LAB Assignment No. 5

Topic: Artificial Neural Network Model

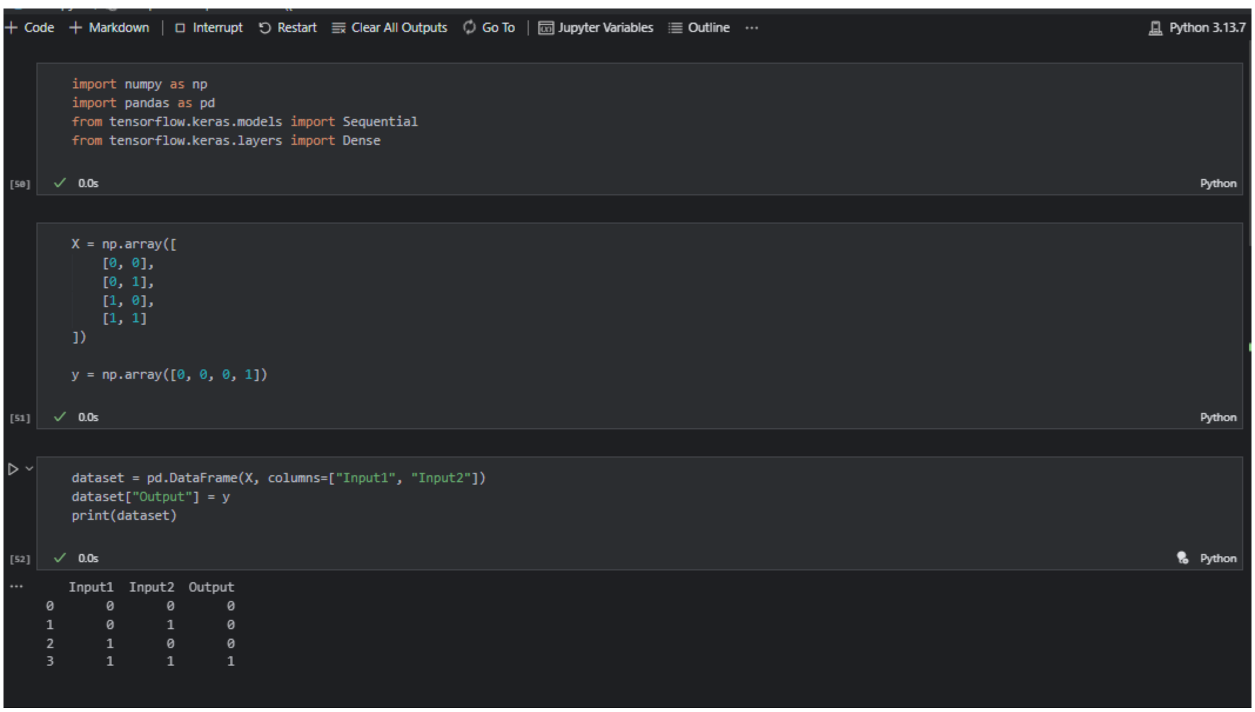
Question 1

Logic Gates with Neural Network. Implement a feed-forward neural network to learn the AND gate.

- Inputs: (0,0), (0,1), (1,0), (1,1)
- Output: 0, 0, 0, 1

  Tasks:

1. Create dataset using NumPy or pandas.
2. Build a neural network with one hidden layer using TensorFlow/Keras or PyTorch.
3. Train it and show accuracy.
4. Compare model predictions with actual outputs




```python
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```python
X = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])

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

```python
dataset = pd.DataFrame(X, columns=["Input1", "Input2"])
dataset["Output"] = y
print(dataset)
```

```
   Input1  Input2  Output
0       0       0       0
1       0       1       0
2       1       0       0
3       1       1       1
```

```
model = Sequential()
model.add(Dense(4, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

[53] ✓ 0.0s Python

```
c:\Users\h\AppData\Local\Programs\Python\Python313\Lib\site-packages\keras\src\layers\core\dense.py:95: UserWarning: Do not pass an `input_shape`/`input_dim` a
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

[54] ✓ 0.0s Python

