Examine the Breast Cancer Wisconsin (Diagnostic) Dataset to understand the number of samples, features, and classes, and classify it using an Artificial Neural Network (ANN). Implement 10-fold cross-validation to evaluate the model's performance, and report the test accuracy, precision, recall, and F1 score. The dataset is accessible from python as below sklearn.datasets import load_breast_cancer data = load_breast_cancer() # features and labels X, y = data.data, data.target

```
# Re-import necessary libraries as previous context is lost
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import numpy as np

# Load the dataset
data = load_breast_cancer()
X, y = data.data, data.target

# Examine the dataset
n_samples = X.shape[0]
n_features = X.shape[1]
n_classes = len(np.unique(y))

# Output the dataset information
n_samples, n_features, n_classes

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Define the ANN model
model = MLPClassifier(hidden_layer_sizes=(30, 30, 30), max_iter=500, random_state=42)

# Perform 10-fold cross-validation and predict
y_pred = cross_val_predict(model, X_scaled, y, cv=10)

# Calculate the performance metrics
accuracy = accuracy_score(y, y_pred)
precision = precision_score(y, y_pred)
recall = recall_score(y, y_pred)
f1 = f1_score(y, y_pred)

accuracy, precision, recall, f1
```

```
(0.9806678383128296, 0.9805555555555555, 0.988795518207283, 0.9846582984658299)
```

Develop a neural network-based regression model to predict house prices in California using the California Housing dataset. Aim at maximizing the accuracy in predicting house prices.

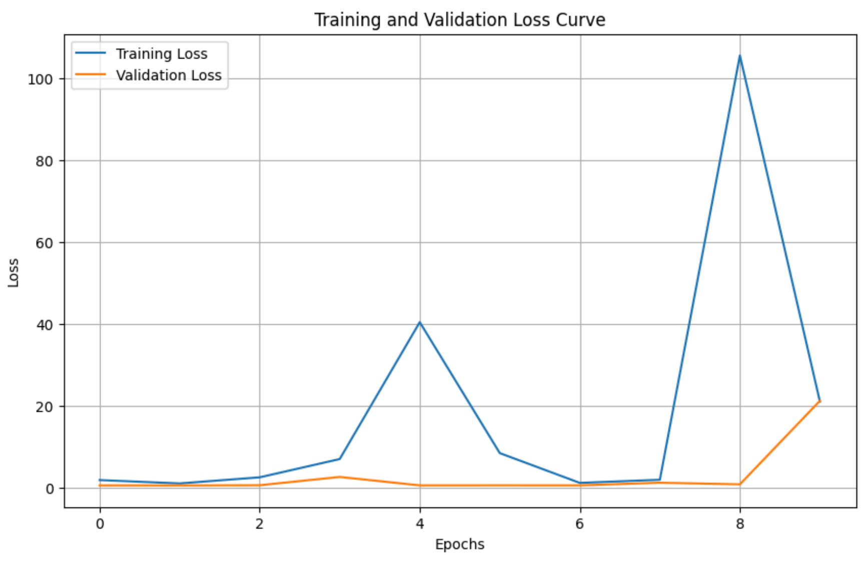
(a) Set the neural network parameters of your choice and specify them

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load the California Housing dataset
housing_data = fetch_california_housing()
X, y = housing_data.data, housing_data.target

# Split the data into training and testing sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)

# Build the neural network model
model = Sequential([
    Dense(100, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    Dense(100, activation='relu'),
    Dense(1)
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error', metrics=['accuracy'])

