```
# Lab : 04
'''
This code demonstrates a simple binary classification problem using a neural ,
where the goal is to predict whether a student will pass or fail based on the number of hours studied.
 '''


import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
'''
numpy:          Used for numerical operations and array manipulations.
tensorflow:     A deep learning framework.
Sequential:     A linear stack of layers in Keras.
Dense:          A fully connected layer in a neural network.
SGD:            Stochastic Gradient Descent optimizer.
'''


# Input: Hours studied
X = np.array([[1], [2], [3], [4], [5], [6], [7], [8], [9], [10]])

# Output: 0 = Fail, 1 = Pass
y = np.array([[0], [0], [0], [0], [0], [1], [1], [1], [1], [1]])

# Scale input to range [0, 1] for better convergence during training
X = X / 10


# A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.

# Define a Sequential model, which is a linear stack of layers
model = Sequential([
    # Hidden layer with 3 neurons and ReLU activation function
    Dense(3, activation='relu', input_shape=(1,)),
    # Output layer with 1 neuron and sigmoid activation function for binary classification
    Dense(1, activation='sigmoid')
])
'''
Sequential:   A model where layers are added sequentially.
Dense(3, activation='relu', input_shape=(1,)):     A hidden layer with 3 neurons and ReLU activation function. The input_shape=(1,) spec
Dense(1, activation='sigmoid'):      Output layer with 1 neuron and sigmoid activation function, which outputs a probability between 0 a
'''
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` arg
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# Compile the model with Stochastic Gradient Descent (SGD) optimizer, binary crossentropy loss, and accuracy metric
model.compile(optimizer=SGD(learning_rate=0.1), loss='binary_crossentropy', metrics=['accuracy'])

'''
optimizer=SGD(learning_rate=0.1):   Stochastic Gradient Descent optimizer with a learning rate of 0.1. The optimizer updates the model's
loss='binary_crossentropy':     Binary crossentropy loss function, suitable for binary classification problems.
metrics=['accuracy']:    Tracks the accuracy during training.
'''
```

```
'\nmodel.compile(optimizer=SGD(learning_rate=0.1), loss='binary_crossentropy', metrics=['accuracy']):\noptimizer=SGD(learning_rate=
0.1): Stochastic Gradient Descent optimizer with a learning rate of 0.1. The optimizer updates the model's weights to minimize the
loss.\nloss='binary_crossentropy': Binary crossentropy loss function, suitable for binary classification problems.\nmetrics=['accur
```