```
model.fit(X, y, epochs=500, verbose=0)
```

[55] ✓ 55.0s Python

```
<keras.src.callbacks.history.History at 0x15e27566780>
```

```
loss, accuracy = model.evaluate(X, y, verbose=0)
print("Model Accuracy:", accuracy)
```

[56] ✓ 0.3s Python

```
Model Accuracy: 1.0
```

lab5.ipynb > comparison = pd.DataFrame({

+ Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ... Python 3.13.7


```
loss, accuracy = model.evaluate(X, y, verbose=0)
print("Model Accuracy:", accuracy)
```

[56] ✓ 0.3s Python

```
Model Accuracy: 1.0
```


```
predictions = model.predict(X)
```

[57] ✓ 0.1s Python

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 108ms/step
```

```
predicted_output = (predictions > 0.5).astype(int)
```

[58] ✓ 0.0s Python

```
comparison = pd.DataFrame({
    "Input1": X[:, 0],
    "Input2": X[:, 1],
    "Actual Output": y,
    "Predicted Output": predicted_output.flatten()
})

print(comparison)
```

[59] ✓ 0.0s Python

```
   Input1  Input2  Actual Output  Predicted Output
0       0       0              0                 0
1       0       1              0                 0
2       1       0              0                 0
3       1       1              1                 1
```

Question 2

Create a dataset $y = x^2$ + noise for x in range [-3,3]. Regression Task with Neural Network

Tasks:

1. Generate 100 samples.
2. Build a neural network to predict y from x.
3. Plot actual vs. predicted results.
4. Discuss how increasing hidden neurons changes results.

+ Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ··· Python 3.13.

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

[87] ✓ 0.0s Python

```python
np.random.seed(0)

# 100 samples in range [-3,3]
x = np.linspace(-3, 3, 100).reshape(-1, 1)
noise = np.random.randn(100,1) * 0.5
y = x**2 + noise
```

[88] ✓ 0.0s Python

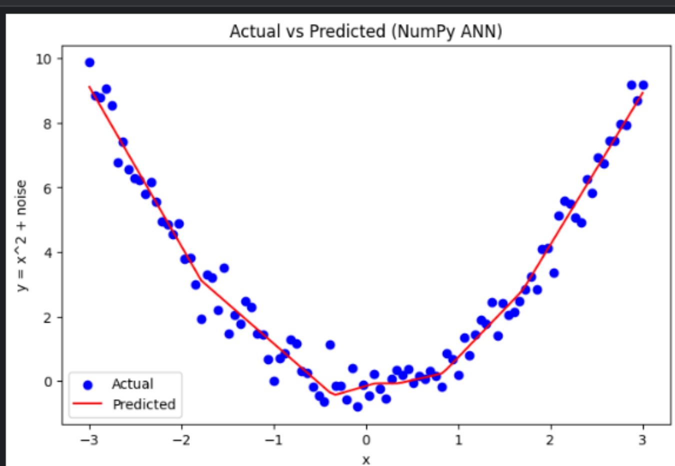
```python
# Network architecture
input_neurons = 1
hidden_neurons = 8
output_neurons = 1

# Initialize weights and biases
W1 = np.random.randn(input_neurons, hidden_neurons)
b1 = np.zeros((1, hidden_neurons))

W2 = np.random.randn(hidden_neurons, output_neurons)
b2 = np.zeros((1, output_neurons))

# Activation function
def relu(z):
    return np.maximum(0, z)

def relu_derivative(z):
    return (z > 0).astype(float)
```

