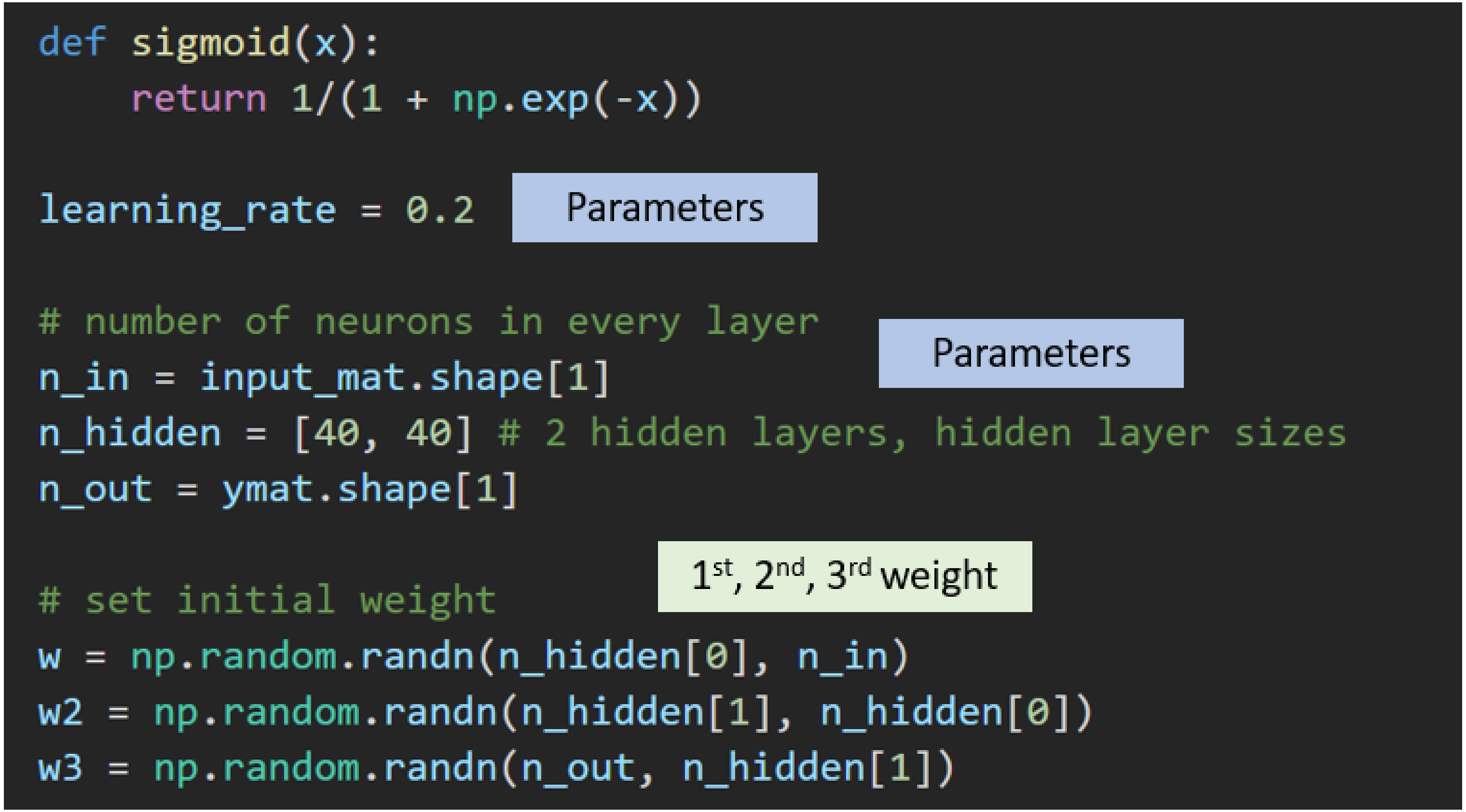
711033123 商巧昀

Learning Process



Learning Process





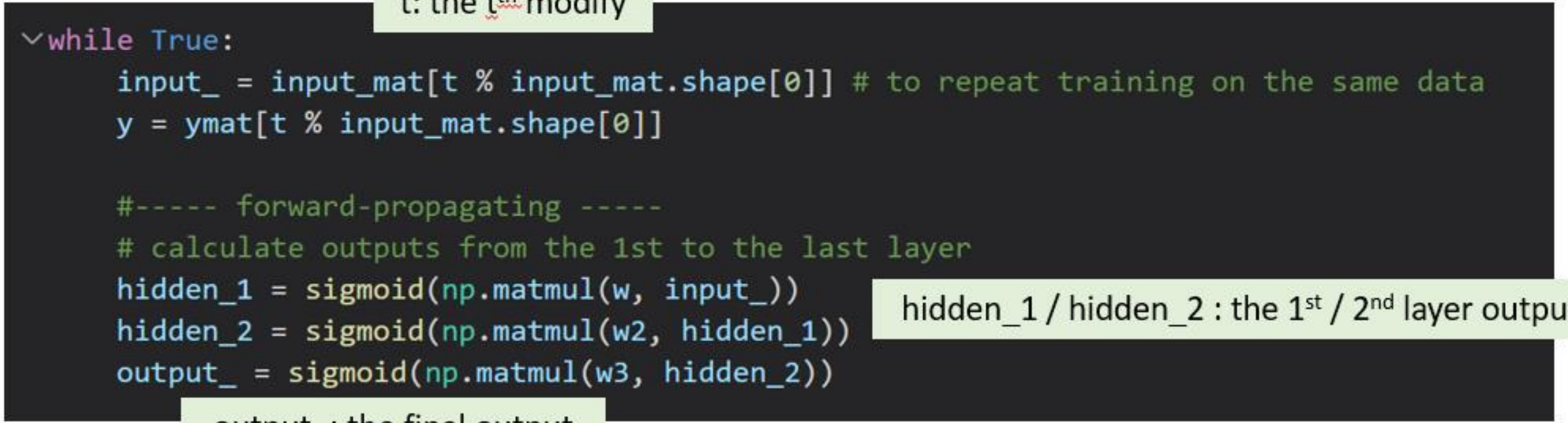
```
def sigmoid(x):
    return 1/(1 + np.exp(-x))

learning_rate = 0.2

# number of neurons in every layer
n_in = input_mat.shape[1]
n_hidden = [40, 40] # 2 hidden layers, hidden layer sizes
n_out = ymat.shape[1]

# set initial weight
w = np.random.randn(n_hidden[0], n_in)
w2 = np.random.randn(n_hidden[1], n_hidden[0])
w3 = np.random.randn(n_out, n_hidden[1])
```

