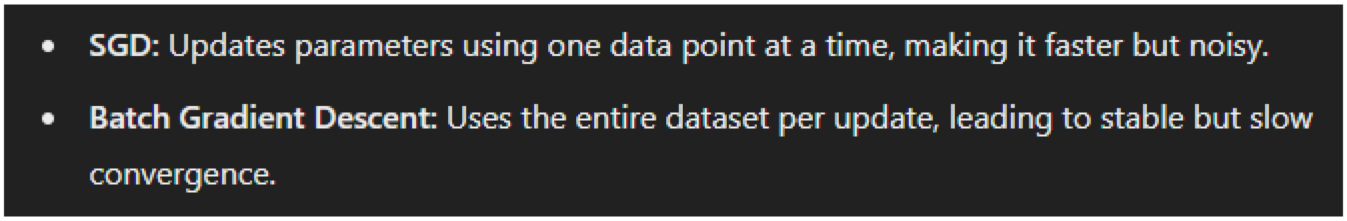
Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_bot] THANOS

Experiment 6: Implementing the Stochastic Gradient Descent update rule.

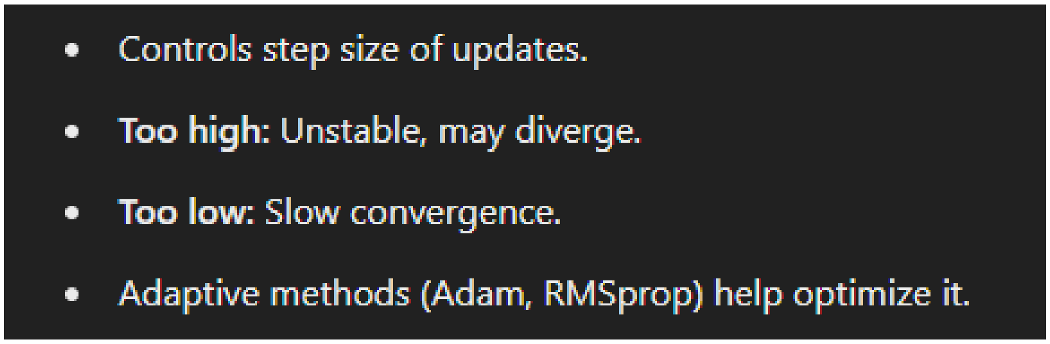
Aim/Objective: To implement and understand the Stochastic Gradient Descent (SGD) update rule, a variant of the gradient descent algorithm, demonstrating its application in optimizing model parameters for machine learning tasks.
Description: In this lab experiment, the objective is to implement the Stochastic Gradient Descent update rule from scratch. The experiment involves understanding the concept of SGD, coding the update rule in a programming language (such as Python), and comparing it with batch gradient descent. This exercise aims to showcase the advantages and differences of SGD, especially in scenarios involving large datasets.
Pre-Requisites: Basic knowledge of Machine Learning Basics, Gradient Descent Understanding, Mathematics Understanding, Python Programming: Linear Algebra Basics.
Pre-Lab:

1. What is the fundamental concept behind Stochastic Gradient Descent, and how does it differ from batch gradient descent?



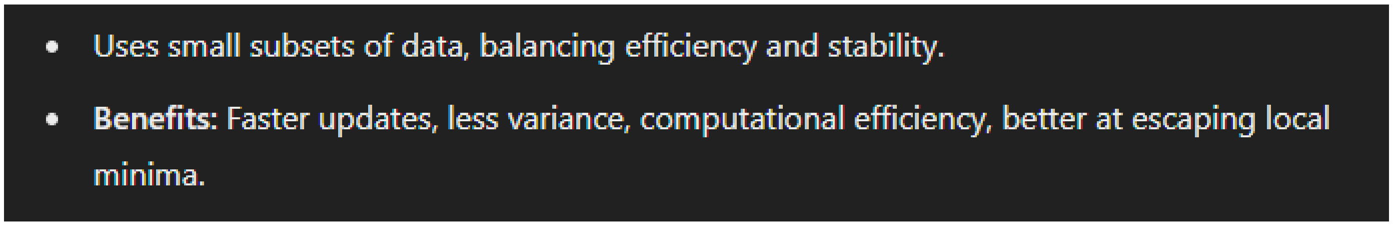
- **SGD:** Updates parameters using one data point at a time, making it faster but noisy.
- **Batch Gradient Descent:** Uses the entire dataset per update, leading to stable but slow convergence.

