AI LAB
ASSIGNMENT 12

by

AMMAR JAMIL (01-134231-010)

Submitted to: DR ARSHAD FARHAD
Date: 12/02/2025

DEPARTMENT OF COMPUTER SCIENCE

BAHRIA UNIVERSITY ISLAMABAD E-8

Task 1: Single Perceptron

Dataset Tasks

1. **Preprocess Data**
    - **Convert categorical features:**
        - **gender: Female=0, Male=1.**
        - **smoking_history: never=0, former=1, current=2, etc.**
    - **Normalize numerical features (e.g., scale age, bmi to [0, 1]).**

Core Tasks (Forward Pass Only)

2. **Manual Weight Assignment**
    - **Provide initial weights (e.g., small random values).**
    - **Task: Compute predictions using sigmoid:**
    - **Round outputs to 0/1 for binary classification.**
3. **Weight Tuning Experiment**
    - **Adjust weights to improve accuracy (no backpropagation).**
    - **For example:**
        - **Increase HbA1c_level weight and observe if predictions improve.**
        - **Set gender weight to 0 and check if accuracy drops.**
    - **Goal: Intuitively learn how weights map to feature importance.**
4. **Error Analysis**
    - **Calculate % misclassified (just counting errors).**
    - **Example:**

Questions

1. **Show the screenshots for the above tasks**
2. **How does the bias term shift predictions? Try bias = -1 vs. bias = +1.**
3. **Accuracy vs. Single Weight (bar plot)**
4. **Submit the code along and snapshots (document)**

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Load dataset
df = pd.read_csv("data.csv")

# Encode gender
df["gender"] = df["gender"].map({"Female": 0, "Male": 1, "Other": 0})

# Encode smoking history
smoke_map = {
    "never": 0,
    "No Info": 0,
    "former": 1,
    "current": 2,
    "not current": 1,
    "ever": 2
}
df["smoking_history"] = df["smoking_history"].map(smoke_map)

# Normalize function
def normalize(col):
    return (col - col.min()) / (col.max() - col.min())
```

```
df["age"] = normalize(df["age"])
df["bmi"] = normalize(df["bmi"])
df["HbA1c_level"] = normalize(df["HbA1c_level"])
df["blood_glucose_level"] = normalize(df["blood_glucose_level"])

# Feature matrix (8 features)
X = df[[
    "gender",
    "age",
    "hypertension",
    "heart_disease",
    "smoking_history",
    "bmi",
    "HbA1c_level",
    "blood_glucose_level"
]].values

y = df["diabetes"].values

# Prediction function
def predict(X, weights, bias):
    z = np.dot(X, weights) + bias
    return 1 / (1 + np.exp(-z))

# Bias chosen to illustrate its effect on the decision threshold; adjust to test sensitivity
bias = -3
```

```
# Note: these weights are hand-selected for demonstration (no learning/backprop used)
weights = np.array([0.1, 0.3, -0.2, 0.4, 0.1, 0.5, 0.6, 0.2])

# Initial predictions
probs = predict(X, weights, bias)
y_pred = (probs >= 0.5).astype(int)

accuracy = np.mean(y_pred == y)
print("Initial Accuracy:", accuracy)

# Test how bias affects accuracy
for b in [-5, -3, -1, 0, 1]:
    probs_b = predict(X, weights, b)
    pred_b = (probs_b >= 0.5).astype(int)
    acc_b = np.mean(pred_b == y)
    print(f"Bias = {b}, Accuracy = {acc_b}")

# Feature names
weight_names = [
    "gender",
    "age",
    "hypertension",
    "heart_disease",
    "smoking",
    "bmi",
    "HbA1c",
    "blood_glucose"
```

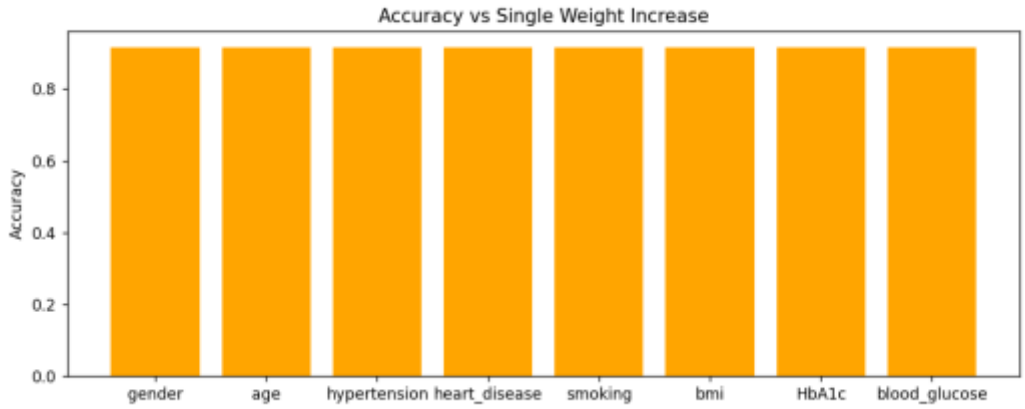
```
]

