|  |  |
| --- | --- |
| **Ex No: 3**  **Date:20/08/2024** | **Building a Deep Neural Network** |

**Objective:**

To build and train a deep neural network for image classification tasks using Python and the TensorFlow/Keras framework, exploring the implementation of forward and backward propagation, parameter initialization, and model optimization.

**Descriptions:**

**Introduction to Deep Learning and Image Classification:**

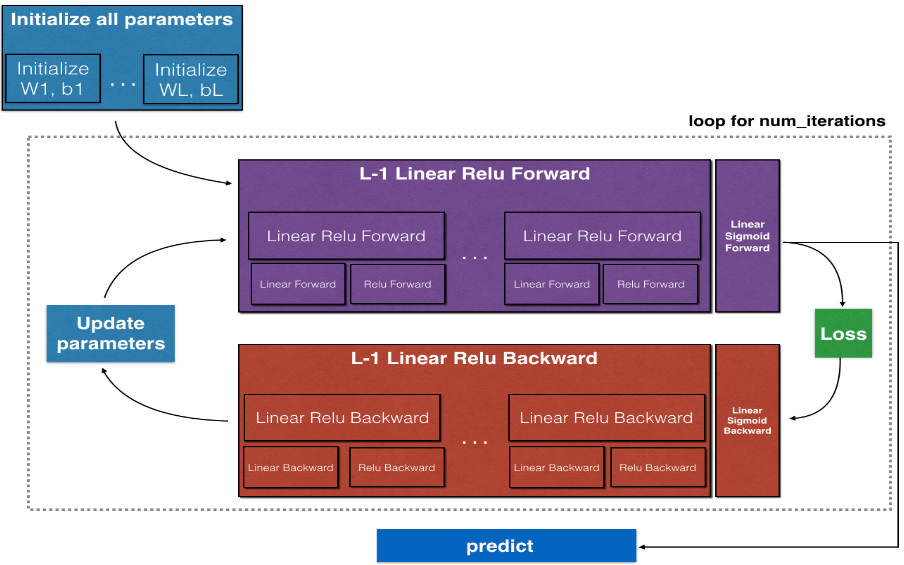
Deep learning involves training neural networks with multiple hidden layers to capture complex patterns in data. Image classification is a common application where a model is trained to recognize objects, animals, or scenes within images. This experiment demonstrates building a deep neural network using multiple layers to classify images into predefined categories.

**Neural Network Architecture:**

This experiment involves constructing a deep neural network with the following components:

* **Input Layer:** Receives the input features (e.g., pixel values of an image).
* **Hidden Layers:** Multiple layers using the ReLU activation function to capture intricate patterns.
* **Output Layer:** Uses the Sigmoid activation function to output probabilities, suited for binary classification.

The architecture's depth, controlled by the number of hidden layers, is crucial for learning from complex datasets.



**Model Implementation:**

**1. Data Handling:**

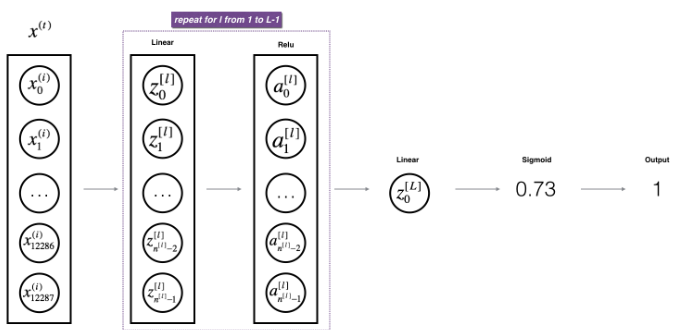
* Data is pre-processed, including normalization and splitting into training and test sets.
* Data augmentation may be applied to improve generalization.

**2. Initialization:**

* Parameters (weights and biases) are initialized using methods like Xavier/He initialization to facilitate better convergence during training.

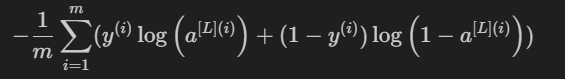
**3. Forward Propagation:**

* **Linear Step:** Each layer computes a linear combination of inputs and weights.
* **Activation Step:** ReLU is applied to hidden layers, and Sigmoid is applied to the output layer.



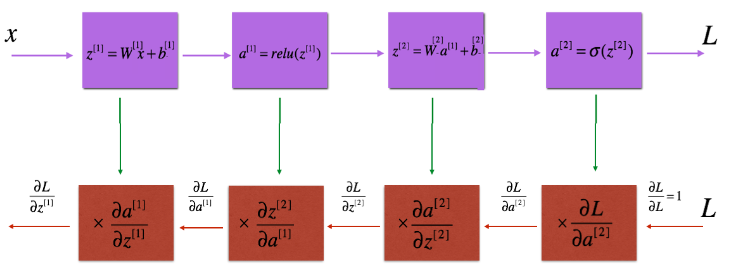
**4. Cost Function:**

* The cross-entropy loss function is utilized to quantify the error between predicted outputs and actual labels.



**5. Backward Propagation:**

* Gradients are computed with respect to each parameter using the chain rule, backpropagating from the output to the input layer.



**6. Parameter Update:**

* Parameters are updated using the computed gradients, typically employing optimization algorithms like gradient descent.

**Results and Discussion:**

The deep neural network is trained on a given dataset, and its performance is evaluated using accuracy and loss metrics on the test data. The model’s ability to classify images accurately is analyzed, highlighting the importance of depth (i.e., number of layers) in capturing complex patterns.

By utilizing both ReLU and Sigmoid activations, the network effectively learns non-linear decision boundaries, which are critical for high performance in image classification tasks.

**GitHub Link:** [**https://github.com/FMS07/Fundamentals\_of\_DL/blob/main/Lab03-%20Build%20A%20Deep%20Neural%20Network/Building\_Deep\_Neural\_Network\_Distri.ipynb**](https://github.com/FMS07/Fundamentals_of_DL/blob/main/Lab03-%20Build%20A%20Deep%20Neural%20Network/Building_Deep_Neural_Network_Distri.ipynb)