|  |  |
| --- | --- |
| **Ex No: 1**  **Date: 08-08-2024** | **Implementing a Multi-Class Classification Model using gradient decent** |

**Objective:**

* **Develop a Multi-Class Classification Model**: Implement a SoftMax regression model to classify images into multiple flower categories.
* **Generalize Logistic Regression**: Extend logistic regression to handle multiple classes using the SoftMax function.
* **Optimize Parameters**: Utilize gradient descent to optimize model parameters for better classification performance.
* **Evaluate Performance**: Assess the model’s accuracy on a test dataset to gauge its effectiveness.
* **Enhance Classification Accuracy**: Apply advanced techniques and fine-tune hyperparameters to improve model accuracy and robustness.

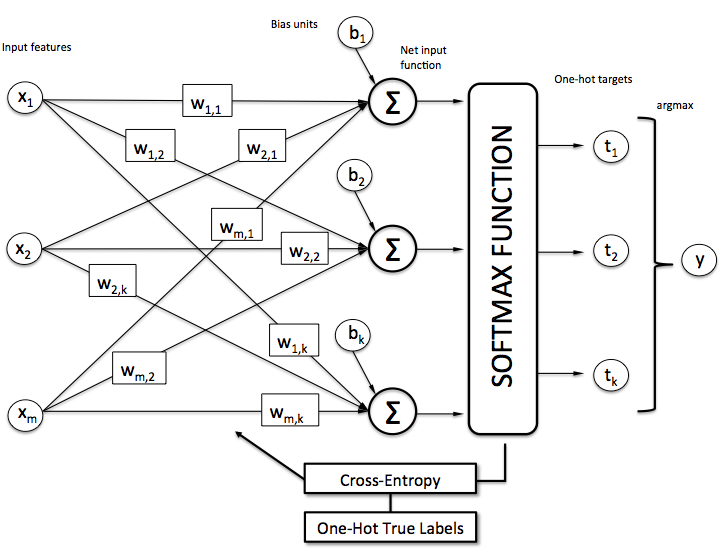
**Descriptions:**

Multi-class classification extends the concept of binary classification to handle more than two classes. In a multi-class classification problem, the goal is to classify input elements into one of several predefined categories. For instance, given an input image, the task is to classify it into one of multiple classes, such as different types of animals or flowers.

SoftMax regression, also known as multinomial logistic regression, is commonly used for multi-class classification problems. Unlike binary logistic regression, which outputs a probability for two classes, softmax regression outputs a probability distribution over multiple class. The algorithm assigns each input to the class with the highest probability. In softmax regression, the model does not have hidden layers; it directly maps the input features to class probabilities through a softmax function applied to the weighted sum of the input features. When initialized with zeros, the weights in a softmax regression model will initially produce uniform probabilities, but as the algorithm iterates through the training process, the weights are adjusted based on the input data, optimizing the classification performance across all classes.

4o mini

**Model:**



* Model Initialization:
* Initialize weights and biases for the neural network layers.
* Forward Propagation:
* Compute predictions using the SoftMax function and forward pass through the network.
* Cost Function:
* Calculate the cross-entropy loss to measure the performance of the model.
* Backward Propagation:
* Compute gradients for weights and biases using backpropagation to update the model parameters

**Building the parts of algorithm**

* **Define the Model Structure**:
* **Input Layer**: Specify the number of input features, which corresponds to the flattened size of each image.
* **Hidden Layers**: Design the architecture by determining the number of hidden layers and neurons in each layer. Use activation functions such as ReLU for hidden layers.
* **Output Layer**: Set up the output layer with a number of neurons equal to the number of classes, using the softmax activation function to produce probabilities for each class.
* **Initialize the Model's Parameters**:
* **Weights**: Initialize weights for each layer, typically using random values or specific initialization techniques like Xavier or He initialization.
* **Biases**: Initialize biases for each layer, often starting with zeros or small values.
* **Loop Through Training**:
* **Calculate Current Loss (Forward Propagation)**: Pass the input data through the network to compute the predicted class probabilities using the softmax function, and calculate the loss using a cross-entropy loss function.
* **Calculate Current Gradient (Backward Propagation)**: Compute gradients of the loss function with respect to each parameter by backpropagation, which involves calculating how the loss changes with changes in weights and biases.
* **Update Parameters (Gradient Descent)**: Adjust the weights and biases using an optimization algorithm like gradient descent, which updates parameters to minimize the loss function. This step typically involves adjusting the learning rate to control the update magnitude.

**GitHub Link:**