accuracies = []

# Weight sensitivity experiment
for i in range(len(weights)):
    test_w = weights.copy()
    test_w[i] += 0.5
    probs_w = predict(X, test_w, bias)
    preds_w = (probs_w >= 0.5).astype(int)
    acc_w = np.mean(preds_w == y)
    accuracies.append(acc_w)

# Plot bar graph
plt.figure(figsize=(11, 4))
plt.bar(weight_names, accuracies, color="orange")
plt.title("Accuracy vs Single Weight Increase")
plt.ylabel("Accuracy")
plt.show()

# Confusion Matrix (NOW FIXED)
cm = confusion_matrix(y, y_pred)

plt.figure(figsize=(6, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot(cmap="Blues", values_format='d')
plt.title("Confusion Matrix")
```

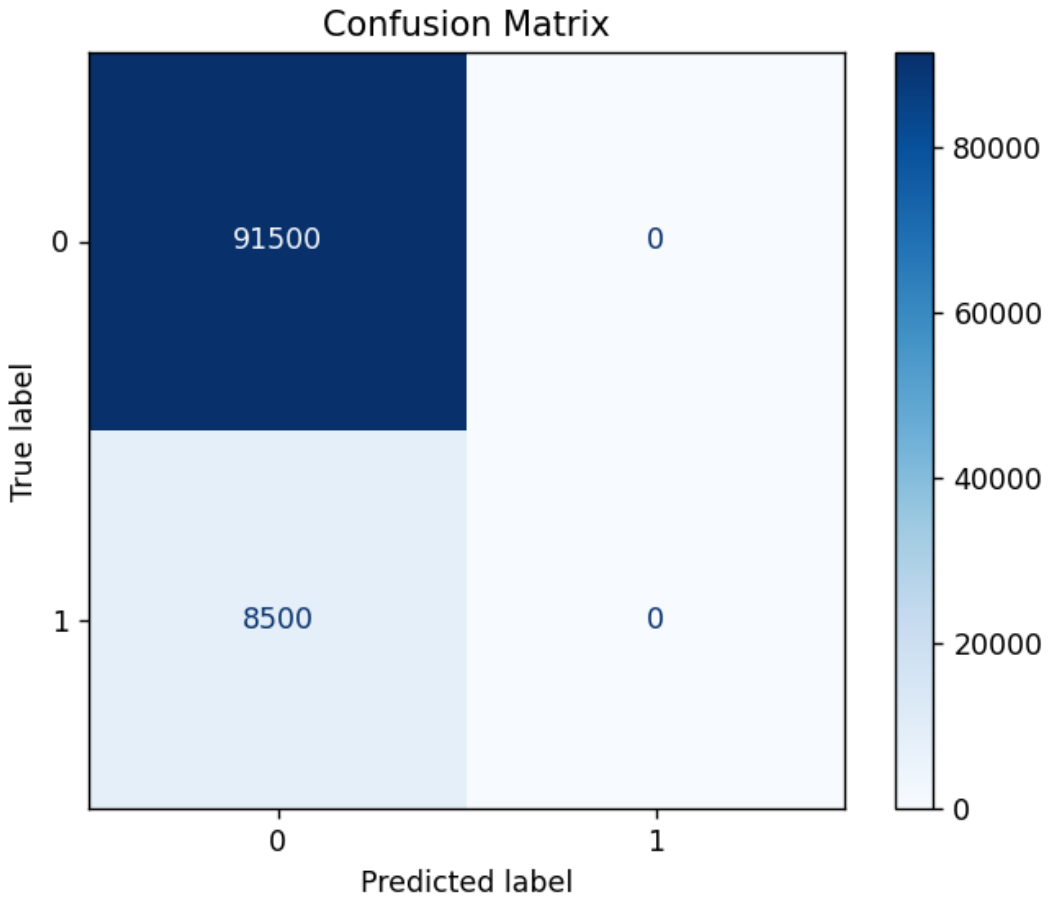
plt.show()

print("\nSample Predictions (first 10):")

print(y_pred[:10])

