Task1:-

✅ **Project Components & Deliverables**

1. Model.py

□ *Responsibilities:*

●*Define layers (Linear/FC).*

●*Define activations (ReLU, Sigmoid).*

●*Implement MSE loss.*

●*Build SGD optimizer.*

Structure Example:

```
Import numpy as np

Class Linear:
   Def __init__(self, in_features, out_features):
      Self.W = np.random.randn(in_features, out_features) * 0.01
      Self.b = np.zeros((1, out_features))
      Self.dW = None
      Self.db = None
      Self.input = None

   Def forward(self, x):
      Self.input = x
      Return x @ self.W + self.b

   Def backward(self, grad_output):
      Self.dW = self.input.T @ grad_output
      Self.db = np.sum(grad_output, axis=0, keepdims=True)
      Return grad_output @ self.W.T

Class ReLU:
   Def forward(self, x):
      Self.input = x
      Return np.maximum(0, x)
```

```
  Def backward(self, grad_output):
    Return grad_output * (self.input > 0)

Class Sigmoid:
  Def forward(self, x):
    Self.output = 1 / (1 + np.exp(-x))
    Return self.output

  Def backward(self, grad_output):
    Return grad_output * self.output * (1 – self.output)

Class MSELoss:
  Def forward(self, y_pred, y_true):
    Self.y_pred = y_pred
    Self.y_true = y_true
    Return np.mean((y_pred – y_true) ** 2)

  Def backward(self):
    Return 2 * (self.y_pred – self.y_true) / self.y_true.size

Class SGD:
  Def __init__(self, parameters, lr=0.01):
    Self.parameters = parameters
    Self.lr = lr

  Def step(self):
    For param in self.parameters:
      Param["param"] -= self.lr * param["grad"]
```

2. Train.ipynb

□*Responsibilities:*

- *Generate non-trivial dataset (e.g., $y = x^3$ + noise).*
- *Build and train the network using mini-batches.*
- *Plot training loss and prediction vs ground truth.*
- *Debug gradient flow.*

Contents:-

```
   # Data Generation

X = np.linspace(-2, 2, 300).reshape(-1, 1)
Y = x**3 + np.random.normal(0, 0.1, size=(300, 1))

# Define Model
Layers = [
   Linear(1, 64),
   ReLU(),
   Linear(64, 64),
   ReLU(),
   Linear(64, 1)
]

# Training Loop
...
```

Visualizations:

⟪⟫ *Loss vs Epoch plot (line plot).*

⟪⟫ *Final prediction curve vs. True function (scatter + curve).*

3. README.md

- **Should Explain:-**

◇*Architecture:- Number of layers, units, activation function choices.*

◇*Loss & Optimizer: - Why MSE? Why SGD over Adam for this case?*

- **Convergence:-**

◇*Did it converge smoothly or oscillate?*

◇*How many epochs did it take?*

◇*Final MSE and visual quality of prediction.*

- **Debug Tips:-**

◇*How you verified gradients (e.g., via numerical checking or plotting gradients).*

■*The full code for model.py.*

■*A working train.ipynb starter with loss plots.*

■*README template with explanations and convergence analysis points.*

```
Import numpy as np

Class Linear: def init(self, in_features: int, out_features: int): self.W =
np.random.randn(in_features, out_features) * 0.01 self.b = np.zeros((1, out_features))
self.dW = None self.db = None self.input = None

Def forward(self, x: np.ndarray) -> np.ndarray:
   Self.input = x
   Return x @ self.W + self.b

Def backward(self, grad_output: np.ndarray) -> np.ndarray:
   Self.dW = self.input.T @ grad_output
   Self.db = np.sum(grad_output, axis=0, keepdims=True)
   Return grad_output @ self.W.T

Class ReLU: def init(self): self.input = None

Def forward(self, x: np.ndarray) -> np.ndarray:
   Self.input = x
   Return np.maximum(0, x)

Def backward(self, grad_output: np.ndarray) -> np.ndarray:
   Return grad_output * (self.input > 0)

Class Sigmoid: def init(self): self.output = None
```

