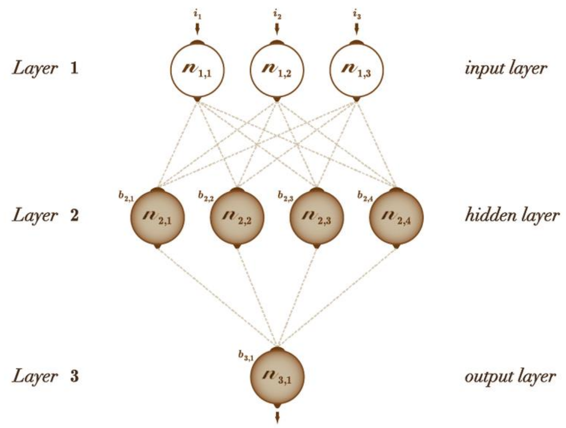
Experiment -2

Aim-Implementation of ANN Model for Regression and Classification Problem in Python.

Theory-

Artificial Neural Networks (ANNs) are a subset of machine learning models inspired by the structure and function of the human brain. They consist of interconnected layers of nodes (neurons) that work together to solve complex problems, including both classification and regression tasks. ANNs are highly flexible and can approximate a wide variety of functions, making them powerful tools for various types of predictions.

Structure of an Artificial Neural Network



1. **Input Layer**: The input layer receives the raw data and passes it on to the subsequent layers. Each neuron in the input layer represents a feature in the dataset.
2. **Hidden Layers**: Between the input and output layers, there can be one or more hidden layers. Each hidden layer consists of neurons that apply linear transformations followed by non-linear activation functions to the input data. The depth (number of hidden layers) and width (number of neurons per layer) can vary depending on the complexity of the task.
3. **Output Layer**: The output layer produces the final predictions. For classification tasks, it usually consists of neurons with activation functions like softmax, which outputs probabilities for each class. For regression tasks, the output layer typically has a single neuron with a linear activation function to produce a continuous value.

Activation Functions

Activation functions introduce non-linearity into the network, enabling it to learn and represent more complex relationships. Common activation functions include:

- **Sigmoid**: Outputs values between 0 and 1, often used in binary classification.
- **ReLU (Rectified Linear Unit)**: Outputs the input directly if it's positive; otherwise, it outputs zero. ReLU is widely used in hidden layers due to its efficiency and simplicity.

- **Tanh**: Outputs values between -1 and 1, often used when the output can be negative or positive.
- **Softmax**: Converts the output layer into a probability distribution, typically used in multi-class classification.

ANN for Classification

In classification tasks, the goal of an ANN is to categorize input data into predefined classes. The network learns to map inputs to the correct class labels by minimizing a loss function that measures the difference between the predicted and actual labels.

- **Binary Classification**: For tasks where there are only two possible outcomes (e.g., spam vs. not spam), the output layer typically contains a single neuron with a sigmoid activation function. The network outputs a probability between 0 and 1, which can be thresholded to make a binary decision.
- **Multi-Class Classification**: For tasks with more than two classes (e.g., recognizing digits 0-9), the output layer consists of multiple neurons, each corresponding to a class. The softmax activation function is used to ensure that the output values sum to 1, representing the probability of each class. The class with the highest probability is selected as the prediction.

Loss Function for Classification:

- **Binary Cross-Entropy**: Used for binary classification, it measures the difference between two probability distributions (predicted and actual).
- **Categorical Cross-Entropy**: Used for multi-class classification, it extends the concept of binary cross-entropy to multiple classes.

ANN for Regression

In regression tasks, the goal of an ANN is to predict a continuous output value based on the input data. Unlike classification, where the output is a category, regression outputs a real number.

- **Structure**: The output layer in a regression ANN typically has a single neuron with a linear activation function, meaning it can output a range of values.
- **Applications**: Regression ANNs are used in tasks like predicting house prices, forecasting stock prices, or estimating any quantity that can take a continuous value.

Loss Function for Regression:

- **Mean Squared Error (MSE)**: The most commonly used loss function for regression tasks, MSE measures the average squared difference between the predicted and actual values.
- **Mean Absolute Error (MAE)**: Another loss function that measures the average absolute difference between predicted and actual values, offering a more robust metric in the presence of outliers.

Training an ANN

Training an ANN involves adjusting the weights and biases of the neurons to minimize the loss function. This is done through a process called **backpropagation**, which computes the gradient of the loss function with respect to each weight by applying the chain rule. An **optimizer** like Stochastic Gradient Descent (SGD) or Adam is used to update the weights iteratively in the direction that reduces the loss.