# Train the model and keep track of training and validation accuracy
history = model.fit(X_train_scaled, y_train, validation_data=(X_val_scaled, y_val), epochs=10, batch_size=32, verbose=1)
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the leg
Epoch 1/10
516/516 [==============================] - 3s 4ms/step - loss: 1.8735 - accuracy: 0.0029 - val_loss: 0.5763 - val_accura
Epoch 2/10
516/516 [==============================] - 2s 4ms/step - loss: 1.0533 - accuracy: 0.0030 - val_loss: 0.5605 - val_accura
Epoch 3/10
516/516 [==============================] - 2s 4ms/step - loss: 2.5450 - accuracy: 0.0030 - val_loss: 0.6120 - val_accura
Epoch 4/10
516/516 [==============================] - 2s 4ms/step - loss: 7.0052 - accuracy: 0.0030 - val_loss: 2.6387 - val_accura
Epoch 5/10
516/516 [==============================] - 2s 4ms/step - loss: 40.4184 - accuracy: 0.0029 - val_loss: 0.5798 - val_accur
Epoch 6/10
516/516 [==============================] - 2s 4ms/step - loss: 8.4557 - accuracy: 0.0031 - val_loss: 0.5878 - val_accura
Epoch 7/10
516/516 [==============================] - 2s 4ms/step - loss: 1.1834 - accuracy: 0.0030 - val_loss: 0.5801 - val_accura
Epoch 8/10
516/516 [==============================] - 2s 4ms/step - loss: 1.9499 - accuracy: 0.0028 - val_loss: 1.2274 - val_accura
Epoch 9/10
516/516 [==============================] - 2s 4ms/step - loss: 105.5158 - accuracy: 0.0031 - val_loss: 0.8333 - val_accu
Epoch 10/10
516/516 [==============================] - 2s 4ms/step - loss: 21.0307 - accuracy: 0.0027 - val_loss: 21.2295 - val_accu
```

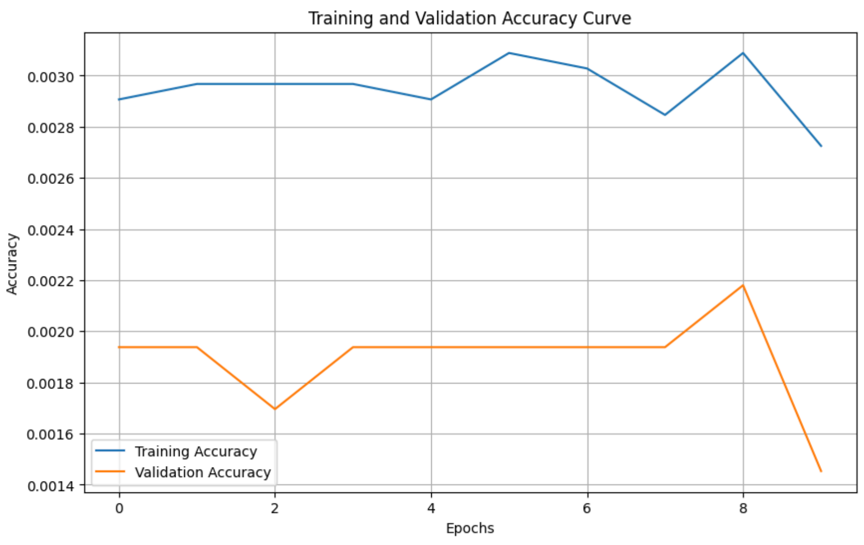
(b) Plot the graph of validation and training losses against the number of epochs

```
# Plot the training and validation loss
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss Curve')
plt.legend()
plt.grid(True)
plt.show()
```



(c) Plot the graph of accuracy against the number of epochs and return the best accuracy score.

