09/09/2026.

Exp: 6

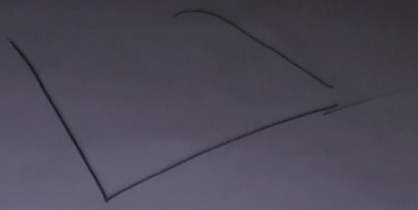
6. Implement gradient descent and back Propagation in deep neural network.

Aim:

To implement a deep neural network using pytorch and train it with gradient decent and backpropagation for binary classification on a synthetic dataset

OBJECTIVE:

1. To create a simple dataset for training a neural network
2. To design and implement a deep neural network with multiple layer
3. To apply forward propagation for computing predictions
4. To use backpropagation for calculating gradients of loss, with weights
5. To update the network parameters using gradient descent.
6. To Evaluate the performance of the trained modeling using accuracy.



Output

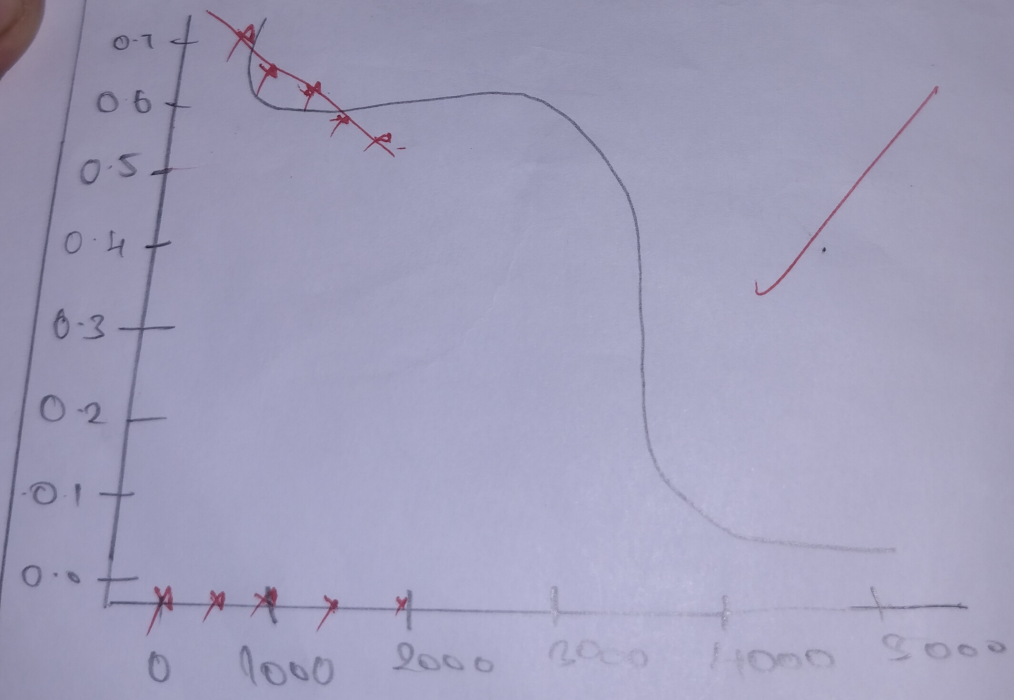
epoch 0 - loss : 0.7198

epoch 500 - Loss : 0.6869

epoch 1000 - loss : 0.6755

epoch 1500 - loss : 0.6402

epoch 2000 - loss : 0.5343



Pseudocode:-

1. Import Pytorch libraries
2. Generate synthetic dataset X (features) and y (labels)
3. Define a deep neural network class
   - Input layer → Hidden layers → output layer
   - Apply ReLU activation in hidden layer.
4. Initialize
   - Loss function (Cross Entropy Loss)
   - Optimizer (SGD with learning rate)
5. For each epoch in training loop:
   a. perform forward pass & compute Predictions
   b. Calculate loss between predictions and actual labels.
   c. Zero gradients of model parameter.