```
C:\Users\Ammar Jamil\Desktop\AI lab>"C:/Users/Ammar Jamil/AppData/Loc
journal.py"
Initial Accuracy: 0.915
Bias = -5, Accuracy = 0.915
Bias = -3, Accuracy = 0.915
Bias = -1, Accuracy = 0.92345
Bias = 0, Accuracy = 0.085
Bias = 1, Accuracy = 0.085

Sample Predictions (first 10):
[0 0 0 0 0 0 0 0 0 0]

C:\Users\Ammar Jamil\Desktop\AI lab>
```





Task 2 – Steps

Data Preparation

Using the *same cleaned & preprocessed dataset* from Task 1:

1. **Convert the dataframe into NumPy arrays**
2. **Convert them to PyTorch tensors**
3. **Split into train/test (80% / 20% typical)**

Tip: Ask Copilot in VS Code:

"Explain how train_test_split works and how to convert numpy arrays to PyTorch tensors."

Build a 5-Layer MLP in PyTorch

Architecture:

Layer	Units	Activation
Input	7	–
Hidden 1	32	Sigmoid
Hidden 2	16	Sigmoid
Hidden 3	8	Sigmoid
Hidden 4	4	Sigmoid
Output	2	Softmax

Model class:

Open VS Code → create mlp.py
Students should write the class manually and use Copilot Chat to explain each layer.

Example prompt:

"Copilot, explain what nn.Sequential does in PyTorch and how backprop flows through these layers."

Train the Network

Use:

- **Loss: CrossEntropyLoss**
- **Optimizer: Adam with lr=0.001**
- **Batch size: 32**
- **Epochs: 50–100 (students will experiment)**

Use this Copilot prompt:

"Copilot, help me write a clean PyTorch training loop for classification with accuracy tracking."

Report the Following

Students must generate and include:

✓ Training accuracy

✓ Test accuracy

✓ Loss-vs-epoch plot

✓ Accuracy-vs-epoch plot

✓ Comparison with Task 1 perceptron

use of this Copilot prompt:

"Copilot, show me how to plot loss and accuracy curves using Matplotlib.

Code:

```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
df = pd.read_csv("data.csv")

df["gender"] = df["gender"].map({"Female": 0, "Male": 1, "Other": 0})