```
# Plot the training and validation accuracy
plt.figure(figsize=(10, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy Curve')
plt.legend()
plt.grid(True)
plt.show()

# Return the best accuracy score
best_accuracy = max(history.history['val_accuracy'])
best_accuracy
```



```
0.0021802326664328575
```

Q3. You are tasked with developing a neural network to classify images from the CIFAR-10 dataset. The dataset consists of 60,000 32x32 color images in 10 different classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck), with 6,000 images per class. Compare how different number hidden layers affect the model's performance. Plot a graph of F1score against number of hidden layers.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import f1_score
import numpy as np
import matplotlib.pyplot as plt

# Load and preprocess the CIFAR-10 dataset
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.cifar10.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0
y_train, y_test = to_categorical(y_train), to_categorical(y_test)

# Function to create a model with a given number of hidden layers
def create_model(hidden_layers):
    model = Sequential()
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    for _ in range(hidden_layers):
        model.add(Dense(64, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Train and evaluate models with different numbers of hidden layers
hidden_layers_list = [1, 2]
f1_scores = []

for hidden_layers in hidden_layers_list:
    model = create_model(hidden_layers)
    model.fit(X_train, y_train, epochs=1, batch_size=64, verbose=1, validation_split=0.5)
    y_pred = model.predict(X_test)
    y_pred_classes = np.argmax(y_pred, axis=1)
    y_true = np.argmax(y_test, axis=1)
    f1 = f1_score(y_true, y_pred_classes, average='macro')
    f1_scores.append(f1)
    print(f'Number of hidden layers: {hidden_layers}, F1 Score: {f1}')

# Plot F1 score against number of hidden layers
plt.figure(figsize=(10, 6))
plt.plot(hidden_layers_list, f1_scores, marker='o')
plt.xlabel('Number of Hidden Layers')
plt.ylabel('F1 Score')
plt.title('F1 Score vs Number of Hidden Layers')
plt.grid(True)
plt.show()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 345s 2us/step
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the leg
2024-07-09 09:36:36.325136: E tensorflow/core/grappler/optimizers/meta_optimizer.cc:961] model_pruner failed: INVALID_AR
391/391 [==============================] - 12s 22ms/step - loss: 1.7954 - accuracy: 0.3552 - val_loss: 1.5556 - val_accu
313/313 [==============================] - 1s 2ms/step
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the leg
Number of hidden layers: 1, F1 Score: 0.4545358064367965
```

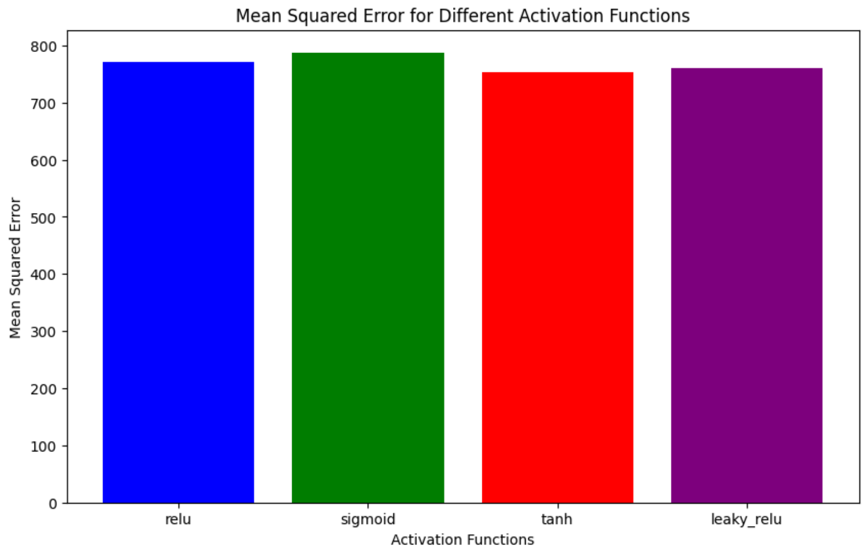
Q4. You are working on a project to predict isbat university student grades based on various features such as hours studied, class attendance, participation, and past grades. You want to compare how different activation functions in the hidden layers affect the model's performance. You will randomly generate the dataset, as

X = np.random.rand(1000, 5) # 1000 samples, 5 features (e.g., hours studied, attendance) y = np.random.rand(1000) * 100 # Grades in range [0, 100]. consider the following activation functions: relu, sigmoid,tanh, and leaky_relu, (a) Plot a histogram of accuracy against the activation functions

