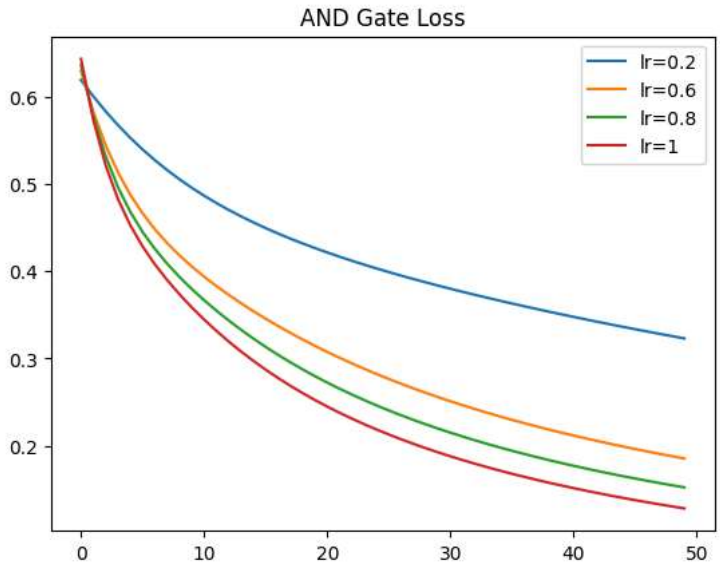
```
import numpy as np
import matplotlib.pyplot as plt
# Inputs
x1, x2 = 0.6, 0.1
y = 1

# Parameters
w1, w2, b = 0.2, -0.3, 0.4
lr = 0.1

# Sigmoid
def sigmoid(z):
    return 1/(1+np.exp(-z))

# Forward pass
z = w1*x1 + w2*x2 + b
y_pred = sigmoid(z)

# Loss derivative
dL_dy_pred = y_pred - y

# Sigmoid derivative
Dy_pred_dz = y_pred*(1-y_pred)

# Chain rule
dL_dz = dL_dy_pred * Dy_pred_dz

# Gradients
dL_dw1 = dL_dz * x1
dL_dw2 = dL_dz * x2
dL_db = dL_dz

# Update
w1 -= lr*dL_dw1
w2 -= lr*dL_dw2
b  -= lr*dL_db

print(w1, w2, b)
```

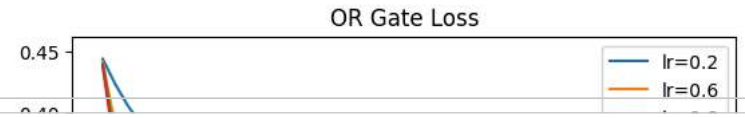
```
0.20536959297953516 -0.2991050678367441 0.4089493216325586
```

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

# Gate data
X = [(0,0),(0,1),(1,0),(1,1)]
Y_AND = [0,0,0,1]
Y_OR  = [0,1,1,1]

# Sigmoid
def sigmoid(z):
    return 1/(1+np.exp(-z))

# training function (same step style)
def train_gate(Y, lr):

    w1, w2, b = 0.2, -0.3, 0.4
    losses = []

    for epoch in range(50):
        total_loss = 0

        for (x1,x2), y in zip(X,Y):

            # Forward
            z = w1*x1 + w2*x2 + b
            y_pred = sigmoid(z)

            loss = 0.5*(y_pred-y)**2
            total_loss += loss

            # Backprop
            dL_dy_pred = y_pred - y
            dy_pred_dz = y_pred*(1-y_pred)
```

```
            dL_dz = dL_dy_pred * dy_pred_dz

            dL_dw1 = dL_dz * x1
            dL_dw2 = dL_dz * x2
            dL_db  = dL_dz

            # Update
            w1 -= lr*dL_dw1
            w2 -= lr*dL_dw2
            b  -= lr*dL_db

        losses.append(total_loss)

    return losses

learning_rates = [0.2,0.6,0.8,1]

# AND plot
plt.figure()
for lr in learning_rates:
    losses = train_gate(Y_AND, lr)
    plt.plot(losses,label=f"lr={lr}")
plt.title("AND Gate Loss")
plt.legend()
plt.show()
```




```
plt.figure()
for lr in learning_rates:
    losses = train_gate(Y_OR, lr)
    plt.plot(losses,label=f"lr={lr}")
plt.title("OR Gate Loss")
plt.legend()
plt.show()
```




```
import numpy as np

# AND data
X = np.array([
    [0,0],
    [0,1],
    [1,0],
    [1,1]
])

y = np.array([0,0,0,1])

# Parameters
w = np.array([0.1, 0.2])
b = 0.3
lr = 0.1

def sigmoid(z):
    return 1/(1+np.exp(-z))

# Training
for epoch in range(1000):
    for i in range(len(X)):
        z = np.dot(X[i], w) + b
        y_pred = sigmoid(z)

        # BCE + Sigmoid gradient shortcut
        dz = y_pred - y[i]
        dw = dz * X[i]
        db = dz
        w -= lr * dw
        b -= lr * db

print("Weights:", w)
print("Bias:", b)
```

```
Weights: [5.60154076 5.59544869]
Bias: -8.565987677634602
```

```
import numpy as np

# OR data
X = np.array([
    [0,0],
    [0,1],
    [1,0],
    [1,1]
])

y = np.array([0,1,1,1])

# Parameters
w = np.array([0.1, 0.2])
b = 0.3
lr = 0.1

def sigmoid(z):
    return 1/(1+np.exp(-z))

# Training
for epoch in range(1000):
    for i in range(len(X)):
        z = np.dot(X[i], w) + b
        y_pred = sigmoid(z)

        # BCE + Sigmoid gradient shortcut
        dz = y_pred - y[i]
        dw = dz * X[i]
        db = dz
        w -= lr * dw
        b -= lr * db
```

```
            b -= lr * db

print("Weights:", w)
print("Bias:", b)
```

```
Weights: [6.7832796  6.78742098]
Bias: -2.9139465005185365
```