因為對於 python 還是不熟悉，所以不知道 numpy.array 的用法，在撰寫 activation function、linear-activation backward 的時候花了非常多時間。最後是靠上網找 numpy.array 的用法跟 python 的教學影片才解決的。

Bonus 的部分在設定 parameter argument 跟 layers_dims 時花了非常多時間，一方面因為每次 train 都花很多時間，再加上有很多 arguments 需要調整。

初始化 parameters 後對 input 進行線性轉換。
再設計 SIGMOID RELU SOFTMAX 等 activation function

```
for l in range(1, L):
        A_prev = A
        A, cache = linear_activation_forward(A_prev, parameters['W' + str(l)], parameters['b' + str(l)], "relu")
        caches.append(cache)

if classes == 2:
    # Implement LINEAR -> SIGMOID. Add "cache" to the "caches" list.
    AL, cache = linear_activation_forward(A, parameters['W' + str(L)], parameters['b' + str(L)], "sigmoid")
    caches.append(cache)
    assert(AL.shape == (1,X.shape[1]))
```

利用 RELU 函數對上一層進行複製，所以要進行 L-1 次，最後在利用 SIGMOID(SOFTMAX for multi-class)函數計算出每個預測值的分數。

設計 binary cross-entropy loss function 來計算 binary classifier 的 cost。

4.1. Binary cross-entropy loss

Exercise: Compute the binary cross-entropy cost J, using the following formula: (5%)

$$-\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right)+(1-y^{(i)})\log\left(1-a^{[L](i)}\right)\right)$$

設計 categorical cross-entropy loss function 來計算 multi-class classifier 的 cost。

Exercise: Compute the categorical cross-entropy cost J, using the following formula: (5%)

$$-\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right)\right)$$

backward function 計算 loss function 的梯度。
之後根據梯度的下降來更新 parameter。

最後 predict

```
def update_parameters(parameters, grads, learning_rate):
    """
    Update parameters using gradient descent

    Arguments:
    parameters -- python dictionary containing your parameters
    grads -- python dictionary containing your gradients, output of L_model_backward

    Returns:
    parameters -- python dictionary containing your updated parameters
                  parameters["W" + str(l)] = ...
                  parameters["b" + str(l)] = ...
    """

    L = len(parameters) // 2 # number of layers in the neural network

    # Update rule for each parameter. Use a for loop.
    ### START CODE HERE ### (≈ 3 lines of code)
    for l in range(L):
        parameters["W" + str(l+1)] = parameters["W" + str(l+1)] - learning_rate * grads["dW" + str(l + 1)]
        parameters["b" + str(l+1)] = parameters["b" + str(l+1)] - learning_rate * grads["db" + str(l + 1)]
    ### END CODE HERE ###
    return parameters
```