|  |  |
| --- | --- |
| **Ex No: 2**  **Date: 20/08/2024** | **Building your Deep Neural Network: Step by Step** |

**Objective:**

To implement and evaluate a deep neural network with multiple layers for a 2-class classification problem, extending from a previous exercise involving a single hidden layer. The exercise involves building the architecture, initializing parameters, performing forward and backward propagation, computing the cost, and updating parameters using gradient descent. The goal is to achieve improved classification accuracy compared to simpler models, such as logistic regression.

**Descriptions:**

In this lab, we advanced from a 2-layer neural network to building a deep neural network with multiple layers. This extension aimed to improve performance on a non-linearly separable dataset, represented as a "flower" pattern with red and blue points. The task involved several steps:

1. Initialization:

- 2-Layer Neural Network: Parameters were initialized for a neural network with the structure LINEAR -> RELU -> LINEAR -> SIGMOID. Weight matrices were initialized with small random values and biases with zeros.

- L-Layer Neural Network: Parameters were initialized for a deeper network with multiple layers using similar methods but generalized for multiple layers. This involved creating weight matrices and bias vectors for each layer based on the specified layer dimensions.

2. Forward Propagation:

- Implemented the linear forward step to compute \( Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]} \).

- Applied activation functions such as ReLU and sigmoid. Forward propagation was then combined to handle multiple layers, leading to a final output layer using sigmoid activation.

3. Cost Computation:

- The cost function used was cross-entropy loss, calculated as \( -\frac{1}{m} \sum\_{i=1}^{m} \left[y^{(i)} \log(a^{[L](i)}) + (1 - y^{(i)}) \log(1 - a^{[L](i)})\right] \). This quantifies the difference between predicted and actual values.

4. Backward Propagation:

- Implemented backward propagation to compute gradients of the cost function with respect to each parameter. This involved calculating the gradients for each layer and propagating errors backward through the network.

- Used the chain rule to combine gradients from the activation and linear parts of each layer.

5. Parameter Updates:

- Applied gradient descent to update weights and biases using the computed gradients. This step aimed to minimize the cost function and improve the model’s performance.

6. Model Evaluation:

- After training the deep neural network, evaluated its performance on the dataset and compared the results to the baseline logistic regression model. The deep network aimed to achieve better classification accuracy by leveraging its ability to capture complex patterns.

Building the Parts of the Algorithm:

1. Define Neural Network Structure:

- Specify the number of input units, hidden units, and output units based on the dataset and design choice.

2. Initialize Parameters:

- Initialize weight matrices and bias vectors with small random values and zeros, respectively.

3. Forward Propagation:

- Compute activations for each layer using the appropriate activation functions. Store intermediate values for backpropagation.

4. Compute Cost:

- Calculate the cross-entropy cost to measure prediction error.

5. Backward Propagation:

- Compute gradients of the cost function with respect to parameters and propagate errors backward through the network.

6. Update Parameters:

- Use gradient descent to update parameters and minimize the cost function.

7. Evaluate Model:

- Assess the model’s performance on the dataset and compare it with the logistic regression baseline to evaluate improvements in classification accuracy.

By following these steps, the assignment successfully demonstrated the process of building and evaluating a deep neural network, improving upon previous models and handling more complex datasets.

**GitHub Link:** https://github.com/dyuthiramesh/Deep\_Learning\_Elective/blob/main/Sem5/Lab3\_1/ Building\_Deep\_Neural\_Network\_Distri.ipynb