Building your Deep Neural Network - Step by Step v8

July 20, 2019

1 Building your Deep Neural Network: Step by Step

Welcome to your week 4 assignment (part 1 of 2)! You have previously trained a 2-layer Neural Network (with a single hidden layer). This week, you will build a deep neural network, with as many layers as you want!

- In this notebook, you will implement all the functions required to build a deep neural network.
- In the next assignment, you will use these functions to build a deep neural network for image classification.

After this assignment you will be able to: - Use non-linear units like ReLU to improve your model - Build a deeper neural network (with more than 1 hidden layer) - Implement an easy-to-use neural network class

Notation: - Superscript $[l]$ denotes a quantity associated with the l^{th} layer. - Example: $a^{[L]}$ is the L^{th} layer activation. $W^{[L]}$ and $b^{[L]}$ are the L^{th} layer parameters. - Superscript (i) denotes a quantity associated with the i^{th} example. - Example: $x^{(i)}$ is the i^{th} training example. - Lowerscript i denotes the i^{th} entry of a vector. - Example: $a_i^{[l]}$ denotes the i^{th} entry of the l^{th} layer's activations).

Let's get started!

1.1 1 - Packages

Let's first import all the packages that you will need during this assignment. - numpy is the main package for scientific computing with Python. - matplotlib is a library to plot graphs in Python. - dnn_utils provides some necessary functions for this notebook. - testCases provides some test cases to assess the correctness of your functions - np.random.seed(1) is used to keep all the random function calls consistent. It will help us grade your work. Please don't change the seed.

```
In [1]: import numpy as np
        import h5py
        import matplotlib.pyplot as plt
        from testCases_v4 import *
        from dnn_utils_v2 import sigmoid, sigmoid_backward, relu, relu_backward

        %matplotlib inline
        plt.rcParams['figure.figsize'] = (5.0, 4.0) # set default size of plots
        plt.rcParams['image.interpolation'] = 'nearest'
```

```
plt.rcParams['image.cmap'] = 'gray'

%load_ext autoreload
%autoreload 2

np.random.seed(1)
```

1.2 2 - Outline of the Assignment

To build your neural network, you will be implementing several "helper functions". These helper functions will be used in the next assignment to build a two-layer neural network and an L-layer neural network. Each small helper function you will implement will have detailed instructions that will walk you through the necessary steps. Here is an outline of this assignment, you will:

- Initialize the parameters for a two-layer network and for an L-layer neural network.
- Implement the forward propagation module (shown in purple in the figure below).
  - Complete the LINEAR part of a layer's forward propagation step (resulting in $Z^{[l]}$).
  - We give you the ACTIVATION function (relu/sigmoid).
  - Combine the previous two steps into a new [LINEAR->ACTIVATION] forward function.
  - Stack the [LINEAR->RELU] forward function L-1 time (for layers 1 through L-1) and add a [LINEAR->SIGMOID] at the end (for the final layer L). This gives you a new L_model_forward function.
- Compute the loss.
- Implement the backward propagation module (denoted in red in the figure below).
  - Complete the LINEAR part of a layer's backward propagation step.
  - We give you the gradient of the ACTIVATE function (relu_backward/sigmoid_backward)
  - Combine the previous two steps into a new [LINEAR->ACTIVATION] backward function.
  - Stack [LINEAR->RELU] backward L-1 times and add [LINEAR->SIGMOID] backward in a new L_model_backward function
- Finally update the parameters.

Figure 1

Note that for every forward function, there is a corresponding backward function. That is why at every step of your forward module you will be storing some values in a cache. The cached values are useful for computing gradients. In the backpropagation module you will then use the cache to calculate the gradients. This assignment will show you exactly how to carry out each of these steps.

1.3 3 - Initialization

You will write two helper functions that will initialize the parameters for your model. The first function will be used to initialize parameters for a two layer model. The second one will generalize this initialization process to L layers.

1.3.1 3.1 - 2-layer Neural Network

Exercise: Create and initialize the parameters of the 2-layer neural network.

Instructions: - The model's structure is: *LINEAR -> RELU -> LINEAR -> SIGMOID*. - Use random initialization for the weight matrices. Use `np.random.randn(shape)*0.01` with the correct shape. - Use zero initialization for the biases. Use `np.zeros(shape)`.

In [4]:
```
# GRADED FUNCTION: initialize_parameters

def initialize_parameters(n_x, n_h, n_y):
    """
    Argument:
    n_x -- size of the input layer
    n_h -- size of the hidden layer
    n_y -- size of the output layer

    Returns:
    parameters -- python dictionary containing your parameters:
                    W1 -- weight matrix of shape (n_h, n_x)
                    b1 -- bias vector of shape (n_h, 1)
                    W2 -- weight matrix of shape (n_y, n_h)
                    b2 -- bias vector of shape (n_y, 1)
    """

    np.random.seed(1)

    ### START CODE HERE ### ( 4 lines of code)
    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))
    ### END CODE HERE ###

    assert(W1.shape == (n_h, n_x))
    assert(b1.shape == (n_h, 1))
    assert(W2.shape == (n_y, n_h))
    assert(b2.shape == (n_y, 1))

    parameters = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2}

    return parameters
```

In [5]:
```
parameters = initialize_parameters(3,2,1)
print("W1 = " + str(parameters["W1"]))
print("b1 = " + str(parameters["b1"]))
```

```
        print("W2 = " + str(parameters["W2"]))
        print("b2 = " + str(parameters["b2"]))
```

```
W1 = [[ 0.01624345 -0.00611756 -0.00528172]
 [-0.01072969  0.00865408 -0.02301539]]
b1 = [[ 0.]
 [ 0.]]
W2 = [[ 0.01744812 -0.00761207]]
b2 = [[ 0.]]
```

Expected output:
W1
[[0.01624345 -0.00611756 -0.00528172] [-0.01072969 0.00865408 -0.02301539]]
b1
[[0.] [0.]]
W2
[[0.01744812 -0.00761207]]
b2
[[0.]]

1.3.2 3.2 - L-layer Neural Network

The initialization for a deeper L-layer neural network is more complicated because there are many more weight matrices and bias vectors. When completing the `initialize_parameters_deep`, you should make sure that your dimensions match between each layer. Recall that $n^{[l]}$ is the number of units in layer l. Thus for example if the size of our input X is $(12288, 209)$ (with $m = 209$ examples) then:

```
<tr>
    <td>  </td>
    <td> **Shape of W** </td>
    <td> **Shape of b**  </td>
    <td> **Activation** </td>
    <td> **Shape of Activation** </td>
<tr>

<tr>
    <td> **Layer 1** </td>
    <td> $(n^{[1]},12288)$ </td>
    <td> $(n^{[1]},1)$ </td>
    <td> $Z^{[1]} = W^{[1]}  X + b^{[1]} $ </td>

    <td> $(n^{[1]},209)$ </td>
<tr>

<tr>
    <td> **Layer 2** </td>
```