Implementation of ANN for Binary Classification and Regression:-

Binary Classification Model:

In [1]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam, RMSprop
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import make_classification

# Generate synthetic data for binary classification task
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0, n_clusters_per_class=1, weights=[0.975],
                           flip_y=0, random_state=1)

# 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)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(8000, 2) (2000, 2) (8000,) (2000,)
```

In [2]:
```
# Define the model architecture
model = Sequential()
model.add(Dense(128, input_shape=(X_train.shape[1],), activation='relu'))
model.add(Dropout(0.2))  # Dropout layer for regularization
model.add(Dense(64, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))  # Output layer with sigmoid activation for binary classification
```

Compilation of the model with Adam optimizer:

In [3]:
```
# Adam optimizer
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.1)
```

```
# Evaluate the model on test data
accuracy = model.evaluate(X_test, y_test)[1]
print("Test Accuracy:", accuracy)
```

```
Epoch 1/10
225/225 [==============================] - 1s 3ms/step - loss: 0.1691 - accuracy: 0.9422 - val_loss: 0.0808 - val_accuracy: 0.9
787
Epoch 2/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0721 - accuracy: 0.9803 - val_loss: 0.0574 - val_accuracy: 0.9
850
Epoch 3/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0578 - accuracy: 0.9840 - val_loss: 0.0461 - val_accuracy: 0.9
887
Epoch 4/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0529 - accuracy: 0.9850 - val_loss: 0.0401 - val_accuracy: 0.9
912
Epoch 5/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0495 - accuracy: 0.9861 - val_loss: 0.0412 - val_accuracy: 0.9
887
Epoch 6/10
225/225 [==============================] - 1s 3ms/step - loss: 0.0484 - accuracy: 0.9864 - val_loss: 0.0378 - val_accuracy: 0.9
912
Epoch 7/10
225/225 [==============================] - 1s 2ms/step - loss: 0.0465 - accuracy: 0.9874 - val_loss: 0.0388 - val_accuracy: 0.9
887
Epoch 8/10
225/225 [==============================] - 1s 2ms/step - loss: 0.0463 - accuracy: 0.9874 - val_loss: 0.0379 - val_accuracy: 0.9
900
Epoch 9/10
225/225 [==============================] - 1s 2ms/step - loss: 0.0455 - accuracy: 0.9879 - val_loss: 0.0400 - val_accuracy: 0.9
862
Epoch 10/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0461 - accuracy: 0.9875 - val_loss: 0.0377 - val_accuracy: 0.9
912
63/63 [==============================] - 0s 2ms/step - loss: 0.0406 - accuracy: 0.9870
Test Accuracy: 0.9869999885559082
```

Compilation of the model with RMSprop optimizer:

In [4]:
```
# RMSprop optimizer
model.compile(optimizer=RMSprop(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])
```

```
# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.1)

# Evaluate the model on test data
accuracy = model.evaluate(X_test, y_test)[1]
print("Test Accuracy:", accuracy)
```

```
Epoch 1/10
225/225 [==============================] - 1s 3ms/step - loss: 0.0460 - accuracy: 0.9881 - val_loss: 0.0392 - val_accuracy: 0.9
862
Epoch 2/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0452 - accuracy: 0.9879 - val_loss: 0.0375 - val_accuracy: 0.9
912
Epoch 3/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0438 - accuracy: 0.9876 - val_loss: 0.0403 - val_accuracy: 0.9
862
Epoch 4/10
225/225 [==============================] - 1s 2ms/step - loss: 0.0444 - accuracy: 0.9876 - val_loss: 0.0367 - val_accuracy: 0.9
912
Epoch 5/10
225/225 [==============================] - 1s 2ms/step - loss: 0.0448 - accuracy: 0.9878 - val_loss: 0.0368 - val_accuracy: 0.9
912
Epoch 6/10
225/225 [==============================] - 1s 2ms/step - loss: 0.0448 - accuracy: 0.9869 - val_loss: 0.0392 - val_accuracy: 0.9
875
Epoch 7/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0443 - accuracy: 0.9883 - val_loss: 0.0378 - val_accuracy: 0.9
887
Epoch 8/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0442 - accuracy: 0.9885 - val_loss: 0.0373 - val_accuracy: 0.9
912
Epoch 9/10
225/225 [==============================] - 1s 2ms/step - loss: 0.0452 - accuracy: 0.9883 - val_loss: 0.0372 - val_accuracy: 0.9
912
Epoch 10/10
225/225 [==============================] - 1s 2ms/step - loss: 0.0438 - accuracy: 0.9889 - val_loss: 0.0374 - val_accuracy: 0.9
912
63/63 [==============================] - 0s 2ms/step - loss: 0.0388 - accuracy: 0.9885
Test Accuracy: 0.9884999990463257
```

Regression Model:

In [5]:
```
from tensorflow.keras.metrics import MeanSquaredError
from sklearn.metrics import mean_squared_error