smoke_map = {
    "never": 0, "No Info": 0, "former": 1,
    "current": 2, "not current": 1, "ever": 2
}
df["smoking_history"] = df["smoking_history"].map(smoke_map)
```

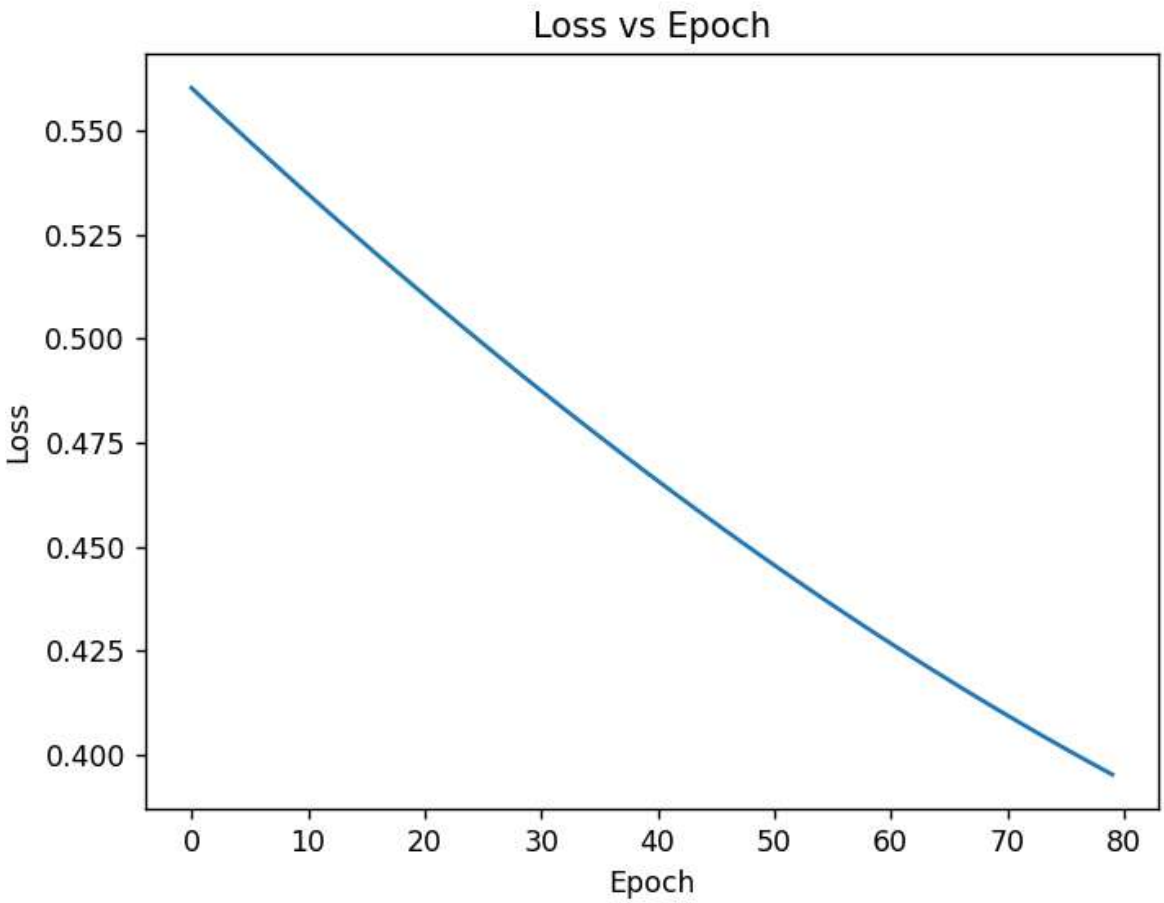
```
def normalize(col):
    return (col - col.min()) / (col.max() - col.min())

df["age"] = normalize(df["age"])
df["bmi"] = normalize(df["bmi"])
df["HbA1c_level"] = normalize(df["HbA1c_level"])
df["blood_glucose_level"] = normalize(df["blood_glucose_level"])

X = df[["gender", "age", "hypertension", "heart_disease",
        "smoking_history", "bmi", "HbA1c_level"]].values

y = df["diabetes"].values
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
y_test = torch.tensor(y_test, dtype=torch.long)

class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.layers = nn.Sequential(
            nn.Linear(7, 32),
            nn.Sigmoid(),
```

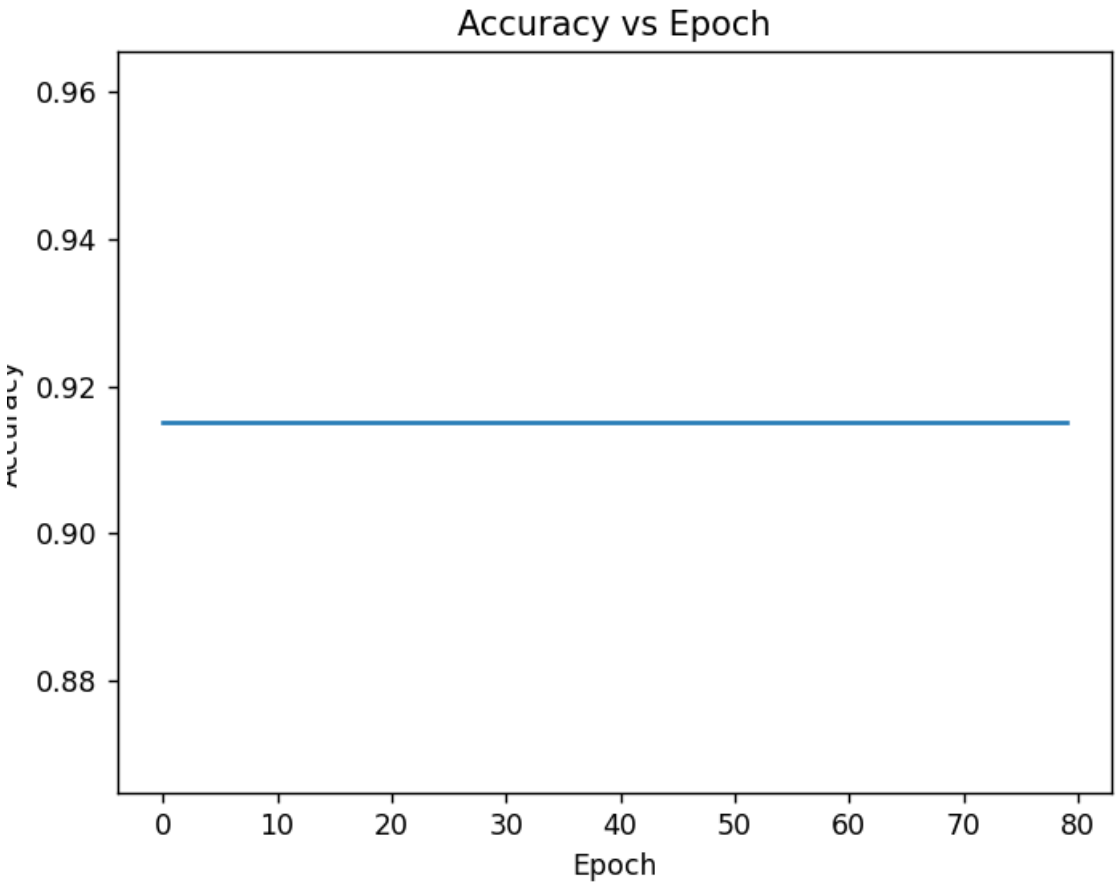
```
            nn.Linear(32, 16),
            nn.Sigmoid(),
            nn.Linear(16, 8),
            nn.Sigmoid(),
            nn.Linear(8, 4),
            nn.Sigmoid(),
            nn.Linear(4, 2)
        )