Observations:-

→ The loss decreases steadily with each epoch, showing that the model is learning

→ The accuracy improves significantly indicating successful training

→ Using ReLU activation helps in faster convergence

→ The network efficiently separates data into two classes using learned weights.

Result:

Therefore implementation of gradient descent and backpropagation in neural network

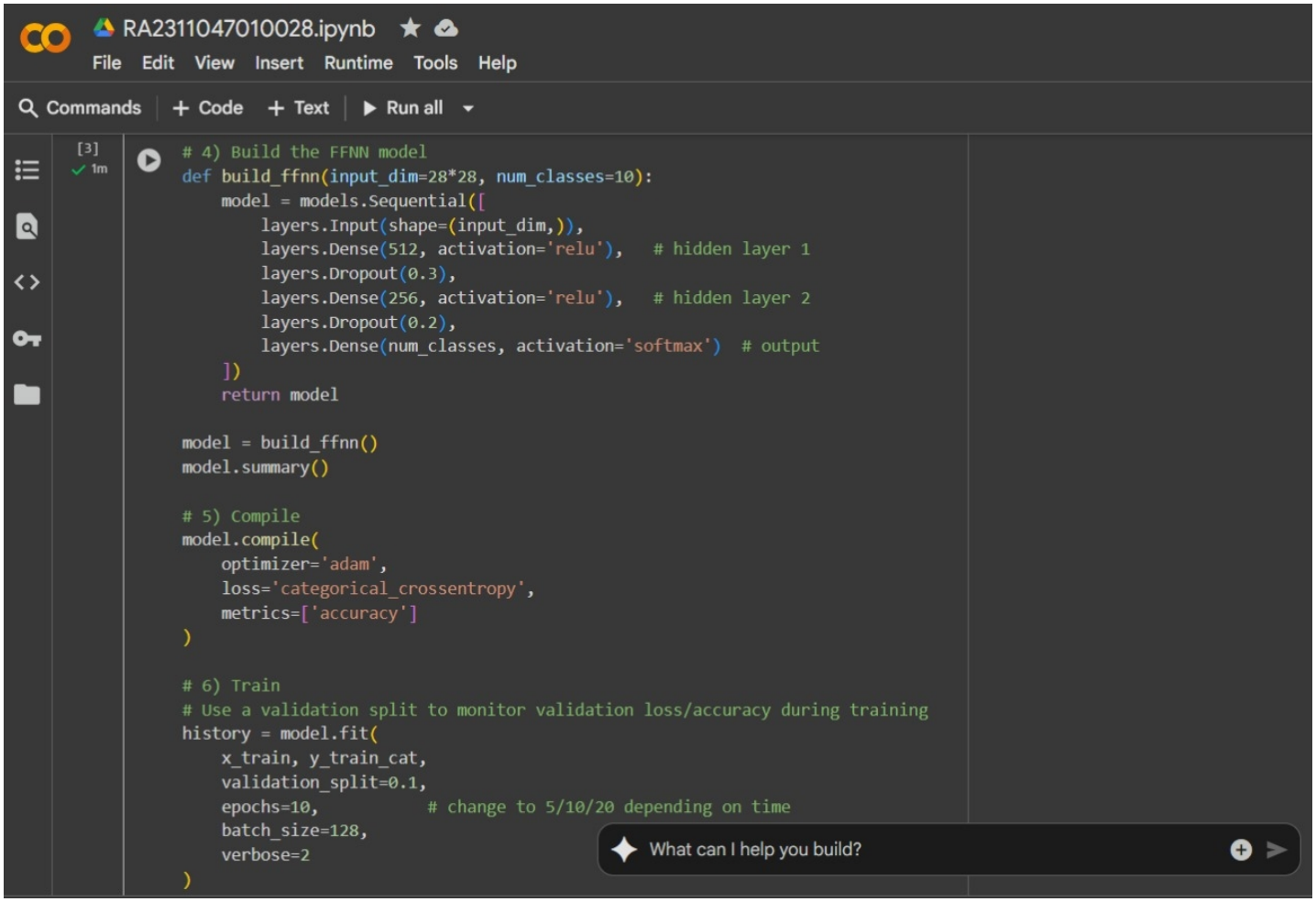
RA2311047010028.ipynb

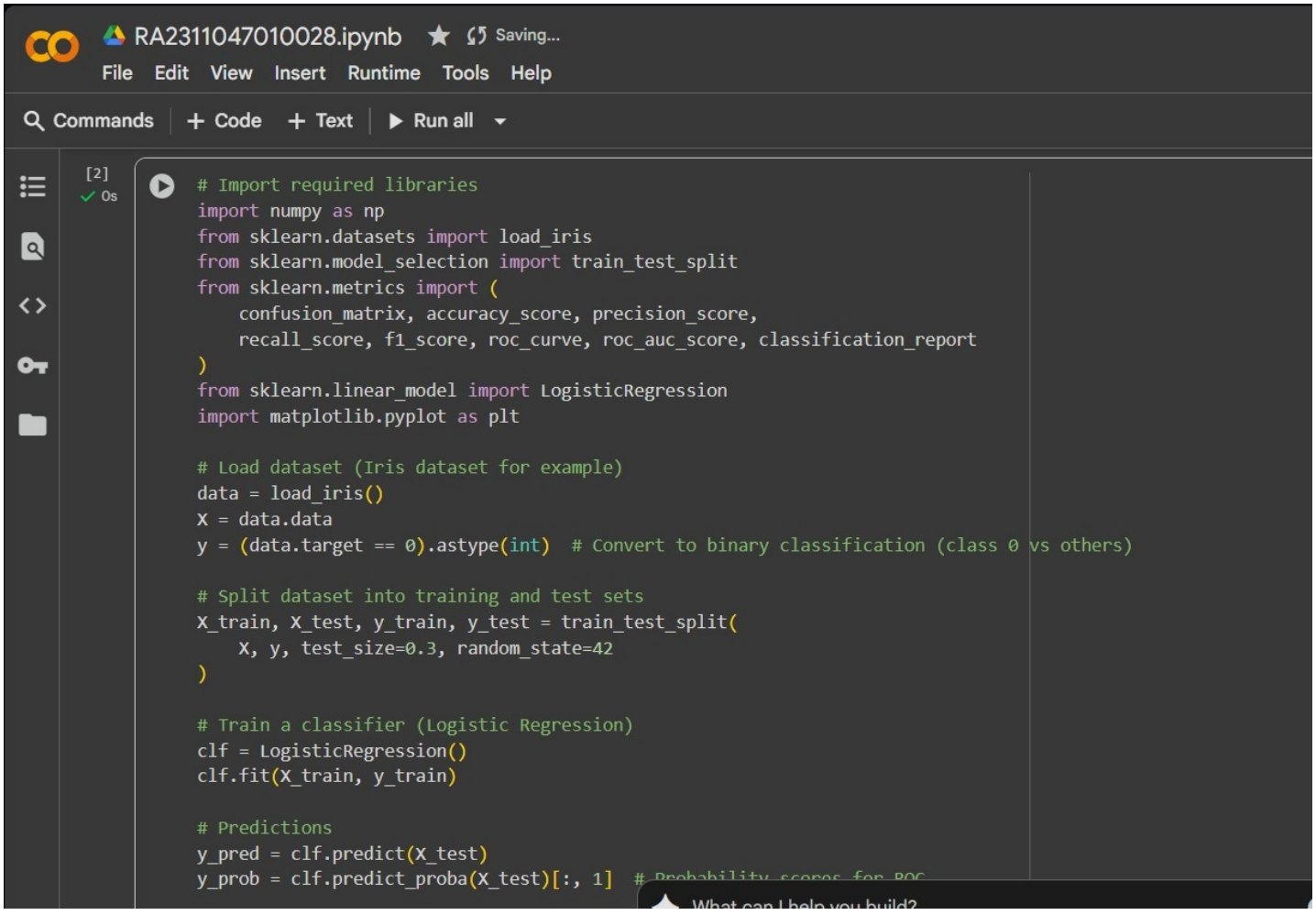
File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