[89]

```python
learning_rate = 0.01
epochs = 2000

for epoch in range(epochs):
    # Forward pass
    z1 = x @ W1 + b1
    a1 = relu(z1)
    z2 = a1 @ W2 + b2
    y_pred = z2  # Linear output for regression

    # Loss (MSE)
    loss = np.mean((y_pred - y)**2)

    # Backpropagation
    dloss = 2*(y_pred - y)/y.size
    dW2 = a1.T @ dloss
    db2 = np.sum(dloss, axis=0, keepdims=True)

    da1 = dloss @ W2.T
    dz1 = da1 * relu_derivative(z1)
    dW1 = x.T @ dz1
    db1 = np.sum(dz1, axis=0, keepdims=True)

    # Update weights
    W2 -= learning_rate * dW2
    b2 -= learning_rate * db2
    W1 -= learning_rate * dW1
    b1 -= learning_rate * db1

    if (epoch+1) % 500 == 0:
        print(f"Epoch {epoch+1}, Loss: {loss:.4f}")
```

```
Epoch 500, Loss: 0.3950
Epoch 1000, Loss: 0.2915
Epoch 1500, Loss: 0.2392
Epoch 2000, Loss: 0.2207
```

```python
plt.figure(figsize=(8,5))
plt.scatter(x, y, label="Actual", color='blue')
plt.plot(x, y_pred, label="Predicted", color='red')
plt.xlabel("x")
plt.ylabel("y = x^2 + noise")
plt.title("Actual vs Predicted (NumPy ANN)")
plt.legend()
plt.show()
```



Question 3:

Use the XOR gate and train networks with different activation functions (sigmoid, tanh, ReLU).

- Compare accuracy, loss, and convergence speed.

- Plot and discuss results.

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

```python
# Inputs
X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[0],[1],[1],[0]])
```

```python
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    s = sigmoid(x)
    return s * (1 - s)

def tanh(x):
    return np.tanh(x)

def tanh_derivative(x):
    return 1 - np.tanh(x)**2

def relu(x):
    return np.maximum(0, x)

def relu_derivative(x):
    return (x > 0).astype(float)
```

```python
def train_xor(activation, activation_derivative, hidden_neurons=4, lr=0.1, epochs=10000):
    # Initialize weights
    np.random.seed(0)
    W1 = np.random.randn(2, hidden_neurons)
    b1 = np.zeros((1, hidden_neurons))
    W2 = np.random.randn(hidden_neurons, 1)
    b2 = np.zeros((1,1))

    losses = []

    for epoch in range(epochs):
        # Forward pass
        z1 = X @ W1 + b1
        a1 = activation(z1)
        z2 = a1 @ W2 + b2
        y_pred = sigmoid(z2)  # Output layer always sigmoid

        # Loss
        loss = np.mean((y_pred - y)**2)
        losses.append(loss)

        # Backpropagation
        dloss = 2*(y_pred - y)/y.size
        dz2 = dloss * sigmoid_derivative(z2)
        dW2 = a1.T @ dz2
        db2 = np.sum(dz2, axis=0, keepdims=True)

        da1 = dz2 @ W2.T
        dz1 = da1 * activation_derivative(z1)
        dW1 = X.T @ dz1
        db1 = np.sum(dz1, axis=0, keepdims=True)

        # Update weights
        W2 -= lr * dW2
        b2 -= lr * db2
        W1 -= lr * dW1
        b1 -= lr * db1
```

```python
        # Update weights
        W2 -= lr * dW2
        b2 -= lr * db2
        W1 -= lr * dW1
        b1 -= lr * db1

    # Final predictions
    final_pred = (y_pred > 0.5).astype(int)
    accuracy = np.mean(final_pred == y)
    return losses, accuracy, final_pred
```

[113] ✓ 0.0s Python

```python
# Sigmoid
loss_sigmoid, acc_sigmoid, pred_sigmoid = train_xor(sigmoid, sigmoid_derivative)

# Tanh
loss_tanh, acc_tanh, pred_tanh = train_xor(tanh, tanh_derivative)

# ReLU
loss_relu, acc_relu, pred_relu = train_xor(relu, relu_derivative)
```