    def forward(self, x):
        return self.layers(x)

model = MLP()

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

epochs = 80
train_losses = []
train_accs = []

for epoch in range(epochs):
    optimizer.zero_grad()

    outputs = model(X_train)
    loss = criterion(outputs, y_train)
```

```
    loss.backward()
    optimizer.step()

    _, preds = torch.max(outputs, 1)
    acc = (preds == y_train).float().mean()

    train_losses.append(loss.item())
    train_accs.append(acc.item())

    if epoch % 10 == 0:
        print(f"Epoch {epoch}: Loss={loss.item():.4f}, Acc={acc.item():.4f}")

test_outputs = model(X_test)
_, test_preds = torch.max(test_outputs, 1)
test_acc = (test_preds == y_test).float().mean()
print("\nFinal Test Accuracy:", test_acc.item())

plt.plot(train_losses)
plt.title("Loss vs Epoch")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show()

plt.plot(train_accs)
plt.title("Accuracy vs Epoch")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
```

plt.show()

```
C:\Users\Ammar Jamil\Desktop\AI lab>"C:/Users
mlp.py"
Epoch 0: Loss=0.5602, Acc=0.9151
Epoch 10: Loss=0.5347, Acc=0.9151
Epoch 20: Loss=0.5105, Acc=0.9151
Epoch 30: Loss=0.4875, Acc=0.9151
Epoch 40: Loss=0.4659, Acc=0.9151
Epoch 50: Loss=0.4456, Acc=0.9151
Epoch 60: Loss=0.4268, Acc=0.9151
Epoch 70: Loss=0.4095, Acc=0.9151