[2] ✓ 0s

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

# Sigmoid activation function and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Training dataset (XOR problem)
X = np.array([[0,0],
              [0,1],
              [1,0],
              [1,1]])

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

# Seed for reproducibility
np.random.seed(42)

# Initialize weights and biases randomly
input_layer_neurons = 2   # Input features
hidden_layer_neurons = 2  # Hidden layer
output_neurons = 1        # Output

# Weights
```


RA2311047010028.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
[3]
1m
# 4) Build the FFNN model
def build_ffnn(input_dim=28*28, num_classes=10):
model = models.Sequential([
layers.Input(shape=(input_dim,)),
layers.Dense(512, activation='relu'), # hidden layer 1
layers.Dropout(0.3),
layers.Dense(256, activation='relu'), # hidden layer 2
layers.Dropout(0.2),
layers.Dense(num_classes, activation='softmax') # output
])
return model
model = build_ffnn()
model.summary()
# 5) Compile
model.compile(
optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy']
)
# 6) Train
# Use a validation split to monitor validation loss/accuracy during training
history = model.fit(
x_train, y_train_cat,
validation_split=0.1,
epochs=10, # change to 5/10/20 depending on time
batch_size=128,
verbose=2
)
What can I help you build?

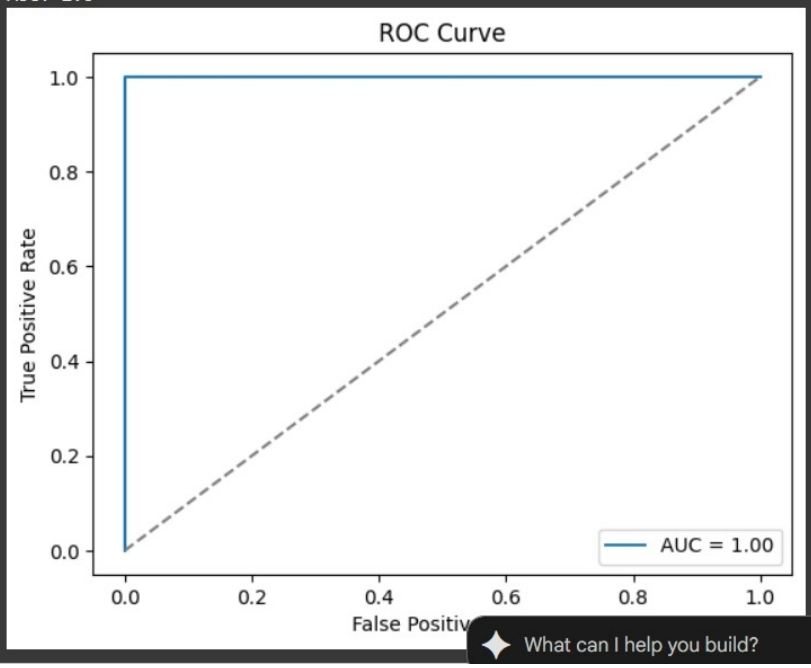

RA2311047010028.ipynb
Saving...
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
[2]
0s
# Import required libraries
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
confusion_matrix, accuracy_score, precision_score,
recall_score, f1_score, roc_curve, roc_auc_score, classification_report
)
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
# Load dataset (Iris dataset for example)
data = load_iris()
X = data.data
y = (data.target == 0).astype(int) # Convert to binary classification (class 0 vs others)
# Split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.3, random_state=42
)
# Train a classifier (Logistic Regression)
clf = LogisticRegression()
clf.fit(X_train, y_train)
# Predictions
y_pred = clf.predict(X_test)
y_prob = clf.predict_proba(X_test)[:, 1]
What can I help you build?

RA2311047010028.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
              accuracy                           1.00        45
             macro avg       1.00      1.00      1.00        45
          weighted avg       1.00      1.00      1.00        45

AUC: 1.0
```



What can I help you build?