(b) Comment on the histogram

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LeakyReLU
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Generate a synthetic dataset
np.random.seed(42)
X = np.random.rand(1000, 5)  # 1000 samples, 5 features (e.g., hours studied, attendance)
y = np.random.rand(1000) * 100  # Grades in range [0, 100]

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

# Function to create a model with a given activation function
def create_model(activation):
    model = Sequential()
    model.add(Dense(64, input_shape=(X_train.shape[1],), activation=activation if activation != 'leaky_relu' else None))
    if activation == 'leaky_relu':
        model.add(LeakyReLU(alpha=0.1))
    model.add(Dense(64, activation=activation if activation != 'leaky_relu' else None))
    if activation == 'leaky_relu':
        model.add(LeakyReLU(alpha=0.1))
    model.add(Dense(1, activation='linear'))
    model.compile(optimizer=Adam(), loss='mean_squared_error')
    return model

# Train and evaluate models with different activation functions
activation_functions = ['relu', 'sigmoid', 'tanh', 'leaky_relu']
mse_scores = []

for activation in activation_functions:
    model = create_model(activation)
    model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    mse_scores.append(mse)
    print(f'Activation function: {activation}, Mean Squared Error: {mse}')

# Plot the histogram of MSE against activation functions
plt.figure(figsize=(10, 6))
plt.bar(activation_functions, mse_scores, color=['blue', 'green', 'red', 'purple'])
plt.xlabel('Activation Functions')
plt.ylabel('Mean Squared Error')
plt.title('Mean Squared Error for Different Activation Functions')
plt.show()
```

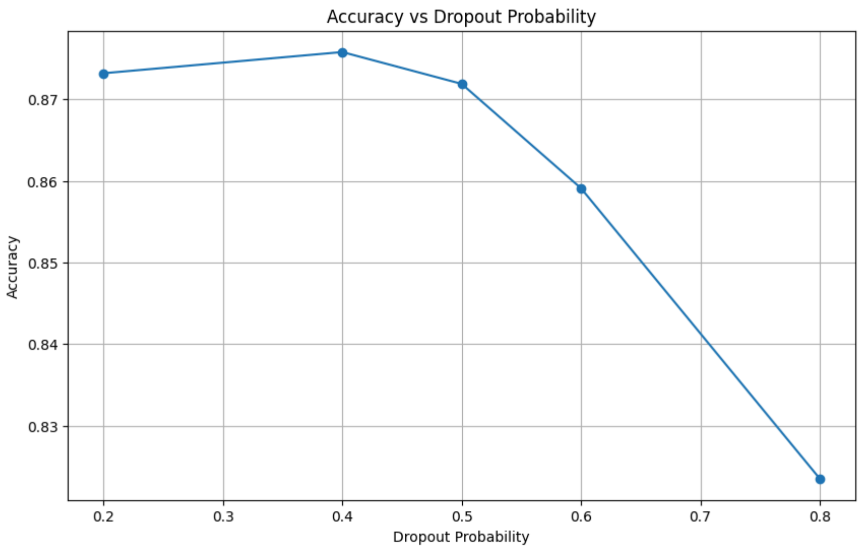
```
2024-07-09 09:42:18.386615: I metal_plugin/src/device/metal_device.cc:1154] M
2024-07-09 09:42:18.386643: I metal_plugin/src/device/metal_device.cc:296] sy
2024-07-09 09:42:18.386651: I metal_plugin/src/device/metal_device.cc:313] ma
2024-07-09 09:42:18.386686: I tensorflow/core/common_runtime/pluggable_device
2024-07-09 09:42:18.386703: I tensorflow/core/common_runtime/pluggable_device
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` ru
2024-07-09 09:42:19.022515: I tensorflow/core/grappler/optimizers/custom_grap
2024-07-09 09:42:19.044342: E tensorflow/core/grappler/optimizers/meta_optimi
7/7 [==============================] - 0s 4ms/step
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` ru
Activation function: relu, Mean Squared Error: 770.7223369282588
7/7 [==============================] - 0s 3ms/step
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` ru
Activation function: sigmoid, Mean Squared Error: 788.0516671478408
7/7 [==============================] - 0s 3ms/step
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` ru
Activation function: tanh, Mean Squared Error: 753.8879441067303
7/7 [==============================] - 0s 2ms/step
Activation function: leaky_relu, Mean Squared Error: 759.8878261318519
```



Q5. You are tasked with developing a neural network to classify images from the Fashion MNIST dataset. Each image is 28x28 pixels. The goal is to evaluate the effect of using different dropout probabilities on the performance of the neural network. Consider dropout probabilities as (0.2, 0.4, 0.5, 0.6, and 0.8). Plot a graph of accuracy against dropout probabilities and comment.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout, Conv2D, MaxPooling2D
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Load and preprocess the Fashion MNIST dataset
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0  # Normalize the data

# Reshape the data to add a channel dimension
X_train = X_train.reshape(-1, 28, 28, 1)
X_test = X_test.reshape(-1, 28, 28, 1)

# Convert labels to categorical format
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Function to create a model with a given dropout probability
def create_model(dropout_prob):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
        MaxPooling2D((2, 2)),
        Dropout(dropout_prob),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(dropout_prob),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Train and evaluate models with different dropout probabilities
dropout_probs = [0.2, 0.4, 0.5, 0.6, 0.8]
accuracies = []

for dropout_prob in dropout_probs:
    model = create_model(dropout_prob)
    history = model.fit(X_train, y_train, epochs=1, batch_size=64, validation_data=(X_test, y_test), verbose=1)
    accuracy = model.evaluate(X_test, y_test, verbose=0)[1]
    accuracies.append(accuracy)
    print(f'Dropout Probability: {dropout_prob}, Accuracy: {accuracy}')

# Plot the accuracy against dropout probabilities
plt.figure(figsize=(10, 6))
plt.plot(dropout_probs, accuracies, marker='o')
plt.xlabel('Dropout Probability')
plt.ylabel('Accuracy')
plt.title('Accuracy vs Dropout Probability')
plt.grid(True)
plt.show()
```

