36 Karan Naik

Experiment 04

Gradient Descent

```
import numpy as np

def gradient_descent(X, y, learning_rate, num_iterations):
    num_samples, num_features = X.shape
    theta = np.zeros(num_features)  # Initialize theta with zeros

    for _ in range(num_iterations):
        # Calculate predictions using current theta
        predictions = np.dot(X, theta)

        # Calculate the error between predictions and actual values
        error = predictions - y

        # Calculate the gradient
        gradient = np.dot(X.T, error) / num_samples

        # Update theta using gradient and learning rate
        theta -= learning_rate * gradient

    return theta

# Example usage
# Generate some random data for demonstration
np.random.seed(42)
num_samples = 100
num_features = 2
X = np.random.rand(num_samples, num_features)
true_theta = np.array([2, 3])
y = np.dot(X, true_theta) + np.random.normal(0, 0.1, num_samples)

learning_rate = 0.01
num_iterations = 1000
learned_theta = gradient_descent(X, y, learning_rate, num_iterations)

print("True theta:", true_theta)
print("Learned theta:", learned_theta)
```

```
True theta: [2 3] Learned theta: [2.21400966 2.79279662]
```

Stochastic Gradient Descent

```
import numpy as np
def stochastic_gradient_descent(X, y, learning_rate, num_epochs):
    num_samples, num_features = X.shape
    theta = np.zeros(num_features)  # Initialize theta with zeros

    for _ in range(num_epochs):
        for i in range(num_samples):
            # Choose a random sample
            random_index = np.random.randint(num_samples)
            xi = X[random_index]
            yi = y[random_index]

            # Calculate prediction for the current sample
            prediction = np.dot(xi, theta)

            # Calculate the error for the current sample
            error = prediction - yi

            # Calculate the gradient for the current sample
            gradient = xi * error

            # Update theta using gradient and learning rate
            theta -= learning_rate * gradient

    return theta

# Example usage
# Generate some random data for demonstration
np.random.seed(42)
num_samples = 100
num_features = 2
X = np.random.rand(num_samples, num_features)
true_theta = np.array([2, 3])
y = np.dot(X, true_theta) + np.random.normal(0, 0.1, num_samples)

learning_rate = 0.01
num_epochs = 100
learned_theta = stochastic_gradient_descent(X, y, learning_rate, num_epochs)

print("True theta:", true_theta)
print("Learned theta:", learned_theta)
```

```
True theta: [2 3] Learned theta: [2.02047617 3.01444191]
```

Adam Gradient Descent

```
import numpy as np
import tensorflow as tf

# Generate random data for demonstration
np.random.seed(42)
num_samples = 100
num_features = 2
X = np.random.rand(num_samples, num_features)
true_theta = np.array([2, 3])
y = np.dot(X, true_theta) + np.random.normal(0, 0.1, num_samples)

# Convert data to TensorFlow tensors
X_tf = tf.constant(X, dtype=tf.float32)
y_tf = tf.constant(y, dtype=tf.float32)

# Define variables
theta = tf.Variable(tf.zeros(num_features, dtype=tf.float32), name="theta")

# Define the prediction and loss
predictions = tf.linalg.matvec(X_tf, theta)
loss = tf.reduce_mean(tf.square(predictions - y_tf))

# Define the optimizer (Adam optimizer)
learning_rate = 0.01
optimizer = tf.optimizers.Adam(learning_rate)

# Perform optimization using Adam optimizer
num_epochs = 1000
for epoch in range(num_epochs):
    with tf.GradientTape() as tape:
        predictions = tf.linalg.matvec(X_tf, theta)
        loss = tf.reduce_mean(tf.square(predictions - y_tf))
    gradients = tape.gradient(loss, theta)
    optimizer.apply_gradients([(gradients, theta)])

learned_theta = theta.numpy()

print("True theta:", true_theta)
print("Learned theta using Adam optimizer:", learned_theta)
```

```
True theta: [2 3] Learned theta using Adam optimizer: [2.140685 2.8843906]
```