2. How is the learning rate utilized in the context of Stochastic Gradient Descent, and what considerations should be made in its selection?



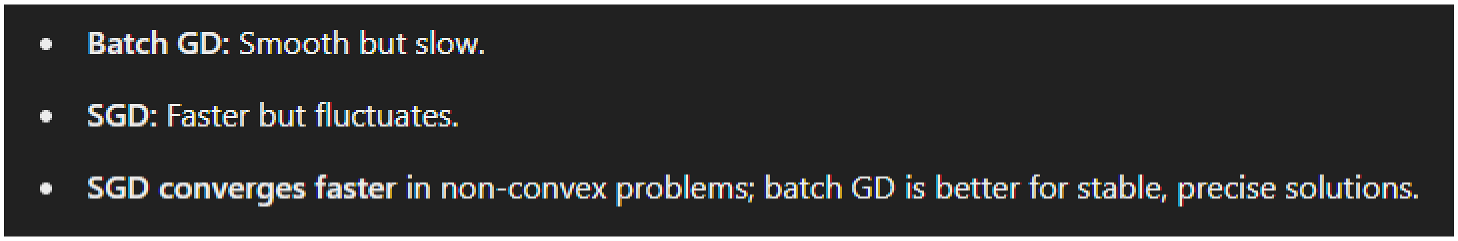
- Controls step size of updates.
- **Too high:** Unstable, may diverge.
- **Too low:** Slow convergence.
- Adaptive methods (Adam, RMSprop) help optimize it.

3. What is mini-batch gradient descent, and how does it combine aspects of both batch and stochastic gradient descent? What are the benefits of using mini batches?



- Uses small subsets of data, balancing efficiency and stability.
- **Benefits:** Faster updates, less variance, computational efficiency, better at escaping local minima.

4. Compare the convergence behavior of Stochastic Gradient Descent with batch gradient descent. In what scenarios might one converge faster than the other?



- **Batch GD:** Smooth but slow.
- **SGD:** Faster but fluctuates.
- **SGD converges faster** in non-convex problems; batch GD is better for stable, precise solutions.

In-Lab:

Program 1: Implement the Stochastic Gradient Descent update rule. The gradient descent rule is,

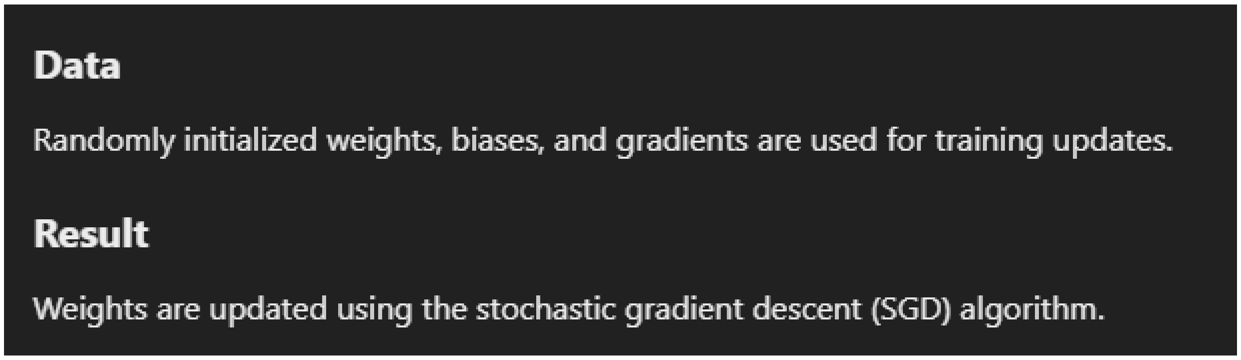
$$W^{[l]} = W^{[l]} - \alpha \, dW^{[l]}$$
$$b^{[l]} = b^{[l]} - \alpha \, db^{[l]}$$

for l=1..., L layers. Assume number of nodes in each layer are 2 and number of nodes in input layer are 4., use only numpy and matplotlib.