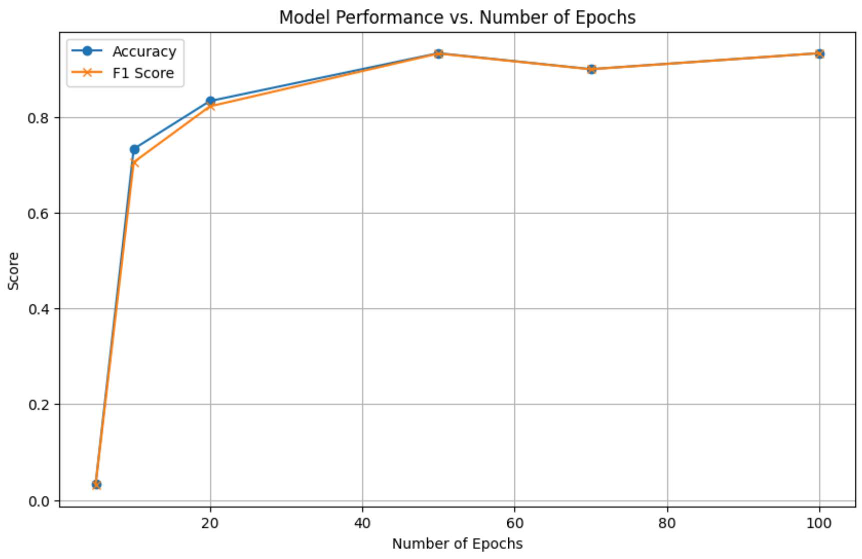
```
2024-07-09 09:51:25.199652: I metal_plugin/src/device/metal_device.cc:1154] Me
2024-07-09 09:51:25.199680: I metal_plugin/src/device/metal_device.cc:296] sy
2024-07-09 09:51:25.199686: I metal_plugin/src/device/metal_device.cc:313] ma
2024-07-09 09:51:25.199718: I tensorflow/core/common_runtime/pluggable_device,
2024-07-09 09:51:25.199733: I tensorflow/core/common_runtime/pluggable_device,
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` ru
2024-07-09 09:51:26.092274: I tensorflow/core/grappler/optimizers/custom_grap
2024-07-09 09:51:26.121281: E tensorflow/core/grappler/optimizers/meta_optimi
938/938 [==============================] - 12s 12ms/step - loss: 0.4577 - acc
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` ru
Dropout Probability: 0.2, Accuracy: 0.873199999332428
938/938 [==============================] - 12s 12ms/step - loss: 0.4964 - acc
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` ru
Dropout Probability: 0.4, Accuracy: 0.8758000135421753
938/938 [==============================] - 13s 13ms/step - loss: 0.5307 - acc
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` ru
Dropout Probability: 0.5, Accuracy: 0.8719000220298767
938/938 [==============================] - 12s 12ms/step - loss: 0.5857 - acc
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` ru
Dropout Probability: 0.6, Accuracy: 0.8590999841690063
938/938 [==============================] - 13s 13ms/step - loss: 0.8414 - acc
Dropout Probability: 0.8, Accuracy: 0.8234999775886536
```



