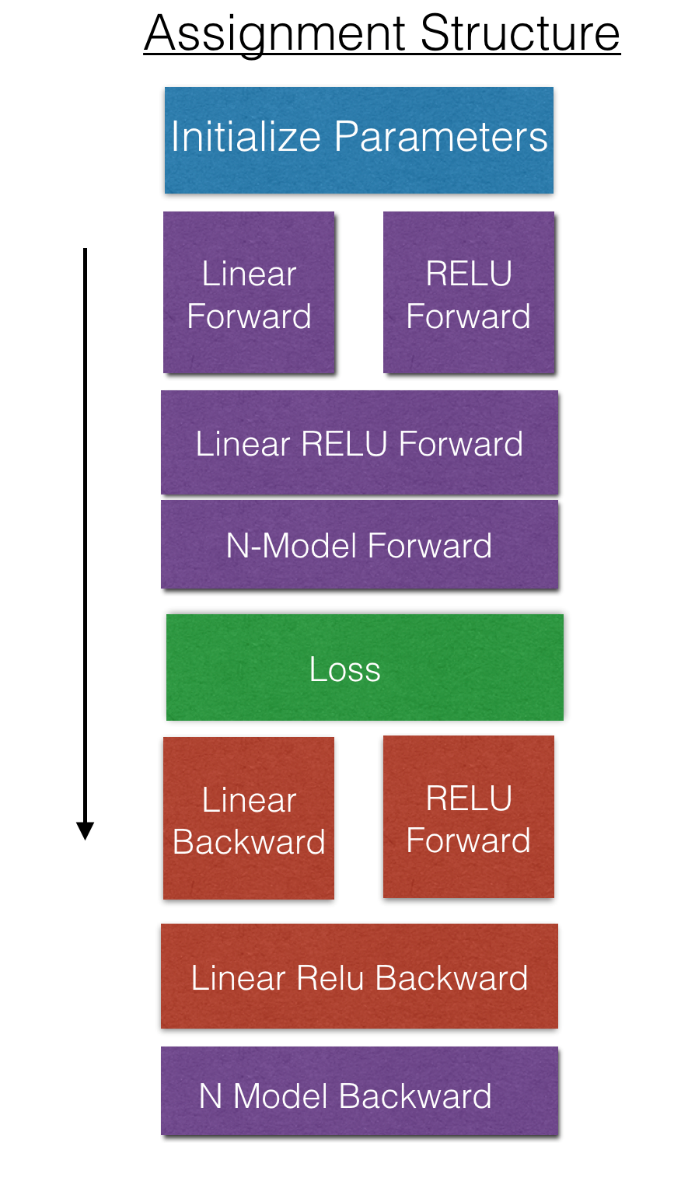
|  |  |
| --- | --- |
| **Ex No: 4**  **Date: 21st August 2024** | **Build and use a deep neural network to classify cat or non-cat** |

**Objective:** To implement the fundamental functions required to build a deep neural network with multiple layers using nonlinear activation units, such as ReLU.

**Description:**

In this exercise, we focus on constructing the essential components of a deep neural network and assembling them into a functional model. The process includes parameter initialization, forward and backward propagation, cost computation, and parameter updates through gradient descent. The goal is to develop deep neural networks capable of handling complex tasks, including image classification.



**Model:**

The architecture includes multiple layers, with possible dense or convolutional layers depending on the data type. The model's purpose is to classify or regress on the input data, and metrics such as accuracy or loss might be used to evaluate performance.

.

**Building the parts of the algorithm:**

1. **Initialization**

* 2-layer Neural Network: Begin by initializing parameters for a simple 2-layer neural network, which involves setting up weights and biases for the layers.
  + initialize\_parameters(): Initializes parameters for a 2-layer neural network.
* L-layer Neural Network: Extend the initialization to a deeper network by iterating over multiple layers.
  + initialize\_parameters\_deep(): Initializes parameters for a deep neural network with L layers.

1. **Forward Propagation Module**

* Linear Forward: Performs the linear portion of forward propagation for a single layer.
  + linear\_forward(): Calculates the linear transformation for a specific layer.
* Linear-Activation Forward: Combines the linear transformation with an activation function (ReLU or Sigmoid) for a layer.
  + linear\_activation\_forward(): Executes the forward pass with an activation function.
* L-Layer Model: Implements forward propagation across the entire network, processing each layer sequentially.
  + L\_model\_forward(): Conducts the forward pass through all layers of the deep network.

1. **Cost Function**

* Calculates the network's loss after forward propagation.
  + compute\_cost(): Computes the cross-entropy loss for the final output.

1. **Propagation Module**

* Linear Backward: Calculates the gradient of the loss with respect to the linear transformation in a single layer.
  + linear\_backward(): Computes gradients for the linear transformation of a layer.
* Linear-Activation Backward: Combines the linear backward step with the activation function's backward pass.
  + linear\_activation\_backward(): Performs the backward pass with an activation function.
* L-Model Backward: Executes backward propagation across the entire network.
  + L\_model\_backward(): Computes gradients for all layers of the network.

**Key Observations:**

The notebook's structure suggests a comprehensive approach to building a model pipeline. Key observations from the training process (e.g., convergence rate, validation performance, and generalization) are likely discussed towards the end. Insights could include the model's strengths, potential overfitting or underfitting, and areas for further improvement or tuning. We tested our model on an image classification task to predict if an image contains a cat or not using the L-layer model.

* **Train Accuracy**: 0.9809
* **Test Accuracy**: 0.8200

**GitHub Link:** <https://github.com/adrocx/DeepLearning>