[114] ✓ 5.2s Python

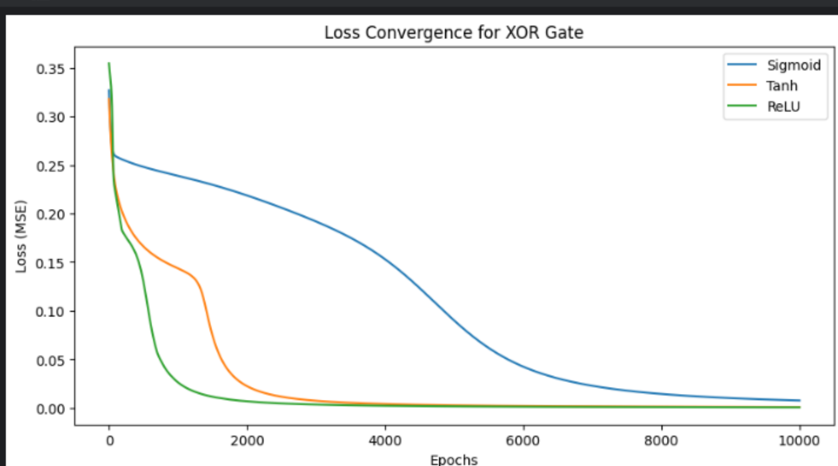
```python
print("Accuracy with Sigmoid:", acc_sigmoid)
print("Accuracy with Tanh   :", acc_tanh)
print("Accuracy with ReLU   :", acc_relu)
```

[115] ✓ 0.0s Python

```
Accuracy with Sigmoid: 1.0
Accuracy with Tanh   : 1.0
Accuracy with ReLU   : 1.0
```

```python
plt.figure(figsize=(10,5))
plt.plot(loss_sigmoid, label='Sigmoid')
plt.plot(loss_tanh, label='Tanh')
plt.plot(loss_relu, label='ReLU')
plt.xlabel("Epochs")
plt.ylabel("Loss (MSE)")
plt.title("Loss Convergence for XOR Gate")
plt.legend()
plt.show()
```

[116] ✓ 0.2s Python



Question 4

Binary Classification using Neural Network

Objective: Build a neural network to classify whether a tumor is malignant or benign using the **Breast Cancer dataset**.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```python
data = load_breast_cancer()
X = data.data
y = data.target.reshape(-1,1)  # 0 = malignant, 1 = benign

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```python
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    s = sigmoid(x)
    return s * (1 - s)
```

```python
input_neurons = X_train.shape[1]
hidden_neurons = 16
output_neurons = 1

np.random.seed(0)
W1 = np.random.randn(input_neurons, hidden_neurons) * 0.01
b1 = np.zeros((1, hidden_neurons))
W2 = np.random.randn(hidden_neurons, output_neurons) * 0.01
b2 = np.zeros((1, output_neurons))
```

```python
lr = 0.01
epochs = 5000
losses = []

for epoch in range(epochs):
    # Forward pass
    z1 = X_train @ W1 + b1
    a1 = sigmoid(z1)
    z2 = a1 @ W2 + b2
    y_pred = sigmoid(z2)

    # Loss (Binary Cross-Entropy)
    loss = -np.mean(y_train*np.log(y_pred+1e-8) + (1-y_train)*np.log(1-y_pred+1e-8))
    losses.append(loss)

    # Backpropagation
    dloss = y_pred - y_train
    dW2 = a1.T @ dloss
    db2 = np.sum(dloss, axis=0, keepdims=True)

    da1 = dloss @ W2.T
    dz1 = da1 * sigmoid_derivative(z1)
    dW1 = X_train.T @ dz1
```

```python
    # Loss (Binary Cross-Entropy)
    loss = -np.mean(y_train*np.log(y_pred+1e-8) + (1-y_train)*np.log(1-y_pred+1e-8))
    losses.append(loss)

    # Backpropagation
    dloss = y_pred - y_train
    dW2 = a1.T @ dloss
    db2 = np.sum(dloss, axis=0, keepdims=True)

    da1 = dloss @ W2.T
    dz1 = da1 * sigmoid_derivative(z1)
    dW1 = X_train.T @ dz1
    db1 = np.sum(dz1, axis=0, keepdims=True)