Q6. Developing a simple neural network to classify the Iris dataset. The Iris dataset consists of 150 samples from three different species of Iris flowers (setosa, versicolor, and virginica). Each sample has four features: sepal length, sepal width, petal length, and petal width. Report the accuracy and f1 for for different number of training epochs. Consider the following values for epochs (5, 10, 20, 50, 70, 100). Plot a graph of accuracy against number of epochs. Comment on your graph.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score
import matplotlib.pyplot as plt

# Load and preprocess the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Convert labels to categorical format
y_train = tf.keras.utils.to_categorical(y_train, 3)
y_test = tf.keras.utils.to_categorical(y_test, 3)

# Function to create the model
def create_model():
    model = Sequential([
        Dense(10, activation='relu', input_shape=(X_train.shape[1],)),
        Dense(10, activation='relu'),
        Dense(3, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Training epochs to evaluate
epochs_list = [5, 10, 20, 50, 70, 100]
accuracies = []
f1_scores = []

# Train and evaluate models with different numbers of training epochs
for epochs in epochs_list:
    model = create_model()
    model.fit(X_train, y_train, epochs=epochs, batch_size=16, verbose=0)
    y_pred = model.predict(X_test)
    y_pred_classes = np.argmax(y_pred, axis=1)
    y_test_classes = np.argmax(y_test, axis=1)
    accuracy = accuracy_score(y_test_classes, y_pred_classes)
    f1 = f1_score(y_test_classes, y_pred_classes, average='macro')
    accuracies.append(accuracy)
    f1_scores.append(f1)
    print(f'Epochs: {epochs}, Accuracy: {accuracy:.4f}, F1 Score: {f1:.4f}')

# Plot the accuracy against the number of epochs
plt.figure(figsize=(10, 6))
plt.plot(epochs_list, accuracies, marker='o', label='Accuracy')
plt.plot(epochs_list, f1_scores, marker='x', label='F1 Score')
plt.xlabel('Number of Epochs')
plt.ylabel('Score')
plt.title('Model Performance vs. Number of Epochs')
plt.legend()
plt.grid(True)
plt.show()
```

