Experiment No. 3

Apply Stochastic Gradient Descent algorithm on a feed forward neural network for Iris Flower classification

Date of Performance:

Date of Submission:

Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Aim: Apply Stochastic Gradient Descent algorithm on a feed forward neural network for Iris

Flower classification.

Objective: Ability to perform optimization technique on a feed forward neural network.

Theory:

Gradient Descent is an iterative optimization process that searches for an objective function's

optimum value (Minimum/Maximum). It is one of the most used methods for changing a

model's parameters in order to reduce a cost function in machine learning projects.

The primary goal of gradient descent is to identify the model parameters that provide the

maximum accuracy on both training and test datasets. In gradient descent, the gradient is a

vector pointing in the general direction of the function's steepest rise at a particular point. The

algorithm might gradually drop towards lower values of the function by moving in the

opposite direction of the gradient, until reaching the minimum of the function.

Types of Gradient Descent:

Typically, there are three types of Gradient Descent:

Batch Gradient Descent

Stochastic Gradient Descent

Mini-batch Gradient Descent

Stochastic Gradient Descent (SGD):

Stochastic Gradient Descent (SGD) is a variant of the Gradient Descent algorithm that is used

for optimizing machine learning models. It addresses the computational inefficiency of

traditional Gradient Descent methods when dealing with large datasets in machine learning

projects.

Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

In SGD, instead of using the entire dataset for each iteration, only a single random training

example (or a small batch) is selected to calculate the gradient and update the model

parameters. This random selection introduces randomness into the optimization process,

hence the term "stochastic" in stochastic Gradient Descent

The advantage of using SGD is its computational efficiency, especially when dealing with

large datasets. By using a single example or a small batch, the computational cost per

iteration is significantly reduced compared to traditional Gradient Descent methods that

require processing the entire dataset.

Stochastic Gradient Descent Algorithm

Initialization: Randomly initialize the parameters of the model.

Set Parameters: Determine the number of iterations and the learning rate (alpha) for updating

the parameters.

Stochastic Gradient Descent Loop: Repeat the following steps until the model converges or

reaches the maximum number of iterations:

a. Shuffle the training dataset to introduce randomness.

b. Iterate over each training example (or a small batch) in the shuffled order.

c. Compute the gradient of the cost function with respect to the model parameters using the

current training.

d. Update the model parameters by taking a step in the direction of the negative gradient,

scaled by the learning rate.

e. Evaluate the convergence criteria, such as the difference in the cost function between

iterations of the gradient.



Vidyavardhini's College of Engineering and Technology Department of Artificial Intelligence & Data Science

Return Optimized Parameters: Once the convergence criteria are met or the maximum number of iterations is reached, return the optimized model parameters.

In SGD, since only one sample from the dataset is chosen at random for each iteration, the path taken by the algorithm to reach the minima is usually noisier than your typical Gradient Descent algorithm. But that doesn't matter all that much because the path taken by the algorithm does not matter, as long as we reach the minimum and with a significantly shorter training time.

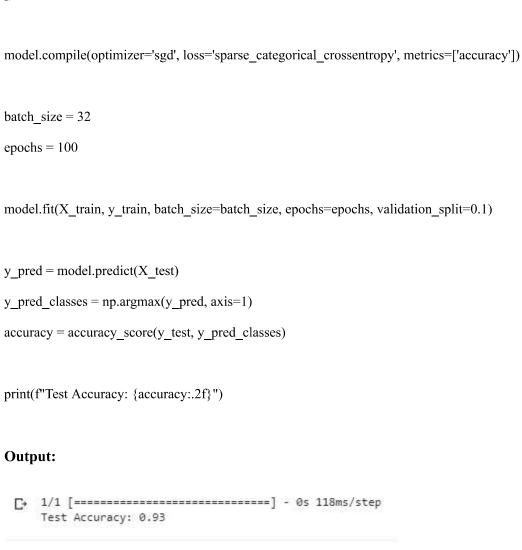
Code:

```
import numpy as np
import tensorflow as tf
from sklearn.datasets import load iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy score
iris = load iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X \text{ test} = \text{scaler.transform}(X \text{ test})
model = tf.keras.Sequential([
  tf.keras.layers.Input(shape=(4,)),
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.Dense(3, activation='softmax')
CSL701: Deep Learning Lab
```



Vidyavardhini's College of Engineering and Technology Department of Artificial Intelligence & Data Science

])



Conclusion: