Ata Berk Çakır
21703127
Tutorial 2

```
    W1 = np.random.randn(n_h, n_x) * 0.01
    b1 = np.zeros((n_h, 1))
    W2 = np.random.randn(n_y, n_h) * 0.01
    b2 = np.zeros((n_y, 1))

        parameters["W" + str(l)] = np.random.randn(layer_dims[l], layer_dims[l-1]) * 0.01
        parameters["b" + str(l)] = np.zeros((layer_dims[l], 1))

    Z = np.dot(W,A)+b

        Z, activation_cache = linear_forward(A_prev, W, b)
        A, activation_cache = sigmoid(Z)

        Z, activation_cache = linear_forward(A_prev, W, b)
        A, activation_cache = relu(Z)

        A, linear_activation_cache = linear_activation_forward(A_prev, parameters['W' +
str(l)],parameters['b' + str(l)], activation = "relu")
        caches.append(linear_activation_cache)

    AL, linear_activation_cache = linear_activation_forward(A, parameters['W3'], parameters['b3'],
activation = "sigmoid")
    caches.append(linear_activation_cache)

    cost = - np.sum(np.multiply(np.log(AL),Y)+np.multiply(np.log(1-AL),(1-Y)))/m

    dW = np.dot(dZ,np.transpose(A_prev))/m
    db = np.sum(dZ,axis=1,keepdims = True)/m
    dA_prev = np.dot(np.transpose(W),dZ)

        dZ = relu_backward(dA, activation_cache)
```

```
        dZ = sigmoid_backward(dA, activation_cache)

    dA_prev, dW, db = linear_backward(dZ, linear_cache)

    dAL = - (np.divide(Y, AL) - np.divide(1 - Y, 1 - AL))

    dAL, dWL, dbL = linear_activation_backward(dAL, caches[L-1], activation = "sigmoid")
    grads = {"dA"+str(L-1): dAL,"dW"+str(L): dWL,"db"+str(L): dbL}

        dA = grads["dA" + str(l + 1)]
        cache = caches[l]
        activation = "relu"
        grads["dA" + str(l)],grads["dW" + str(l+1)],grads["db"+str(l+1)] = linear_activation_backward(dA, cache, activation)

    for l in range (1, L+1):
        parameters["W" + str(l)] -= learning_rate*grads["dW" + str(l)]
        parameters["b" + str(l)] -= learning_rate*grads["db" + str(l)]
```