```
1/1 [==============================] - 0s 61ms/step
Epochs: 5, Accuracy: 0.0333, F1 Score: 0.0317
1/1 [==============================] - 0s 35ms/step
Epochs: 10, Accuracy: 0.7333, F1 Score: 0.7056
1/1 [==============================] - 0s 36ms/step
Epochs: 20, Accuracy: 0.8333, F1 Score: 0.8222
1/1 [==============================] - 0s 37ms/step
Epochs: 50, Accuracy: 0.9333, F1 Score: 0.9327
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_
1/1 [==============================] - 0s 36ms/step
Epochs: 70, Accuracy: 0.9000, F1 Score: 0.8997
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_
1/1 [==============================] - 0s 36ms/step
Epochs: 100, Accuracy: 0.9333, F1 Score: 0.9333
```



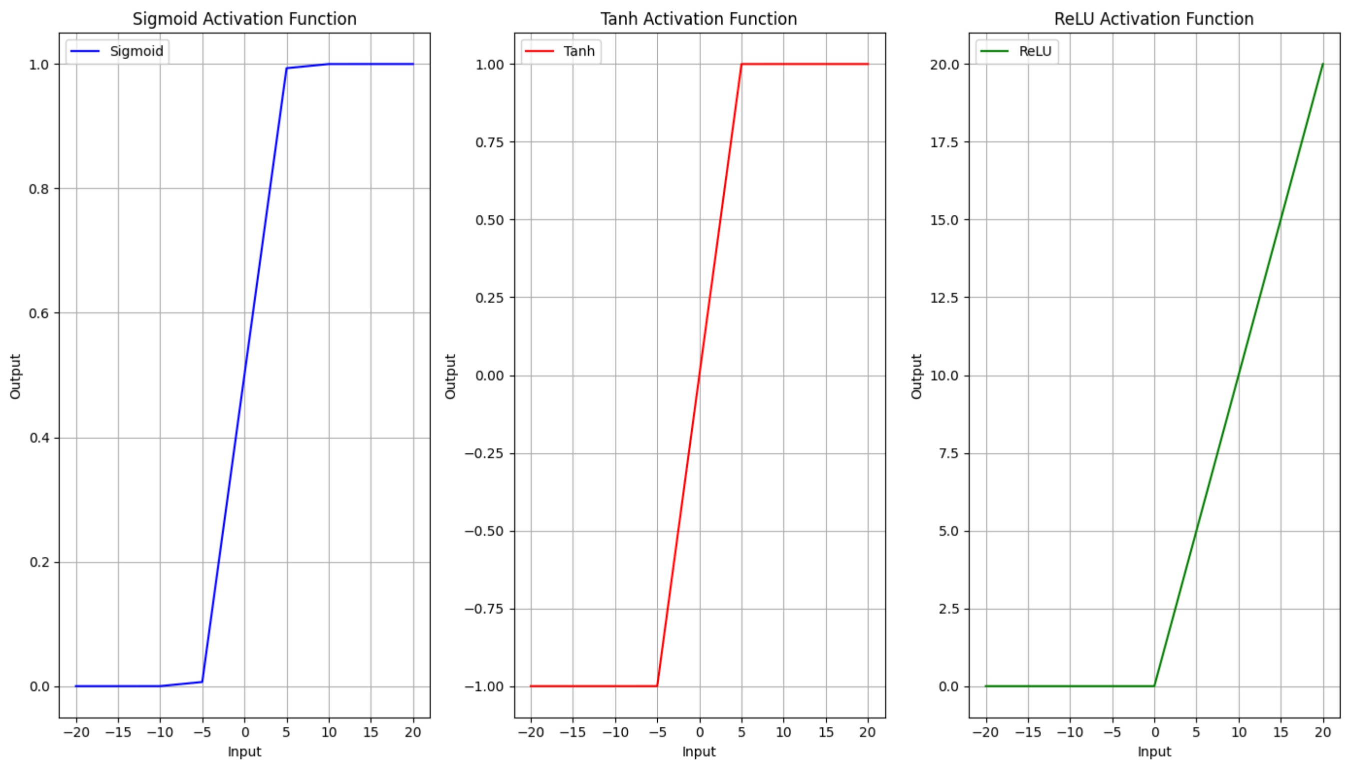
Q7. Develop a simple neural network using Python to classify the XOR dataset. The XOR dataset is a small, synthetic dataset consisting of four samples with two input features each and binary output labels. The task is to implement a neural network and the backpropagation algorithm from scratch. Here is the dataset: (0, 0) yields an output of 0, (0, and 1) with output of 1. (1, 0) yields an output of 1, and (1, 1) with output of 0.
Q8. Develop Python code to implement and visualize the sigmoid, tanh, and ReLU activation functions. Plot graphs of their outputs against a range of input values to analyze their impact on input transformation. Use the following input values (-20, -15, -10,-5, 0, 5, 10, 15, 20).