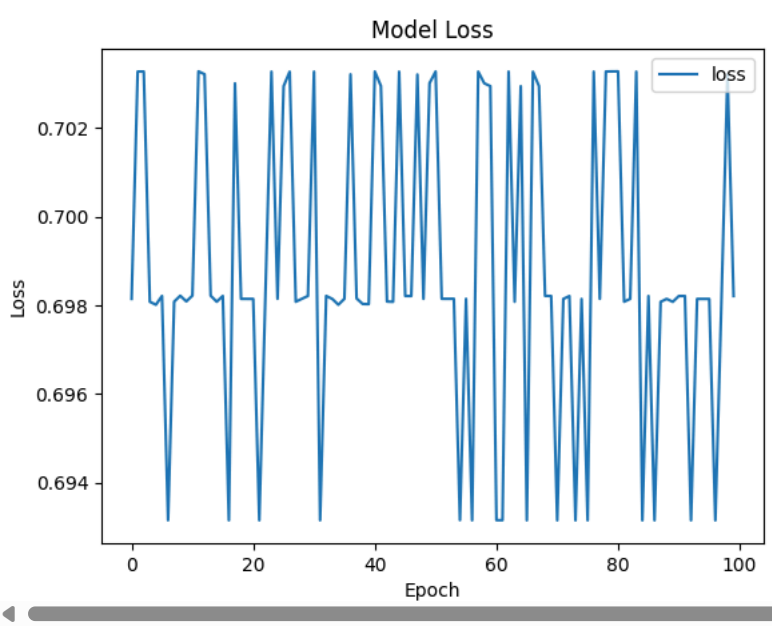
```
# Train the model for 100 epochs with a batch size of 2
history = model.fit(X, y, epochs=5, batch_size=2, verbose=1)
'''
X: Input data.
y: Target labels.
epochs=100: The model will be trained for 100 iterations over the entire dataset.
batch_size=2: The model will update its weights after every 2 samples.
verbose=1: Shows the progress bar and loss/accuracy metrics during training.
history: Stores the training history, including loss and accuracy over epochs.
'''
```

```
Epoch 1/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5417 - loss: 0.6973
Epoch 2/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5417 - loss: 0.6973
Epoch 3/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.3222 - loss: 0.6959
Epoch 4/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5208 - loss: 0.6948
Epoch 5/5
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.6042 - loss: 0.7010
'\nX: Input data.\ny: Target labels.\nepochs=100: The model will be trained for 100 iterations over the entire dataset.\nbatch_size
```

```
# Predict the pass probability for a new student who studied 4.5 hours
new_student = np.array([[2]]) / 10
prediction = model.predict(new_student)
print("Pass Probability:", prediction[0][0])

# Predict the pass probability for a new student who studied 8 hours
new_student = np.array([[8]]) / 10
prediction = model.predict(new_student)
print("Pass Probability:", prediction[0][0])

# Predict the pass probability for a new student who studied 5.5 hours
new_student = np.array([[5.5]]) / 10
prediction = model.predict(new_student)
print("Pass Probability:", prediction[0][0])

# Predict the pass probability for a new student who studied 7.5 hours
new_student = np.array([[7.5]]) / 10
prediction = model.predict(new_student)
print("Pass Probability:", prediction[0][0])
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
Pass Probability: 0.49974084
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
Pass Probability: 0.49974084
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
Pass Probability: 0.49974084
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
Pass Probability: 0.49974084
```

```
# Plot the training loss
import matplotlib.pyplot as plt
plt.plot(history.history['loss'], label='loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




Start coding or generate with AI.

```
import numpy as np
import tensorflow as tf
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD

# Input: Hours studied
X = np.array([[1], [2], [3], [4], [5], [6], [7], [8], [9], [10]])

# Output: 0 = Fail, 1 = Pass
y = np.array([[0], [0], [0], [0], [0], [1], [1], [1], [1], [1]])

# Scale input to range [0, 1] for better convergence during training
X = X / 10

# Define a Sequential model, which is a linear stack of layers
model = Sequential([
    # Hidden layer with 3 neurons and ReLU activation function
    Dense(3, activation='relu', input_shape=(1,)),
    # Output layer with 1 neuron and sigmoid activation function for binary classification
    Dense(1, activation='sigmoid')
])

# Compile the model with Stochastic Gradient Descent (SGD) optimizer, binary crossentropy loss, and accuracy metric
model.compile(optimizer=SGD(learning_rate=0.1), loss='binary_crossentropy', metrics=['accuracy'])

# Train the model for 100 epochs with a batch size of 2
history = model.fit(X, y, epochs=100, batch_size=2, verbose=1)

# Predict the pass probability for a new student who studied 4.5 hours
new_student = np.array([[4.5]]) / 10
prediction = model.predict(new_student)
print("Pass Probability:", prediction[0][0])

# Predict the pass probability for a new student who studied 8 hours
new_student = np.array([[8]]) / 10
prediction = model.predict(new_student)
print("Pass Probability:", prediction[0][0])

# Predict the pass probability for a new student who studied 5.5 hours
new_student = np.array([[5.5]]) / 10
prediction = model.predict(new_student)
print("Pass Probability:", prediction[0][0])

# Predict the pass probability for a new student who studied 7.5 hours
new_student = np.array([[7.5]]) / 10
prediction = model.predict(new_student)
print("Pass Probability:", prediction[0][0])

# Plot the training loss
import matplotlib.pyplot as plt
plt.plot(history.history['loss'], label='loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

## Explanation of the Code:

### 1. **Imports:**

- `numpy`: Used for numerical operations and array manipulations.
- `tensorflow`: A deep learning framework.
- `Sequential`: A linear stack of layers in Keras.
- `Dense`: A fully connected layer in a neural network.
- `SGD`: Stochastic Gradient Descent optimizer.

