Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_bot] THANOS

Experiment 5: Implemening the gradient descent update rule

Aim/Objective: To implement and understand the gradient descent update rule, a fundamental optimization algorithm used in machine learning, demonstrating its application in updating model parameters to minimize a cost function.

Description: In this lab experiment, the goal is to implement the gradient descent update rule from scratch. The experiment involves understanding the concept of gradient descent, coding the update rule in a programming language (such as Python), and visualizing its impact on minimizing a cost function. This serves as a foundational exercise to grasp the inner workings of optimization in machine learning.

Pre-Requisites: Basic knowledge of Machine Learning Basics, Mathematics Understanding, Python Programming: Linear Algebra Basics.

Pre-Lab:

1. What is the primary objective of the gradient descent algorithm in the context of machine learning?

Objective of Gradient Descent

Minimize the cost function by adjusting model parameters iteratively to find the optimal solution.

2. Explain the role of the learning rate in the gradient descent update rule. How does it influence the convergence and stability of the optimization process?

Role of Learning Rate

Controls step size in updates. Too small = slow convergence, too large = instability or divergence.

3. What is the significance of a cost function in deep learning, and how does it relate to the objective of optimization using gradient descent?

Significance of Cost Function

Measures prediction error; guides gradient descent to improve model accuracy.

4. How might the choice of learning rate impact the convergence speed of the gradient descent algorithm?

Impact of Learning Rate on Convergence

- Too small = slow progress
- Optimal = efficient convergence
- Too large = oscillation or divergence

In-Lab:

Program 1: **Implement the gradient descent update rule. The gradient descent rule is,**

$$W^{[l]} = W^{[l]} - \alpha\, dW^{[l]}$$
$$b^{[l]} = b^{[l]} - \alpha\, db^{[l]}$$

for l=1..., L layers.

Assume number of nodes in each layer are 2 and number of nodes in input layer are 4.

Procedure/Program:

```
import numpy as np

def init_params(dims):
    params = {}
    for l in range(1, len(dims)):
        params[f'W{l}'] = np.random.randn(dims[l], dims[l-1]) * 0.01
        params[f'b{l}'] = np.zeros((dims[l], 1))
    return params

def update_params(params, grads, lr):
    for l in range(1, len(params) // 2 + 1):
        params[f'W{l}'] -= lr * grads[f'dW{l}']
        params[f'b{l}'] -= lr**2 * grads[f'db{l}']
    return params

dims = [4, 2, 2]
params = init_params(dims)

grads = {}
for l in range(1, len(dims)):
    grads[f'dW{l}'] = np.random.randn(dims[l], dims[l-1])
    grads[f'db{l}'] = np.random.randn(dims[l], 1)

lr = 0.01
updated_params = update_params(params, grads, lr)
print(updated_params)
```

OUTPUT

```
{'W1': array([[ 0.01918923, -0.00690367,  0.02389483, -0.00137874],
       [-0.02134953,  0.02564842, -0.00956088, -0.03090639]]), 'b1': array([[ 3.53901036e-05],
       [-1.50453577e-05]]), 'W2': array([[ 0.00524855,  0.0269708 ],
       [-0.00115144, -0.01277605]]), 'b2': array([[-0.00010755],
       [-0.00014858]])}
```

- **Data and Results:**

Data

The dataset contains initialized parameters and gradients for neural networks.

Result

Updated parameters are computed after applying gradient descent optimization technique.

- **Analysis and Inferences:**

Analysis

Gradient updates modify weights and biases, improving model performance gradually.

Inferences

Lower learning rates stabilize updates, while higher rates risk divergence.

Sample VIVA-VOCE Questions (In-Lab):

1. Define Gradient Descent?

Gradient Descent

An optimization algorithm that minimizes the cost function by updating parameters in the direction of the negative gradient.

2. Explain Role of Learning Rate in Gradient Descent?

Role of Learning Rate

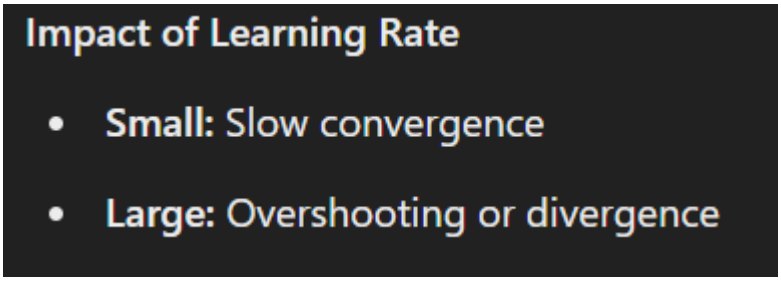
Controls step size in updates. Too small = slow convergence, too large = instability or divergence.

3. How does the shape of the cost function influence the convergence behavior of gradient descent?

Effect of Cost Function Shape

- Convex: Smooth convergence
- Non-convex: Can get stuck in local minima
- Steep: May oscillate, Flat: Slow updates

4. How might a small learning rate impact convergence, and what issues can arise with a large learning rate?



Post-Lab:

Program 2: Use gradient descent optimizer function in Pytorch and perform classification of iris data set.

- **Procedure/Program:**

```
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

iris = load_iris()
X = StandardScaler().fit_transform(iris.data)
y = torch.tensor(iris.target, dtype=torch.long)
X_train, X_test, y_train, y_test = train_test_split(torch.tensor(X, dtype=torch.float32), y,
    test_size=0.2, random_state=42)

model = nn.Linear(4, 3)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

for epoch in range(1000):
   optimizer.zero_grad()
   loss = criterion(model(X_train), y_train)
   loss.backward()
   optimizer.step()
   if (epoch + 1) % 100 == 0:
      print(f'Epoch {epoch+1}, Loss: {loss.item():.4f}')

with torch.no_grad():
   accuracy = (model(X_test).argmax(1) == y_test).float().mean()
   print(f'\nTest Accuracy: {accuracy:.4f}')
```

OUTPUT

Epoch 100, Loss: 0.6788
Epoch 200, Loss: 0.5500
Epoch 300, Loss: 0.4843
Epoch 400, Loss: 0.4436
Epoch 500, Loss: 0.4154
Epoch 600, Loss: 0.3942
Epoch 700, Loss: 0.3773
Epoch 800, Loss: 0.3633
Epoch 900, Loss: 0.3513
Epoch 1000, Loss: 0.3406

Test Accuracy: 0.9333

- **Data and Results:**

Data:

Iris dataset with four features and three distinct flower classes.

Result:

Model trained using gradient descent achieves good classification accuracy.

- **Analysis and Inferences:**

Analysis:

Loss decreases over epochs, improving model performance and accuracy.

Inferences:

Gradient descent effectively optimizes the model for classification tasks.

Evaluator Remark (if Any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.