    # Update weights
    W2 -= lr * dW2
    b2 -= lr * db2
    W1 -= lr * dW1
    b1 -= lr * db1

    if (epoch+1) % 500 == 0:
        print(f"Epoch {epoch+1}, Loss: {loss:.4f}")
```

5] ✓ 4.3s

```
Epoch 500, Loss: 0.0229
Epoch 1000, Loss: 0.0062
Epoch 1500, Loss: 0.0023
Epoch 2000, Loss: 0.0012
Epoch 2500, Loss: 0.0008
Epoch 3000, Loss: 0.0006
Epoch 3500, Loss: 0.0005
Epoch 4000, Loss: 0.0004
Epoch 4500, Loss: 0.0003
Epoch 5000, Loss: 0.0003
```

+ Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ··· Python 3.13.

```
Epoch 3000, Loss: 0.0006
Epoch 3500, Loss: 0.0005
Epoch 4000, Loss: 0.0004
Epoch 4500, Loss: 0.0003
Epoch 5000, Loss: 0.0003
```

```python
train_pred = (y_pred > 0.5).astype(int)
train_accuracy = np.mean(train_pred == y_train)
print("Training Accuracy:", train_accuracy)
```

Success

[186] ✓ 0.0s Python

```
Training Accuracy: 1.0
```

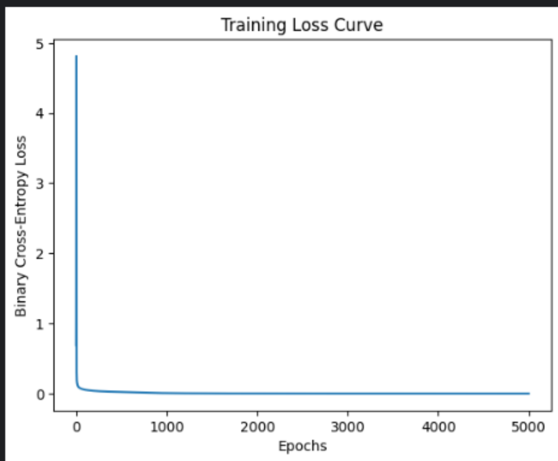
```python
# Forward pass on test set
a1_test = sigmoid(X_test @ W1 + b1)
y_test_pred = sigmoid(a1_test @ W2 + b2)
y_test_pred_binary = (y_test_pred > 0.5).astype(int)

test_accuracy = np.mean(y_test_pred_binary == y_test)
print("Test Accuracy:", test_accuracy)
```

[187] ✓ 0.0s Python

```
Test Accuracy: 0.9736842105263158
```

```python
plt.plot(losses)
plt.xlabel("Epochs")
plt.ylabel("Binary Cross-Entropy Loss")
plt.title("Training Loss Curve")
plt.show()
```

```python
plt.plot(losses)
plt.xlabel("Epochs")
plt.ylabel("Binary Cross-Entropy Loss")
plt.title("Training Loss Curve")
plt.show()
```



Question 5

Multi-Class Classification on Iris Dataset

Objective: Train a neural network to classify flower species (Setosa, Versicolor, Virginica).

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import matplotlib.pyplot as plt
```

```python
# CSV file path
data = pd.read_csv("iris_data.csv")  # Replace with your CSV file name

# Check first few rows
print(data.head())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

  species
0  setosa
1  setosa
2  setosa
3  setosa
4  setosa
```

```python
# Features (all columns except last)
X = data.iloc[:, :-1].values
```

lab5.ipynb > plt.plot(losses)

+ Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ··· Python 3.13.7

[237] ✓ 0.0s Python

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[238] ✓ 0.0s Python

```python
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    s = sigmoid(x)
    return s * (1 - s)

def softmax(x):
    exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))
    return exp_x / np.sum(exp_x, axis=1, keepdims=True)
```

[239] ✓ 0.0s Python

```python
input_neurons = X_train.shape[1]   # 4 features
hidden_neurons = 8
output_neurons = y_train.shape[1]  # 3 classes

np.random.seed(0)
W1 = np.random.randn(input_neurons, hidden_neurons) * 0.01
b1 = np.zeros((1, hidden_neurons))
W2 = np.random.randn(hidden_neurons, output_neurons) * 0.01
b2 = np.zeros((1, output_neurons))
```

