02 Gradient Descent (GD)

- Gradient descent is an optimization algorithm used to minimize the cost function in machine learning and deep learning models
- It is a fundamental concept for training models by adjusting parameters (weights) to reduce the error in predictions

01 Working of GD

- Gradient descent works by iteratively moving towards the minimum of the cost function (often referred to as the loss function) by updating the model's parameters in the opposite direction of the gradient of the cost function with respect to the parameters

02 Mathematical Formulation

Let (J(\theta)) be the cost function, where (\theta) represents the parameters. The update rule for gradient descent is:

$$\theta = \theta - \alpha \cdot \nabla J(\theta)$$

- theta : Parameters (weights) to be updated
- alpha : Learning rate (a small positive number)
- $\nabla J(\theta)$: Gradient of the cost function with respect to the parameters

02 Types of GD

1. **Batch Gradient Descent**
2. **Stochastic Gradient Descent (SGD)**
3. **Mini-batch Gradient Descent**

01 Batch Gradient Descent

- In batch gradient descent, the gradient is calculated using the entire dataset
- It is computationally expensive for large datasets but guarantees convergence to the global minimum for convex functions
- Suitable for small datasets where computation can be done quickly, uses the whole dataset, stable but slow

$$\theta = \theta - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \nabla J(\theta; x^{(i)}, y^{(i)})$$

m : Number of training examples

Example Code

```
import numpy as np

# Sample data
X = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])

# Parameters
theta = 0
alpha = 0.01
iterations = 1000
m = len(y)

# Batch Gradient Descent
for i in range(iterations):
    gradient = (1/m) * sum((theta * X - y) * X)
    theta = theta - alpha * gradient

print(f"Optimized theta: {theta}")
```

02 Stochastic Gradient Descent (SGD)

- In stochastic gradient descent, the gradient is calculated using only one randomly selected training example at a time
- It is faster than batch gradient descent but introduces more noise in the updates, leading to possible oscillations around the minimum
- Best for large datasets where a quick approximation is sufficient, uses one data point at a time, fast but noisy

$\theta = \theta - \alpha \cdot \nabla J(\theta; x^{(i)}, y^{(i)})$

Example Code

```
import numpy as np

# Sample data
X = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])

# Parameters
theta = 0
alpha = 0.01
iterations = 1000
```

```
m = len(y)

# Stochastic Gradient Descent
for i in range(iterations):
    for j in range(m):
        index = np.random.randint(m)
        gradient = (theta * X[index] - y[index]) * X[index]
        theta = theta - alpha * gradient

print(f"Optimized theta: {theta}")
```

03 Mini-batch Gradient Descent

- Mini-batch gradient descent is a compromise between batch and stochastic gradient descent
- The gradient is calculated using a small batch of training examples
- It is more efficient than batch gradient descent and reduces the noise present in stochastic gradient descent
- Offers a good balance between the two, often preferred for training neural networks, uses a small batch of data points, balanced in speed and stability

$\theta = \theta - \alpha \cdot \frac{1}{b} \sum_{i=1}^{b} \nabla J(\theta; x^{(i)}, y^{(i)})$

b : mini-batch size

Example Code

```
import numpy as np

# Sample data
X = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])

# Parameters
theta = 0
alpha = 0.01
iterations = 1000
batch_size = 2
m = len(y)

# Mini-batch Gradient Descent
for i in range(iterations):
    indices = np.random.permutation(m)
    X_shuffled = X[indices]
    y_shuffled = y[indices]
    for j in range(0, m, batch_size):
        X_batch = X_shuffled[j:j+batch_size]
        y_batch = y_shuffled[j:j+batch_size]
        gradient = (1/batch_size) * sum((theta * X_batch - y_batch) *
X_batch)
        theta = theta - alpha * gradient

print(f"Optimized theta: {theta}")
```