Procedure/Program:

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

L, input_size, nodes, α = 3, 4, 2, 0.01

W = [np.random.randn(nodes, input_size if i == 0 else nodes) for i in range(L)]
b = [np.random.randn(nodes, 1) for _ in range(L)]

dW = [np.random.randn(*w.shape) for w in W]
db = [np.random.randn(*b.shape) for b in b]

for l in range(L):
   W[l] -= α * dW[l]
   b[l] -= α * db[l]

plt.figure(figsize=(10, 5))

for i, w in enumerate(W):
   plt.subplot(1, L, i + 1)
   plt.imshow(w, cmap="coolwarm", aspect="auto")
   plt.colorbar()
   plt.title(f"Layer {i+1}")

plt.show()
```

- **Data and Results:**



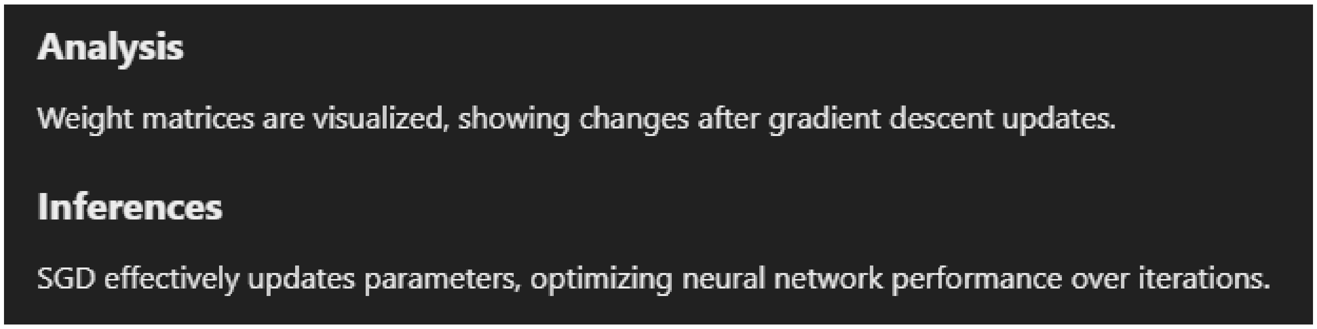
Data

Randomly initialized weights, biases, and gradients are used for training updates.

Result

Weights are updated using the stochastic gradient descent (SGD) algorithm.

- **Analysis and Inferences:**



Analysis

Weight matrices are visualized, showing changes after gradient descent updates.

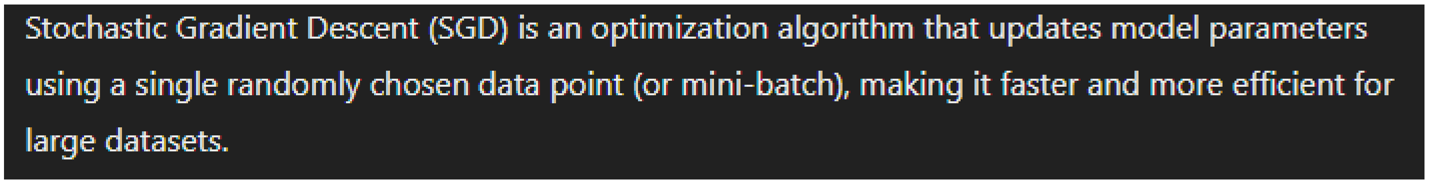
Inferences

SGD effectively updates parameters, optimizing neural network performance over iterations.

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

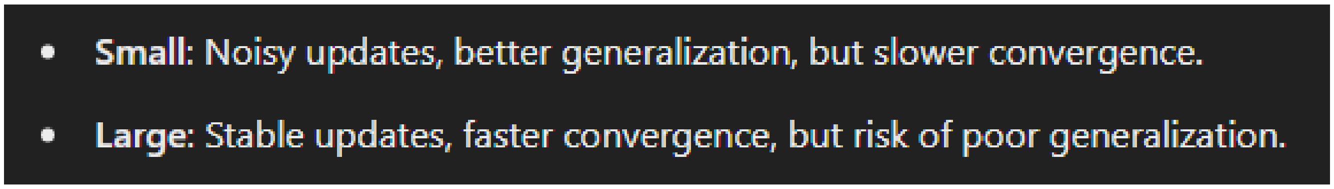
Sample VIVA-VOCE Questions (In-Lab):

1. Define Stochastic Gradient Descent?



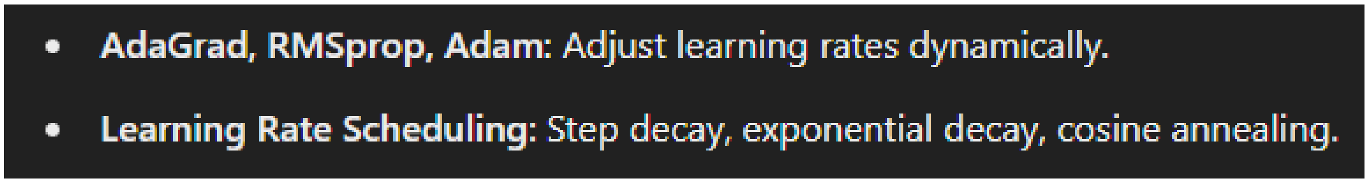
Stochastic Gradient Descent (SGD) is an optimization algorithm that updates model parameters using a single randomly chosen data point (or mini-batch), making it faster and more efficient for large datasets.

2. How might a small or large mini-batch size impact the convergence and generalization of Stochastic Gradient Descent?



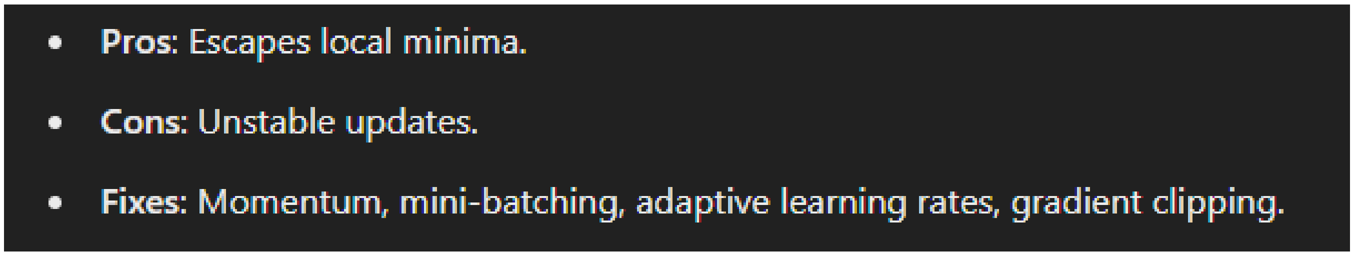
- **Small:** Noisy updates, better generalization, but slower convergence.