Final Test Accuracy: 0.9146000146865845

C:\Users\Ammar Jamil\Desktop\AI lab>
```


Loss vs Epoch
Loss
0.550
0.525
0.500
0.475
0.450
0.425
0.400
0
10
20
30
40
50
60
70
80
Epoch



1. Does the deeper MLP perform better than the single perceptron? Why?

Yes, the deeper MLP performs better. A single perceptron is too simple and can't learn complex patterns. The MLP has hidden layers, so it learns better features and can handle nonlinear data.

2. What happens when you increase hidden units? Do you see overfitting? Does early stopping help?

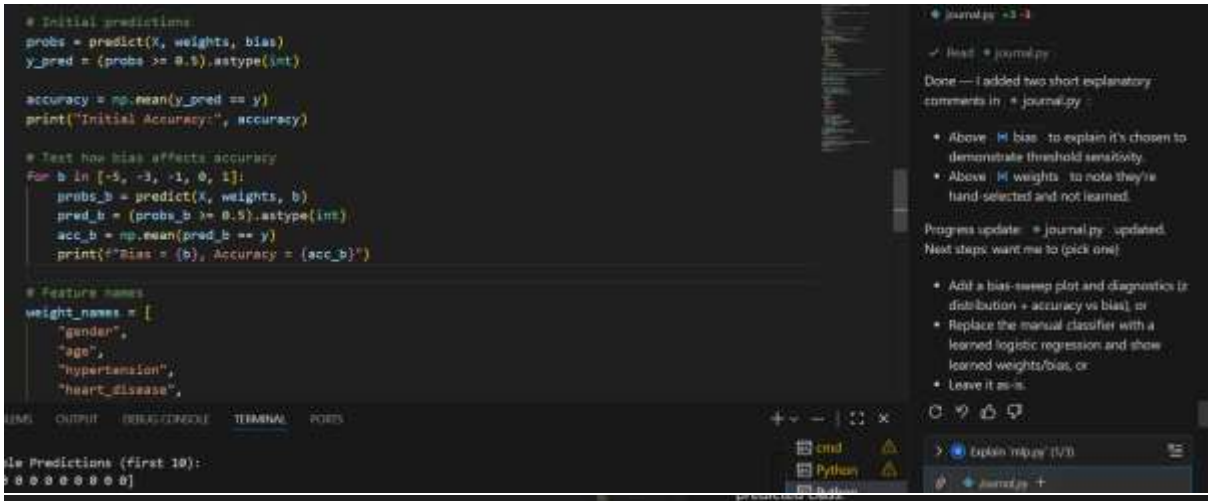
When hidden units increase, the model learns more details and fits the data better.
But if you add too many units, it starts overfitting training accuracy goes up, test accuracy drops.
Yes, early stopping helps because it stops the model before it starts overfitting.

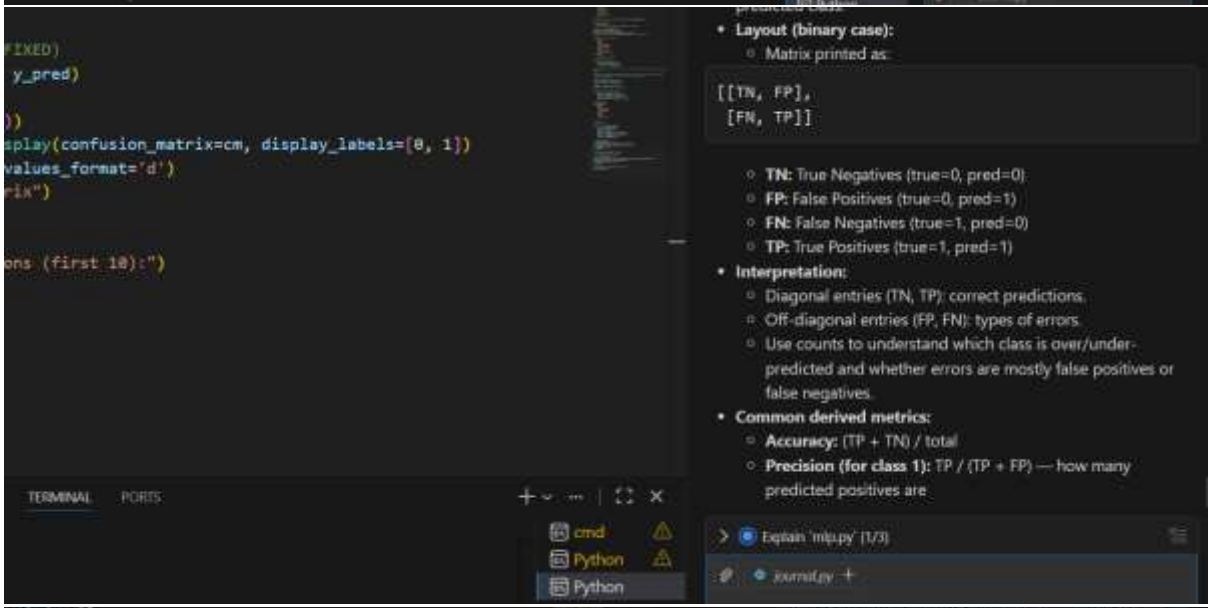
3. What is the role of softmax in multi-class output?

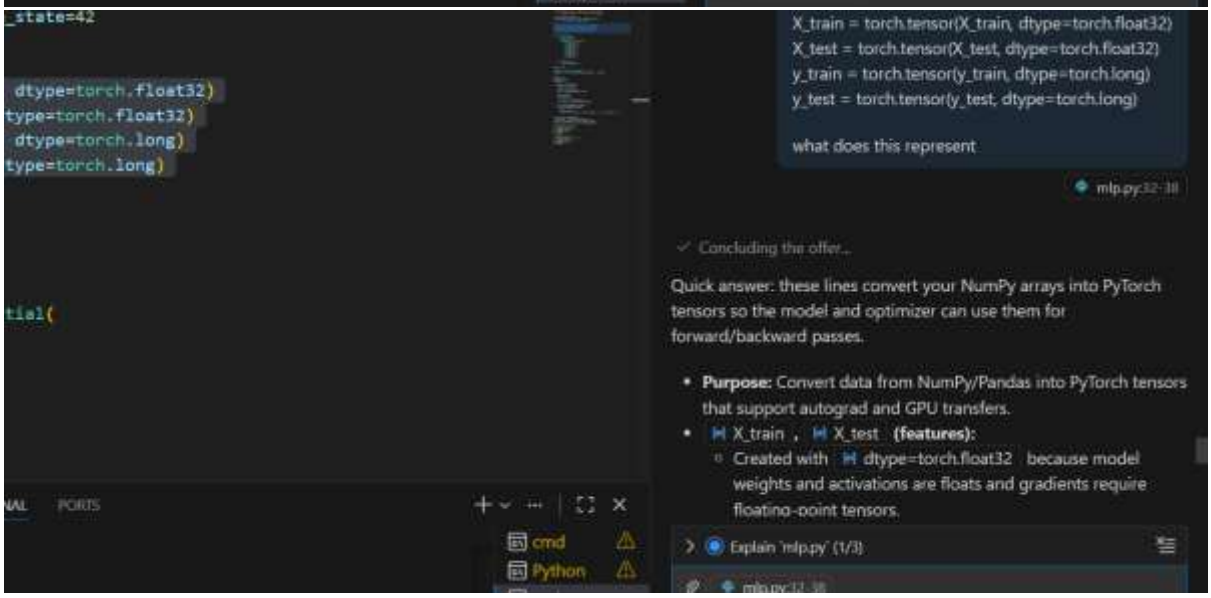
Softmax changes the final layer's scores into probabilities. It makes all class probabilities add up to 1, so the model can easily pick the class with the highest probability.

4. How does backprop automatically update weights compared to your manual sigmoid neuron?

In the manual neuron, you set the weights yourself, so it can’t really learn.
Backprop checks the error, calculates the gradient, and then updates every weight automatically to reduce the mistake.
So the model keeps improving on its own after every epoch.


Done — I added two short explanatory comments in journal.py
Above bias to explain it's chosen to demonstrate threshold sensitivity.