[240]

Ln 11, Col 1 Spaces: 4 Spaces: 4 CRLF Cell 9 of 9

Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ··· Python 3.13.

```python
    # Backpropagation
    dloss = (y_pred - y_train)/y_train.shape[0]
    dW2 = a1.T @ dloss
    db2 = np.sum(dloss, axis=0, keepdims=True)

    da1 = dloss @ W2.T
    dz1 = da1 * sigmoid_derivative(z1)
    dW1 = X_train.T @ dz1
    db1 = np.sum(dz1, axis=0, keepdims=True)

    # Update weights
    W2 -= lr * dW2
    b2 -= lr * db2
    W1 -= lr * dW1
    b1 -= lr * db1

    if (epoch+1) % 500 == 0:
        print(f"Epoch {epoch+1}, Loss: {loss:.4f}")
```

41] ✓ 2.7s Python

```
Epoch 500, Loss: 0.6236
Epoch 1000, Loss: 0.3484
Epoch 1500, Loss: 0.2495
Epoch 2000, Loss: 0.1868
Epoch 2500, Loss: 0.1478
Epoch 3000, Loss: 0.1240
Epoch 3500, Loss: 0.1086
Epoch 4000, Loss: 0.0980
Epoch 4500, Loss: 0.0903
Epoch 5000, Loss: 0.0845
```

```python
# Training accuracy
train_pred = np.argmax(y_pred, axis=1)
y_train_labels = np.argmax(y_train, axis=1)
train_accuracy = np.mean(train_pred == y_train_labels)
print("Training Accuracy:", train_accuracy)

# Test accuracy
a1_test = sigmoid(X_test @ W1 + b1)
y_test_pred = softmax(a1_test @ W2 + b2)
test_pred_labels = np.argmax(y_test_pred, axis=1)
y_test_labels = np.argmax(y_test, axis=1)
test_accuracy = np.mean(test_pred_labels == y_test_labels)
print("Test Accuracy:", test_accuracy)
```

[242] ✓ 0.0s Python

```
Training Accuracy: 0.9583333333333334
Test Accuracy: 1.0
```

```python
plt.plot(losses)
plt.xlabel("Epochs")
plt.ylabel("Categorical Cross-Entropy Loss")
plt.title("Training Loss Curve - Iris Dataset")
plt.show()
```

[243] ✓ 0.3s Python


```python
plt.plot(losses)
plt.xlabel("Epochs")
plt.ylabel("Categorical Cross-Entropy Loss")
plt.title("Training Loss Curve - Iris Dataset")
plt.show()
```

[243] ✓ 0.3s Python

Question 6

Regression Problem (House Price Prediction)

Objective: Predict house prices using the **California Housing dataset**.

+ Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ··· Python 3.13

```python
losses = []

for e in range(epochs):
    # Forward
    a1 = sigmoid(X_train @ W1 + b1)
    y_pred = a1 @ W2 + b2

    # Loss
    loss = np.mean((y_train - y_pred)**2)
    losses.append(loss)

    # Backprop & update
    dW2 = a1.T @ (2*(y_pred - y_train)/y_train.shape[0])
    db2 = np.sum(2*(y_pred - y_train)/y_train.shape[0], axis=0, keepdims=True)
    dz1 = (2*(y_pred - y_train)/y_train.shape[0]) @ W2.T * a1*(1-a1)
    dW1 = X_train.T @ dz1
    db1 = np.sum(dz1, axis=0, keepdims=True)

    W2 -= lr*dW2; b2 -= lr*db2
    W1 -= lr*dW1; b1 -= lr*db1

    if (e+1) % 200 == 0:
        print(f"Epoch {e+1}, Loss: {loss:.4f}")
```