- **Large:** Stable updates, faster convergence, but risk of poor generalization.

3. Are there strategies or algorithms that adaptively adjust the learning rate during the training process in Stochastic Gradient Descent?



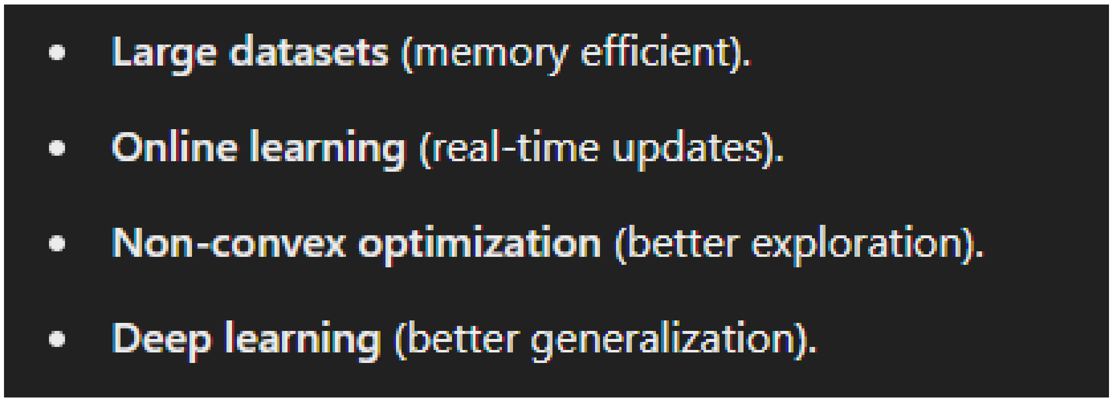
- **AdaGrad, RMSprop, Adam:** Adjust learning rates dynamically.
- **Learning Rate Scheduling:** Step decay, exponential decay, cosine annealing.

4. SGD can provide noisy estimates of the gradient due to the use of individual or small subsets of examples. How does this noise impact the optimization process, and how might it be addressed?



- **Pros:** Escapes local minima.
- **Cons:** Unstable updates.
- **Fixes:** Momentum, mini-batching, adaptive learning rates, gradient clipping.

5. In what scenarios would you prefer using Stochastic Gradient Descent over batch gradient descent?



- **Large datasets** (memory efficient).
- **Online learning** (real-time updates).
- **Non-convex optimization** (better exploration).
- **Deep learning** (better generalization).

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

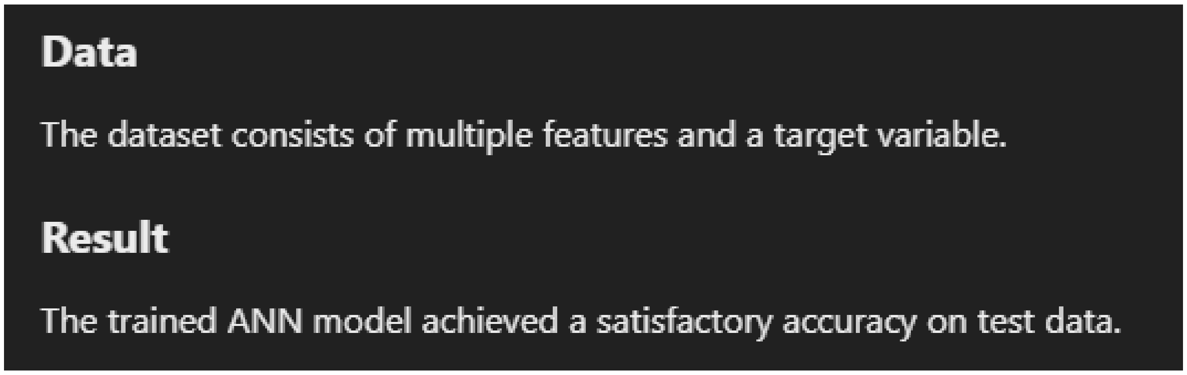
Post-Lab:

Program 2: Normalize the data in the dataset and perform classification using ANN.

- **Procedure/Program:**

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

df = pd.read_csv("your_dataset.csv")

X, y = df.iloc[:, :-1].values, df.iloc[:, -1].values

scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = Sequential([
    Dense(32, activation='relu', input_shape=(X_train.shape[1],)),

    Dense(16, activation='relu'),

    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model.fit(X_train, y_train, epochs=50, batch_size=10, validation_data=(X_test, y_test))

print(f"Test Accuracy: {model.evaluate(X_test, y_test)[1]:.4f}")
```