# Generate synthetic data for regression task
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0, n_clusters_per_class=1, weights=[0.975],
                           flip_y=0, random_state=1)

# 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)
```

In [6]:
```
# Define the regression model architecture
regression_model = Sequential()
regression_model.add(Dense(128, input_shape=(X_train.shape[1],), activation='relu'))
regression_model.add(Dropout(0.2))  # Dropout layer for regularization
regression_model.add(Dense(64, activation='relu'))
regression_model.add(Dense(1, activation='linear'))  # Output layer with linear activation for regression
```

Compilation of the model with Adam optimizer:

In [7]:
```
# Adam optimizer
regression_model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error', metrics=[MeanSquaredError()])

# Train the regression model
regression_model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.1)

# Evaluate the regression model on test data
predictions = regression_model.predict(X_test)
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)
```

```
Epoch 1/10
225/225 [==============================] - 1s 3ms/step - loss: 0.0206 - mean_squared_error: 0.0206 - val_loss: 0.0117 - val_mea
n_squared_error: 0.0117
Epoch 2/10
225/225 [==============================] - 1s 2ms/step - loss: 0.0140 - mean_squared_error: 0.0140 - val_loss: 0.0102 - val_mea
n_squared_error: 0.0102
Epoch 3/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0130 - mean_squared_error: 0.0130 - val_loss: 0.0102 - val_mea
n_squared_error: 0.0102
Epoch 4/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0124 - mean_squared_error: 0.0124 - val_loss: 0.0097 - val_mea
n_squared_error: 0.0097
Epoch 5/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0121 - mean_squared_error: 0.0121 - val_loss: 0.0099 - val_mea
n_squared_error: 0.0099
Epoch 6/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0122 - mean_squared_error: 0.0122 - val_loss: 0.0101 - val_mea
n_squared_error: 0.0101
Epoch 7/10
225/225 [==============================] - 1s 2ms/step - loss: 0.0120 - mean_squared_error: 0.0120 - val_loss: 0.0093 - val_mea
n_squared_error: 0.0093
Epoch 8/10
225/225 [==============================] - 1s 2ms/step - loss: 0.0118 - mean_squared_error: 0.0118 - val_loss: 0.0102 - val_mea
n_squared_error: 0.0102
Epoch 9/10
225/225 [==============================] - 1s 3ms/step - loss: 0.0117 - mean_squared_error: 0.0117 - val_loss: 0.0097 - val_mea
n_squared_error: 0.0097
Epoch 10/10
225/225 [==============================] - 1s 2ms/step - loss: 0.0119 - mean_squared_error: 0.0119 - val_loss: 0.0102 - val_mea
n_squared_error: 0.0102
63/63 [==============================] - 0s 2ms/step
Mean Squared Error: 0.011496093449761855
```

Compilation of the model with RMSprop optimizer:

In [8]:
```
# RMSprop optimizer
regression_model.compile(optimizer=RMSprop(learning_rate=0.001), loss='mean_squared_error', metrics=[MeanSquaredError()])

# Train the regression model
regression_model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.1)

# Evaluate the regression model on test data
```

```
predictions = regression_model.predict(X_test)
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)
```

```
Epoch 1/10
225/225 [==============================] - 1s 2ms/step - loss: 0.0117 - mean_squared_error: 0.0117 - val_loss: 0.0100 - val_mea
n_squared_error: 0.0100
Epoch 2/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0115 - mean_squared_error: 0.0115 - val_loss: 0.0096 - val_mea
n_squared_error: 0.0096
Epoch 3/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0114 - mean_squared_error: 0.0114 - val_loss: 0.0091 - val_mea
n_squared_error: 0.0091
Epoch 4/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0115 - mean_squared_error: 0.0115 - val_loss: 0.0096 - val_mea
n_squared_error: 0.0096
Epoch 5/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0113 - mean_squared_error: 0.0113 - val_loss: 0.0097 - val_mea
n_squared_error: 0.0097
Epoch 6/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0116 - mean_squared_error: 0.0116 - val_loss: 0.0094 - val_mea
n_squared_error: 0.0094
Epoch 7/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0113 - mean_squared_error: 0.0113 - val_loss: 0.0092 - val_mea
n_squared_error: 0.0092
Epoch 8/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0111 - mean_squared_error: 0.0111 - val_loss: 0.0094 - val_mea
n_squared_error: 0.0094
Epoch 9/10
225/225 [==============================] - 1s 2ms/step - loss: 0.0110 - mean_squared_error: 0.0110 - val_loss: 0.0094 - val_mea
n_squared_error: 0.0094
Epoch 10/10
225/225 [==============================] - 0s 2ms/step - loss: 0.0112 - mean_squared_error: 0.0112 - val_loss: 0.0098 - val_mea
n_squared_error: 0.0098
63/63 [==============================] - 0s 1ms/step
Mean Squared Error: 0.011704694829095669
```