[282] ✓ 35.8s Python

```
Epoch 200, Loss: 0.9993
Epoch 400, Loss: 0.9968
Epoch 600, Loss: 0.9872
Epoch 800, Loss: 0.9511
Epoch 1000, Loss: 0.8464
Epoch 1200, Loss: 0.6709
Epoch 1400, Loss: 0.5312
Epoch 1600, Loss: 0.4710
Epoch 1800, Loss: 0.4483
Epoch 2000, Loss: 0.4356
```


```
Epoch 800, Loss: 0.9511
Epoch 1000, Loss: 0.8464
Epoch 1200, Loss: 0.6709
Epoch 1400, Loss: 0.5312
Epoch 1600, Loss: 0.4710
Epoch 1800, Loss: 0.4483
Epoch 2000, Loss: 0.4356
```


```python
a1_test = sigmoid(X_test @ W1 + b1)
y_test_pred = a1_test @ W2 + b2

y_test_pred_orig = scaler_y.inverse_transform(y_test_pred)
y_test_orig = scaler_y.inverse_transform(y_test)

print("R² Score:", r2_score(y_test_orig, y_test_pred_orig))
print("MSE:", mean_squared_error(y_test_orig, y_test_pred_orig))
```

[283] ✓ 0.0s Python

```
R² Score: 0.545118164526881
MSE: 0.5960812412426748
```

```python
plt.scatter(y_test_orig, y_test_pred_orig, alpha=0.5)
plt.plot([y_test_orig.min(), y_test_orig.max()], [y_test_orig.min(), y_test_orig.max()], 'r--')
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual vs Predicted House Prices")
plt.show()
```

[284] ✓ 0.7s Python

```python
plt.scatter(y_test_orig, y_test_pred_orig, alpha=0.5)
plt.plot([y_test_orig.min(), y_test_orig.max()], [y_test_orig.min(), y_test_orig.max()], 'r--')
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual vs Predicted House Prices")
plt.show()
```

✓ 0.7s

Question 7

Neural Network with Dropout Regularization

Objective: Prevent overfitting using **Dropout** layers on the **MNIST digit dataset**.

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
```

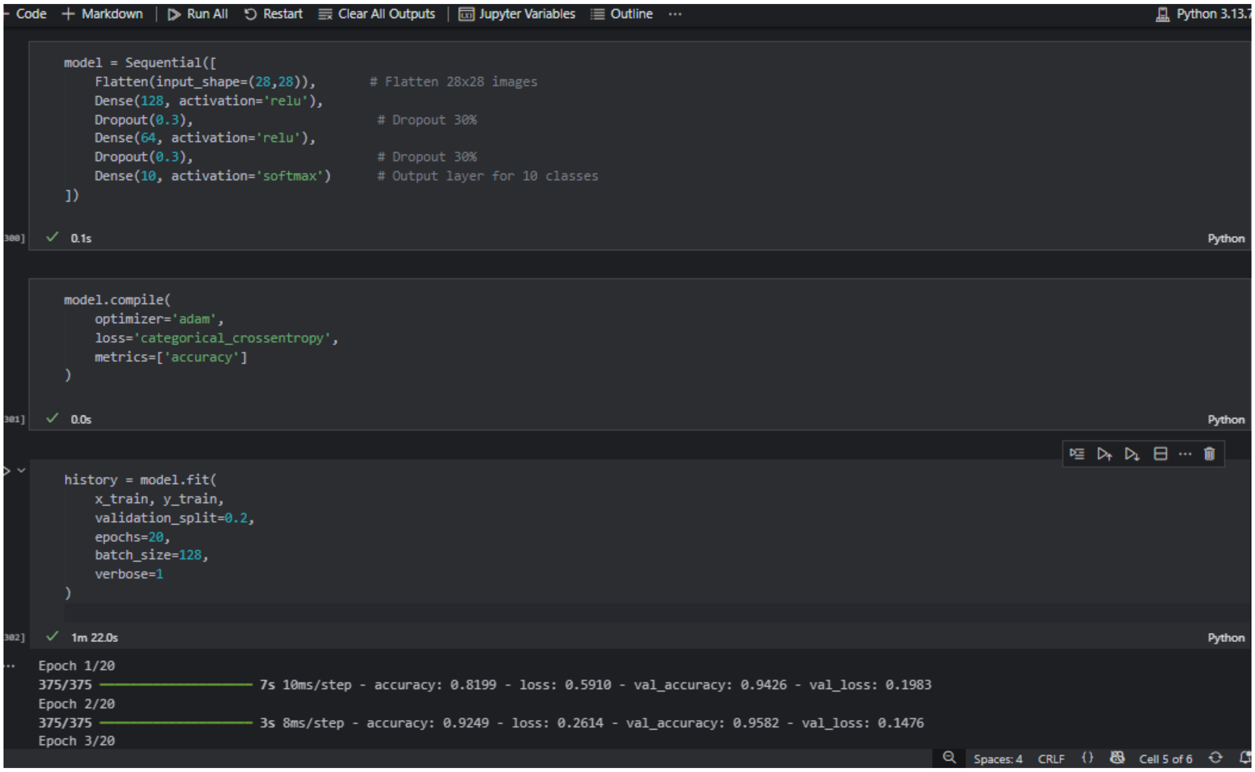
✓ 0.0s