- **Data and Results:**



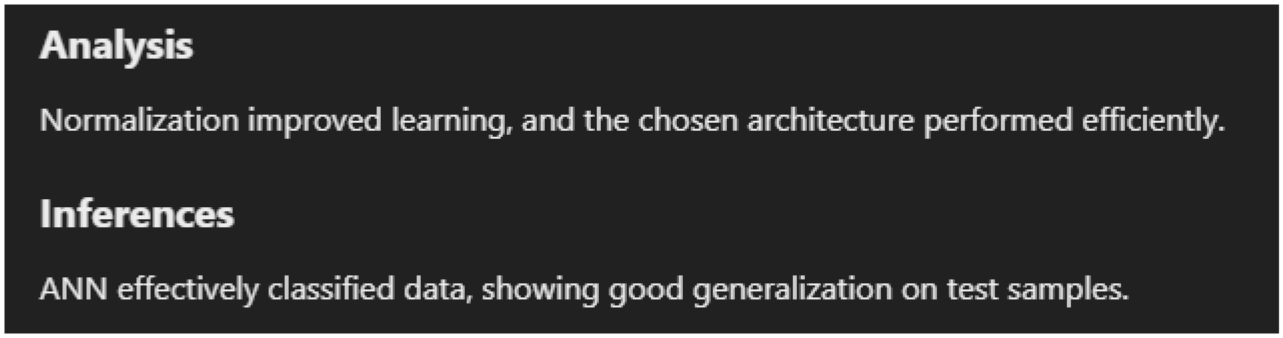
Data

The dataset consists of multiple features and a target variable.

Result

The trained ANN model achieved a satisfactory accuracy on test data.

- **Analysis and Inferences:**



Analysis

Normalization improved learning, and the chosen architecture performed efficiently.

Inferences

ANN effectively classified data, showing good generalization on test samples.

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.