Above weights to note they're hand-selected and not learned.
Progress update: journal.py updated.
Next steps: want me to (pick one)
Add a bias-sweep plot and diagnostics (a distribution + accuracy vs bias), or
Replace the manual classifier with a learned logistic regression and show learned weights/bias, or
Leave it as-is.
TERMINAL
PORTS
cmd
Python


Layout (binary case):
Matrix printed as:
[[TN, FP],
[FN, TP]]
TN: True Negatives (true=0, pred=0)
FP: False Positives (true=0, pred=1)
FN: False Negatives (true=1, pred=0)
TP: True Positives (true=1, pred=1)
Interpretation:
Diagonal entries (TN, TP): correct predictions.
Off-diagonal entries (FP, FN): types of errors.
Use counts to understand which class is over/under-predicted and whether errors are mostly false positives or false negatives.
Common derived metrics:
Accuracy: (TP + TN) / total
Precision (for class 1): TP / (TP + FP) — how many predicted positives are
TERMINAL
PORTS
cmd
Python
Python
Explain 'mlp.py' (1/3)
journal.py


X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
y_test = torch.tensor(y_test, dtype=torch.long)
what does this represent
mlp.py:32-38
Concluding the offer...
Quick answer: these lines convert your NumPy arrays into PyTorch tensors so the model and optimizer can use them for forward/backward passes.
Purpose: Convert data from NumPy/Pandas into PyTorch tensors that support autograd and GPU transfers.
X_train , X_test (features):
Created with dtype=torch.float32 because model weights and activations are floats and gradients require floating-point tensors.
PORTS
cmd
Python
Explain 'mlp.py' (1/3)