### 2. **Input and Output Data:**

- `X`: Input data representing hours studied, scaled to the range [0, 1].

- `y`: Output labels (0 for Fail, 1 for Pass).

3. **Model Definition:**

- `Sequential`: A model where layers are added sequentially.
- `Dense(3, activation='relu', input_shape=(1,))`: A hidden layer with 3 neurons and ReLU activation function. The `input_shape=(1,)` specifies that the input is a single feature (hours studied).
- `Dense(1, activation='sigmoid')`: Output layer with 1 neuron and sigmoid activation function, which outputs a probability between 0 and 1.

4. **Model Compilation:**

- `model.compile(optimizer=SGD(learning_rate=0.1), loss='binary_crossentropy', metrics=['accuracy'])`:
  - `optimizer=SGD(learning_rate=0.1)`: Stochastic Gradient Descent optimizer with a learning rate of 0.1. The optimizer updates the model's weights to minimize the loss.
  - `loss='binary_crossentropy'`: Binary crossentropy loss function, suitable for binary classification problems.
  - `metrics=['accuracy']`: Tracks the accuracy during training.

5. **Model Training:**

- `history = model.fit(X, y, epochs=100, batch_size=2, verbose=1)`:
  - `X`: Input data.
  - `y`: Target labels.
  - `epochs=100`: The model will be trained for 100 iterations over the entire dataset.
  - `batch_size=2`: The model will update its weights after every 2 samples.
  - `verbose=1`: Shows the progress bar and loss/accuracy metrics during training.
  - `history`: Stores the training history, including loss and accuracy over epochs.

6. **Predictions:**

- `model.predict(new_student)`: Predicts the pass probability for a new student based on the hours studied.
- The input is normalized by dividing by 10 to match the training data scale.

7. **Plotting Training Loss:**

- `plt.plot(history.history['loss'], label='loss')`: Plots the training loss over epochs.
- `plt.title('Model Loss')`: Sets the title of the plot.
- `plt.xlabel('Epoch')`: Labels the x-axis as "Epoch".
- `plt.ylabel('Loss')`: Labels the y-axis as "Loss".
- `plt.legend()`: Adds a legend to the plot.
- `plt.show()`: Displays the plot.

## Concepts:

**Optimizers:**

- **SGD (Stochastic Gradient Descent):** An optimization algorithm that updates the model's weights to minimize the loss function. It uses a fixed learning rate (0.1 in this case) to control the step size during weight updates.
- **Learning Rate:** A hyperparameter that determines the size of the steps taken to reach the minimum loss. A smaller learning rate leads to slower convergence but more precise results, while a larger learning rate may lead to faster convergence but risk overshooting the minimum.

**Sequential Model:**

- A linear stack of layers where each layer has exactly one input tensor and one output tensor. It is the simplest type of model in Keras, suitable for most feedforward neural networks.

**Layers:**

- **Dense Layer:** A fully connected layer where each neuron receives input from all the neurons in the previous layer. The number of neurons and activation function are specified.
- **ReLU Activation:** Rectified Linear Unit (ReLU) activation function, defined as `f(x) = max(0, x)`. It introduces non-linearity and helps the model learn complex patterns.
- **Sigmoid Activation:** Outputs a value between 0 and 1, making it suitable for binary classification problems.

**Model Compilation:**

- **Loss Function (`binary_crossentropy`):** Measures the difference between the predicted and actual labels. For binary classification, binary crossentropy is used.
- **Metrics (`accuracy`):** Tracks the accuracy of the model during training, which is the percentage of correct predictions.

