Assignment

Implement ann for binary classification and regression use different activation and optimisers find the model performance, use dropout layers check the performance of model

---

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Dropout
# Load data and preprocess it as needed
# Build the model for binary classification
model = keras.Sequential()
model.add(Dense(64, activation='relu', input_dim=input_shape))
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32)
# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}, Test Accuracy: {accuracy}")
import numpy as np
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
# Load the Boston Housing dataset
boston = load_boston()
X, y = boston.data, boston.target
# Normalize the input data
```

```
scaler = StandardScaler()
X = scaler.fit_transform(X)
# Split the dataset 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)
# Create a regression model with different activation and optimizer
def create_regression_model(activation, optimizer):
    model = keras.Sequential([
        Dense(64, activation=activation, input_shape=(X_train.shape[1],)),
        Dense(64, activation=activation),
        Dropout(0.2),  # Dropout layer
        Dense(1)  # Regression output
    ])
    model.compile(optimizer=optimizer, loss='mean_squared_error')
    return model
# Train and evaluate models with different activation and optimizer
activation_functions = ['relu', 'tanh']
optimizers = [SGD(learning_rate=0.01), Adam(learning_rate=0.001)]
for activation in activation_functions:
    for optimizer in optimizers:
        model = create_regression_model(activation, optimizer)
        model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))
        print(f"Activation: {activation}, Optimizer: {optimizer}")
        test_loss = model.evaluate(X_test, y_test)
        print(f"Test loss: {test_loss}")
```

Output:

```
Epoch 1/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1362 - accuracy: 0.9589 - val_loss: 0.0786 - val_accuracy: 0.9761
...
```

```
Epoch 5/5

1875/1875 [==============================] - 3s 2ms/step - loss: 0.0165 - accuracy: 0.9930 -
val_loss: 0.0756 - val_accuracy: 0.9804

Activation: relu, Optimizer: <tensorflow.python.keras.optimizer_v2.gradient_descent.SGD object at
0x7f6d6c3bf6a0>

313/313 [==============================] - 0s 1ms/step - loss: 0.0756 - accuracy: 0.9804

Test accuracy: 0.9803999667167664
```

```
Epoch 1/50

13/13 [==============================] - 1s 8ms/step - loss: 589.0029 - val_loss: 642.8690

...

Epoch 50/50

13/13 [==============================] - 0s 7ms/step - loss: 26.9779 - val_loss: 30.9978

Activation: relu, Optimizer: <tensorflow.python.keras.optimizer_v2.gradient_descent.SGD object at
0x7f6d6c3bf6a0>

4/4 [==============================] - 0s 3ms/step - loss: 30.9978

Test loss: 30.99782943725586
```