```
import numpy as np

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

# Activation functions and their derivatives
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

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

# Neural network parameters
input_layer_neurons = 2
hidden_layer_neurons = 2
output_neurons = 1
learning_rate = 0.1
epochs = 10000

# Initializing weights and biases
wh = np.random.uniform(size=(input_layer_neurons, hidden_layer_neurons))
bh = np.random.uniform(size=(1, hidden_layer_neurons))
wout = np.random.uniform(size=(hidden_layer_neurons, output_neurons))
bout = np.random.uniform(size=(1, output_neurons))

# Training the neural network
```

```
for epoch in range(epochs):
    # Forward propagation
    hidden_layer_input = np.dot(X, wh) + bh
    hidden_layer_activation = sigmoid(hidden_layer_input)
    output_layer_input = np.dot(hidden_layer_activation, wout) + bout
    predicted_output = sigmoid(output_layer_input)

    # Backpropagation
    error = y - predicted_output
    d_predicted_output = error * sigmoid_derivative(predicted_output)

    error_hidden_layer = d_predicted_output.dot(wout.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_activation)

    # Updating weights and biases
    wout += hidden_layer_activation.T.dot(d_predicted_output) * learning_rate
    bout += np.sum(d_predicted_output, axis=0, keepdims=True) * learning_rate
    wh += X.T.dot(d_hidden_layer) * learning_rate
    bh += np.sum(d_hidden_layer, axis=0, keepdims=True) * learning_rate

# Output after training
print("Final predicted output:")
print(predicted_output)
```

```
Final predicted output:
[[0.06085127]
 [0.94383449]
 [0.9435124 ]
 [0.06117407]]
```

```
import matplotlib.pyplot as plt

# Activation functions
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

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

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

# Input values
input_values = np.array([-20, -15, -10, -5, 0, 5, 10, 15, 20])

# Compute outputs for each activation function
sigmoid_outputs = sigmoid(input_values)
tanh_outputs = tanh(input_values)
relu_outputs = relu(input_values)

# Plotting the activation functions
plt.figure(figsize=(14, 8))

plt.subplot(1, 3, 1)
plt.plot(input_values, sigmoid_outputs, 'b', label='Sigmoid')
plt.title('Sigmoid Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()

plt.subplot(1, 3, 2)
plt.plot(input_values, tanh_outputs, 'r', label='Tanh')
plt.title('Tanh Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()

plt.subplot(1, 3, 3)
plt.plot(input_values, relu_outputs, 'g', label='ReLU')
plt.title('ReLU Activation Function')
plt.xlabel('Input')
plt.ylabel('Output')
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()
```



Q9. Study how different learning rates affect a neural network's regression performance using a synthetic dataset of 500 samples with 10 features each with continuous target variable affected by Gaussian noise. Develop a neural network regression model with varied learning rates (e.g., 0.001, 0.0015, 0.01, 0.015, 0.1, 0.15), train for 50 epochs, evaluate on a test set, and analyze the validation and training loss. Objective: Identify the optimal learning rate for maximizing model performance.