Parameters

Parameters

1st, 2nd, 3rd weight

t: the t^{th} modify

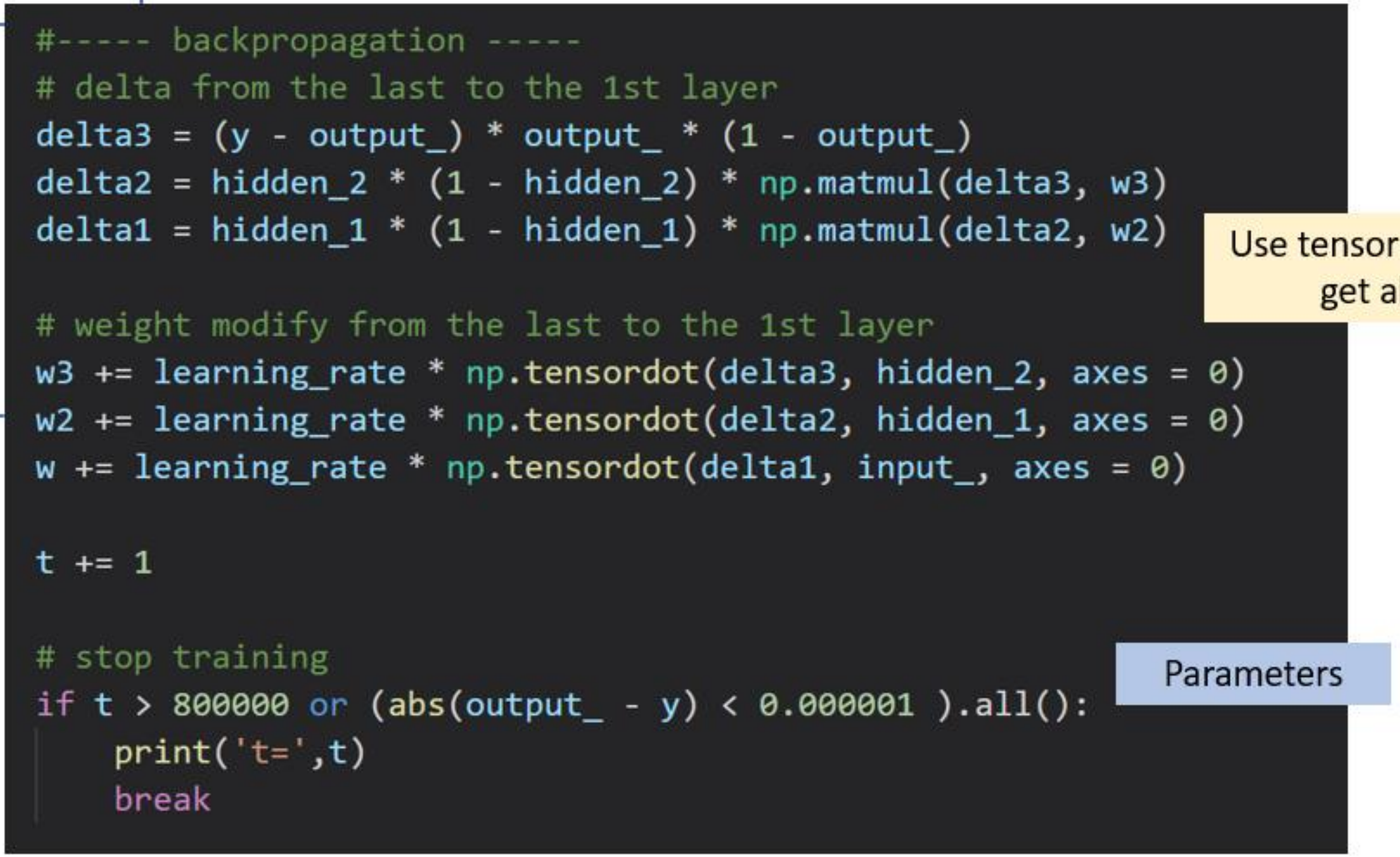


```
while True:
    input_ = input_mat[t % input_mat.shape[0]] # to repeat training on the same data
    y = ymat[t % input_mat.shape[0]]

    #----- forward-propagating -----
    # calculate outputs from the 1st to the last layer
    hidden_1 = sigmoid(np.matmul(w, input_))
    hidden_2 = sigmoid(np.matmul(w2, hidden_1))
    output_ = sigmoid(np.matmul(w3, hidden_2))
```

hidden_1 / hidden_2 : the 1st / 2nd layer output

output_ : the final output

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} = \eta \delta_j a_i$$



```
#----- backpropagation -----
# delta from the last to the 1st layer
delta3 = (y - output_) * output_ * (1 - output_)
delta2 = hidden_2 * (1 - hidden_2) * np.matmul(delta3, w3)
delta1 = hidden_1 * (1 - hidden_1) * np.matmul(delta2, w2)

# weight modify from the last to the 1st layer
w3 += learning_rate * np.tensordot(delta3, hidden_2, axes = 0)
w2 += learning_rate * np.tensordot(delta2, hidden_1, axes = 0)
w += learning_rate * np.tensordot(delta1, input_, axes = 0)

t += 1

# stop training
if t > 800000 or (abs(output_ - y) < 0.000001 ).all():
    print('t=',t)
    break
```

Use tensor product to get all Δw

Parameters