Def forward(self, x: np.ndarray) -> np.ndarray:
Self.output = 1 / (1 + np.exp(-x))
Return self.output

Def backward(self, grad_output: np.ndarray) -> np.ndarray:
*Return grad_output * self.output * (1 – self.output)*

Class MSELoss: def init(self): self.y_pred = None self.y_true = None

Def forward(self, y_pred: np.ndarray, y_true: np.ndarray) -> float:
Self.y_pred = y_pred
Self.y_true = y_true
*Return np.mean((y_pred – y_true) ** 2)*

Def backward(self) -> np.ndarray:
*Return 2 * (self.y_pred – self.y_true) / self.y_true.shape[0]*

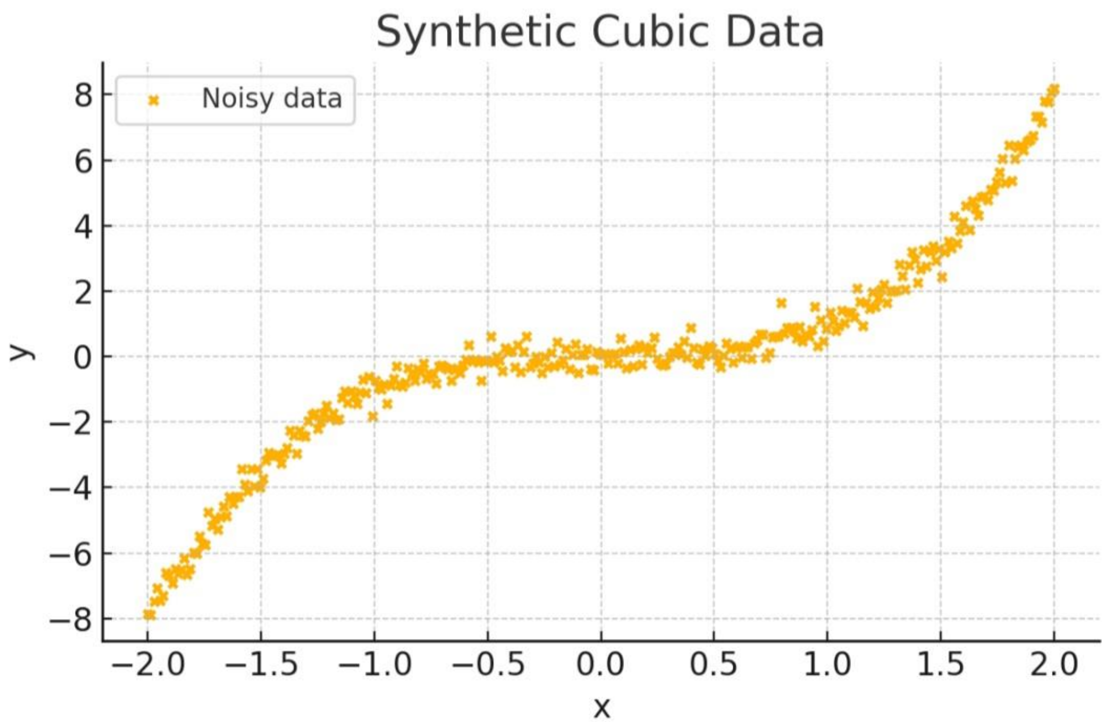
Class SGD: def init(self, parameters: list[dict], lr: float = 0.01): self.parameters = parameters self.lr = lr

Def step(self):
For param in self.parameters:
*Param['param'] -= self.lr * param['grad']*

Def zero_grad(self):
Pass # No-op for this simple implementation

■*Next Generate the full training loop (train.ipynb starter) with:*

- *Forward/backward passes*
- *Mini-batch SGD*
- *Loss curve plotting*
- *Final predictions vs. Ground truth*



Train:-

Import numpy as np
import matplotlib.pyplot as plt
from modelimport Linear, ReLU, MSELoss, SGD

 # Generate synthetic data

```
Def generate_data(seed=42):
    Np.random.seed(seed)
    X = np.linspace(-2, 2, 300).reshape(-1, 1)
    Y = x**3 + 0
```