```python
# Load MNIST
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize images
x_train, x_test = x_train/255.0, x_test/255.0

# One-hot encode labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

✓ 1.3s

```python
model = Sequential([
    Flatten(input_shape=(28,28)),      # Flatten 28x28 images
    Dense(128, activation='relu'),
    Dropout(0.3),                       # Dropout 30%
    Dense(64, activation='relu'),
    Dropout(0.3),                       # Dropout 30%
    Dense(10, activation='softmax')     # Output layer for 10 classes
])
```

✓ 0.1s

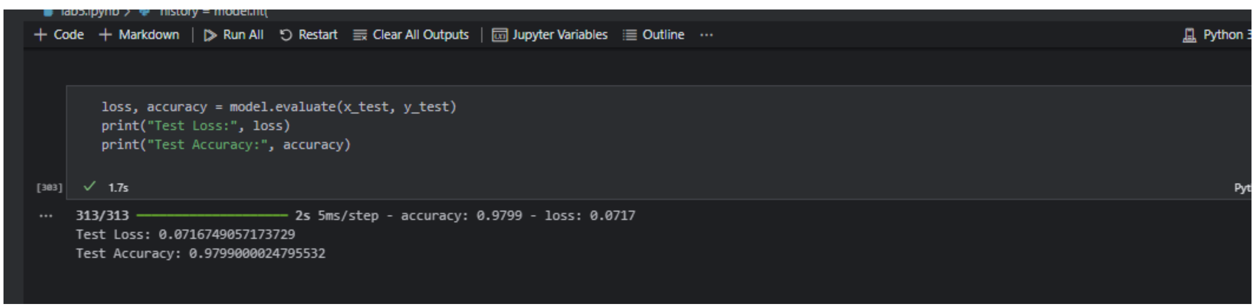



```python
model = Sequential([
    Flatten(input_shape=(28,28)),      # Flatten 28x28 images
    Dense(128, activation='relu'),
    Dropout(0.3),                       # Dropout 30%
    Dense(64, activation='relu'),
    Dropout(0.3),                       # Dropout 30%
    Dense(10, activation='softmax')     # Output layer for 10 classes
])
```

```python
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

```python
history = model.fit(
    x_train, y_train,
    validation_split=0.2,
    epochs=20,
    batch_size=128,
    verbose=1
)
```

```
Epoch 1/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 7s 10ms/step - accuracy: 0.8199 - loss: 0.5910 - val_accuracy: 0.9426 - val_loss: 0.1983
Epoch 2/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.9249 - loss: 0.2614 - val_accuracy: 0.9582 - val_loss: 0.1476
Epoch 3/20
```




```python
loss, accuracy = model.evaluate(x_test, y_test)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9799 - loss: 0.0717
Test Loss: 0.0716749057173729
Test Accuracy: 0.9799000024795532
```