**Model Training:**

- **Epochs:** One epoch means the model has seen the entire dataset once. Training for 100 epochs means the model will iterate over the dataset 100 times.
- **Batch Size:** The number of samples processed before the model updates its weights. A smaller batch size (e.g., 2) leads to more frequent updates but may result in noisier gradients.
- **Verbose:** Controls the amount of information displayed during training. `verbose=1` shows a progress bar and metrics.

This code demonstrates a simple binary classification problem using a neural network, where the goal is to predict whether a student will pass or fail based on the number of hours studied.

## Part 2

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

# Step 1: Load the dataset
df = pd.read_csv('Churn_Modelling.csv')

# Step 2: Drop unnecessary columns
df.drop(columns=['Surname', 'RowNumber', 'CustomerId'], inplace=True)

# Step 3: Separate the target variable (y) and features (X)
y = df['Exited'].values
X = df.drop(columns=['Exited'])

# Step 4: One-Hot Encoding for categorical variables
categorical_cols = ['Geography', 'Gender']
encoder = OneHotEncoder(drop='first')
X_encoded = encoder.fit_transform(X[categorical_cols])
encoded_feature_names = encoder.get_feature_names_out(categorical_cols)
X_encoded_df = pd.DataFrame(X_encoded.toarray(), columns=encoded_feature_names)
X = X.drop(columns=categorical_cols)
X = pd.concat([X, X_encoded_df], axis=1)

# Step 5: Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 6: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Step 7: Initialize the model
model = Sequential()

# Step 8: Add layers to the model
model.add(Dense(128, activation="relu", input_dim=11))  # Hidden layer with 128 neurons and ReLU activation
model.add(Dense(32, activation="sigmoid"))  # Hidden layer with 32 neurons and sigmoid activation
model.add(Dense(1, activation="sigmoid"))  # Output layer with 1 neuron and sigmoid activation for binary classification

# Step 9: Compile the model
model.compile(optimizer='Adam', loss='binary_crossentropy', metrics=['accuracy'])

# Step 10: Train the model
history = model.fit(X_train, y_train, epochs=50)

# Step 11: Evaluate the model and print accuracy
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")
```

```
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8687 - loss: 0.3118
Epoch 32/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8681 - loss: 0.3062
Epoch 33/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8732 - loss: 0.3096
Epoch 34/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8760 - loss: 0.3048
Epoch 35/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8804 - loss: 0.2959
Epoch 36/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8659 - loss: 0.3178
Epoch 37/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8806 - loss: 0.2944
Epoch 38/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8740 - loss: 0.3008
Epoch 39/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8780 - loss: 0.3059
Epoch 40/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8808 - loss: 0.2965
Epoch 41/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8857 - loss: 0.2797
Epoch 42/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8793 - loss: 0.2978
Epoch 43/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8759 - loss: 0.3017
Epoch 44/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8800 - loss: 0.2868
Epoch 45/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8774 - loss: 0.2941
Epoch 46/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8790 - loss: 0.2956
Epoch 47/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8780 - loss: 0.2880
Epoch 48/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8830 - loss: 0.2896
Epoch 49/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8786 - loss: 0.2953
Epoch 50/50
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8850 - loss: 0.2885
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8597 - loss: 0.3407
```

```
# Objective:

# The primary objective of this project is to develop and evaluate a neural network model for binary classification.
# The first part demonstrates a simple model predicting student pass/fail based on study hours. The second part tackles a more complex pr
# The goal is to accurately predict customer churn (whether a customer will leave a bank). This involves data preprocessing, model buildi


# Conclusion:

# The first part successfully demonstrates a basic neural network for binary classification.
# The model's ability to predict pass/fail probability based on study hours showcases the fundamental principles of neural network implem

# The second part shows the model successfully trained on the Churn_Modelling dataset and achieved a test accuracy.
# The preprocessing steps, including one-hot encoding for categorical variables and feature scaling, were crucial in preparing the data f
# The model architecture, incorporating hidden layers with appropriate activation functions, played a key role in learning complex relati
```