Experiment No. 3
Apply Stochastic Gradient Descent algorithm on a feed forward neural network for Iris Flower classification
Date of Performance:
Date of Submission:

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.

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.

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.

Program:

```
import pandas as pd

dataset = pd.read_csv("IRIS.csv")

dataset.head()
```

	sepal-length	sepal-width	petal-length	petal-width	species
91	6.1	3.0	4.6	1.4	1
63	6.1	2.9	4.7	1.4	1
103	6.3	2.9	5.6	1.8	2
6	4.6	3.4	1.4	0.3	0
59	5.2	2.7	3.9	1.4	1

```
dataset['species'].value_counts()

1 50
2 50
0 50
Name: species, dtype: int64
```

```
dataset['species'] = pd.Categorical(dataset['species']).codes
dataset = dataset.sample(frac=1, random_state=1234)

train_input = dataset.values[:120, :4]
train_target = dataset.values[:120, 4]

test_input = dataset.values[120:, :4]
test_target = dataset.values[120:, 4]
```

```
import torch

torch.manual_seed(1234)

hidden_units = 5

net = torch.nn.Sequential(
 torch.nn.Linear(4, hidden_units),
 torch.nn.ReLU(),
 torch.nn.Linear(hidden_units, 3)
)
```

```
# choose optimizer and loss function
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(net.parameters(), lr=0.1, momentum=0.9)
```

```
# train
epochs = 10

for epoch in range(epochs):
 inputs = torch.autograd.Variable(torch.Tensor(train_input).float())
 targets = torch.autograd.Variable(torch.Tensor(train_target).long())

 optimizer.zero_grad()
 out = net(inputs)
 loss = criterion(out, targets)
 loss.backward()
 optimizer.step()

 if epoch == 0 or (epoch + 1) % 1 == 0:
   print('Epoch %d Loss: %.4f' % (epoch + 1, loss.item()))
```

```
Epoch 1 Loss: 0.5945
Epoch 2 Loss: 0.4891
Epoch 3 Loss: 0.4959
Epoch 4 Loss: 0.4165
Epoch 5 Loss: 0.4072
Epoch 6 Loss: 0.3776
Epoch 7 Loss: 0.3555
Epoch 8 Loss: 0.3055
Epoch 9 Loss: 0.3217
Epoch 10 Loss: 0.2447
```

```
import numpy as np

inputs = torch.autograd.Variable(torch.Tensor(test_input).float())
targets = torch.autograd.Variable(torch.Tensor(test_target).long())

optimizer.zero_grad()
out = net(inputs)
_, predicted = torch.max(out.data, 1)

error_count = test_target.size - np.count_nonzero((targets ==
predicted).numpy())
print('Errors: %d; Accuracy: %d%%' % (error_count, 100 * torch.sum(targets
== predicted) / test_target.size))
```

Output:

```
Errors: 5; Accuracy: 83%
```

Conclusion:

Through these practical I get to know about applying Stochastic Gradient Descent algorithm to a feedforward neural network for Iris Flower classification that demonstrates the effectiveness of neural networks for multi-class classification tasks by updating weights with small mini-batches of data , which reduces the computation time compared to using the entire dataset.