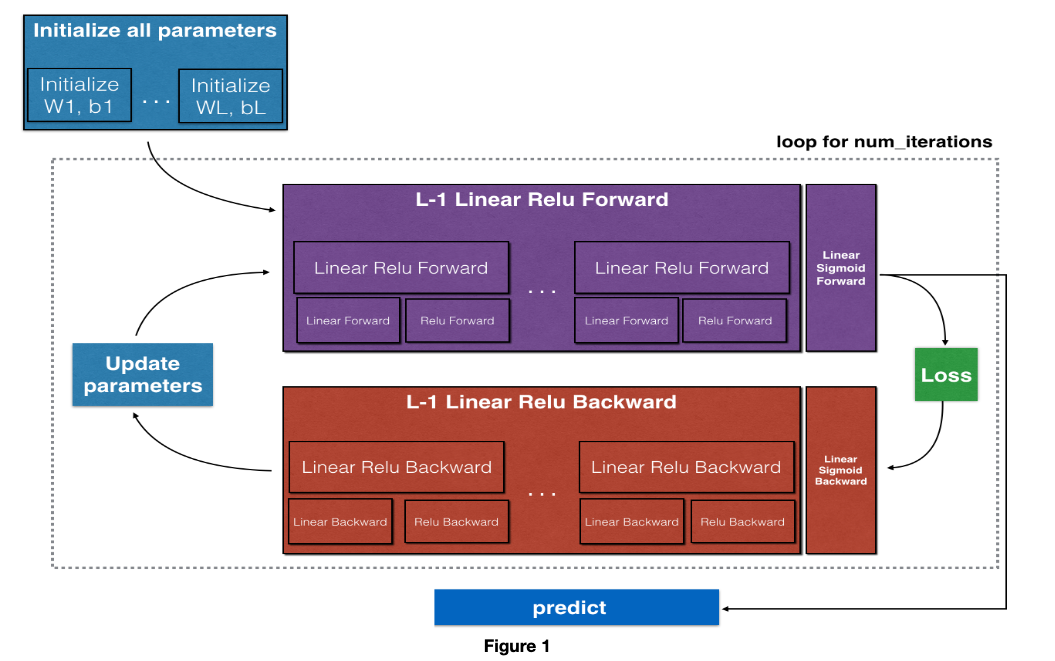
**DLS Course-1 : Deep Learning & Neural Network**

**Outline**

To build your neural network, you'll 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 will have detailed instructions to walk you through the necessary steps. Here's an outline of the steps in this assignment:

* Initialize the parameters for a two-layer network and for an 𝐿L-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]Z[l]).
  + The ACTIVATION function is provided for you (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 𝐿L). 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
  + The gradient of the ACTIVATE function is provided for you(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



### 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 function, you should make sure that your dimensions match between each layer. Recall that 𝑛[𝑙]n[l] is the number of units in layer 𝑙l. For example, if the size of your input 𝑋X is (12288,209)(12288,209) (with 𝑚=209m=209 examples) then:

Table

Description automatically generated

A picture containing calendar

Description automatically generated

### 4.3 - L-Layer Model

For even more convenience when implementing the 𝐿L-layer Neural Net, you will need a function that replicates the previous one (linear\_activation\_forward with RELU) 𝐿−1L−1 times, then follows that with one linear\_activation\_forward with SIGMOID.Diagram

Description automatically generated

### L\_model\_forward

Implement the forward propagation of the above model.

**Instructions**: In the code below, the variable AL will denote 𝐴[𝐿]=𝜎(𝑍[𝐿])=𝜎(𝑊[𝐿]𝐴[𝐿−1]+𝑏[𝐿])A[L]=σ(Z[L])=σ(W[L]A[L−1]+b[L]). (This is sometimes also called Yhat, i.e., this is 𝑌̂ Y^.)

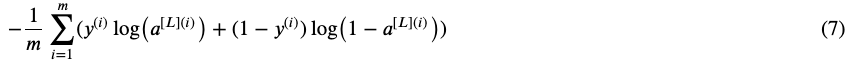
**Hints**:

* Use the functions you've previously written
* Use a for loop to replicate [LINEAR->RELU] (L-1) times
* Don't forget to keep track of the caches in the "caches" list. To add a new value c to a list, you can use list.append(c).

## 5 - Cost Function

Now you can implement forward and backward propagation! You need to compute the cost, in order to check whether your model is actually learning.

### Exercise 6 - compute\_cost

Compute the cross-entropy cost 𝐽J, using the following formula:

## 6 - Backward Propagation Module

Just as you did for the forward propagation, you'll implement helper functions for backpropagation. Remember that backpropagation is used to calculate the gradient of the loss function with respect to the parameters.

**Reminder**:

A picture containing text, clock, sale

Description automatically generated

Now, similarly, to forward propagation, you're going to build the backward propagation in three steps:

1. LINEAR backward
2. LINEAR -> ACTIVATION backward where ACTIVATION computes the derivative of either the ReLU or sigmoid activation
3. [LINEAR -> RELU] ×× (L-1) -> LINEAR -> SIGMOID backward (whole model)

For the next exercise, you will need to remember that:

* b is a matrix(np.ndarray) with 1 column and n rows, i.e: b = [[1.0], [2.0]] (remember that b is a constant)
* np.sum performs a sum over the elements of a ndarray
* axis=1 or axis=0 specify if the sum is carried out by rows or by columns respectively
* keepdims specifies if the original dimensions of the matrix must be kept.
* Look at the following example to clarify:

A picture containing table

Description automatically generated

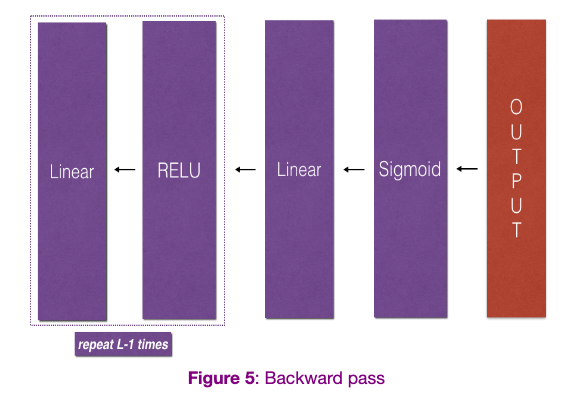
Graphical user interface, text, application, letter

Description automatically generated

### 6.3 - L-Model Backward

Now you will implement the backward function for the whole network!

Recall that when you implemented the L\_model\_forward function, at each iteration, you stored a cache which contains (X,W,b, and z). In the back propagation module, you'll use those variables to compute the gradients. Therefore, in the L\_model\_backward function, you'll iterate through all the hidden layers backward, starting from layer 𝐿. On each step, you will use the cached values for layer 𝑙 to backpropagate through layer 𝑙. Figure 5 below shows the backward pass.



**Initializing backpropagation**:

To backpropagate through this network, you know that the output is:

𝐴[𝐿]=𝜎(𝑍[𝐿])A[L]=σ(Z[L]).

Your code thus needs to compute dAL =∂L/∂A[L].

To do so, use this formula (derived using calculus which, again, you don't need in-depth knowledge of!):

dAL **=** **-** (np.divide(Y, AL) **-** np.divide(1 **-** Y, 1 **-** AL)) *# derivative of cost with respect to AL*

You can then use this post-activation gradient dAL to keep going backward. As seen in Figure 5, you can now feed in dAL into the LINEAR->SIGMOID backward function you implemented (which will use the cached values stored by the L\_model\_forward function).

After that, you will have to use a for loop to iterate through all the other layers using the LINEAR->RELU backward function. You should store each dA, dW, and db in the grads dictionary. To do so, use this formula :



For example, for 𝑙=3l=3 this would store 𝑑𝑊[𝑙] in grads["dW3"].

### Exercise 9 - L\_model\_backward

Implement backpropagation for the \*[LINEAR->RELU] ×× (L-1) -> LINEAR -> SIGMOID\* model.

**6.4 - Update Parameters**

Graphical user interface, text, application, letter, email

Description automatically generated