Key Concepts in ANN Training:

- **Learning Rate**: A hyperparameter that controls how much the weights are adjusted during each update. A learning rate that is too high may cause the model to converge too quickly to a suboptimal solution, while a learning rate that is too low may result in a prolonged training time.
- **Epochs**: The number of times the entire training dataset passes through the network. More epochs allow the network to learn more but also increase the risk of overfitting.
- **Batch Size**: The number of training samples used to calculate the gradient in each update step. Small batch sizes can lead to noisy gradient estimates, while large batch sizes require more memory.

Overfitting and Regularization

Overfitting occurs when the ANN learns the training data too well, including its noise and outliers, leading to poor generalization on new data. Regularization techniques like **L2 regularization** (which penalizes large weights), **dropout** (randomly dropping neurons during training), and **early stopping** (halting training when performance on a validation set stops improving) are commonly used to mitigate overfitting.

Evaluation Metrics

For classification tasks, common evaluation metrics include **accuracy**, **precision**, **recall**, and the **F1 score**. For regression tasks, metrics like **R-squared**, **MSE**, and **MAE** are used to assess the model's performance.

Code and Output-

```
#Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import tensorflow.keras as keras
import sklearn
# Regression
dfr=pd.read_csv("Data.csv")
dfr.head()
```

	AT	V	AP	RH	PE
0	14.96	41.76	1024.07	73.17	463.26
1	25.18	62.96	1020.04	59.08	444.37
2	5.11	39.40	1012.16	92.14	488.56
3	20.86	57.32	1010.24	76.64	446.48
4	10.82	37.50	1009.23	96.62	473.90

```
x=dfr.iloc[:,:-1].values
y=dfr.iloc[:,-1].values
# Split data into training and test sets
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=42,test_size=0.2)
ann=tf.keras.models.Sequential([
    tf.keras.layers.Dense(6,activation="relu"),
    tf.keras.layers.Dense(6,activation="relu"),
    tf.keras.layers.Dense(1)
])
ann.compile(optimizer="adam",loss="mean_squared_error",metrics=["accuracy"])
ann.fit(x_train,y_train,batch_size=32,epochs=100)
```

```
Epoch 1/100
240/240 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.0000e+00 - loss: 140475.4844
Epoch 2/100
240/240 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.0000e+00 - loss: 471.2336
Epoch 3/100
240/240 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.0000e+00 - loss: 399.8553
Epoch 4/100
240/240 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.0000e+00 - loss: 381.5014
Epoch 5/100
240/240 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.0000e+00 - loss: 328.1067
Epoch 6/100
```

```
y_pred=ann.predict([x_test])
print(np.concatenate((y_pred.reshape(len(y_pred),1),y_test.reshape(len(y_test),1)),1))
```

```
60/60 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
[[454.5894165  455.27      ]
 [436.73260498 436.31      ]
 [431.58242798 440.68      ]
 ...
 [479.16275024 479.53      ]
 [433.69058228 435.76      ]
 [457.84527588 457.1       ]]
```

from sklearn.metrics import r2_score

r2_score(y_test, y_pred)

```
0.9002140799169029
```

#Classification

dfc=pd.read_csv("Churn_Modelling.csv")

dfc.head()

RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1
2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0
3	15619304	Onio	502	France	Female	42	8	159660.80	3	1
4	15701354	Boni	699	France	Female	39	1	0.00	2	0
5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1

x=dfc.iloc[:,3:-1].values

y=dfc.iloc[:,-1].values

 # label encoding

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

 # Gender

x[:,2]=le.fit_transform(x[:,2])

 # Geography

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder

ct=ColumnTransformer(transformers=[('encoder',OneHotEncoder(),[1])],remainder='passthrough')

x=np.array(ct.fit_transform(x))

```
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=42,test_size=0.2)
#Standard Scaling
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_test=sc.fit_transform(x_test)
x_train=sc.transform(x_train)
ann_c=tf.keras.models.Sequential([
    tf.keras.layers.Dense(6,activation="relu"),
    tf.keras.layers.Dense(6,activation="relu"),
    tf.keras.layers.Dense(1,activation="sigmoid")
])
ann_c.compile(loss="binary_crossentropy",optimizer="adam",metrics=["accuracy"])
ann_c.fit(x_train, y_train, batch_size = 32, epochs = 100)
```

```
Epoch 1/100
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.6261 - loss: 0.6749
Epoch 2/100
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8037 - loss: 0.4818
Epoch 3/100
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8069 - loss: 0.4404
Epoch 4/100
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8082 - loss: 0.4329
Epoch 5/100
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8173 - loss: 0.4241
```

```
print(ann_c.predict(sc.transform([[1, 0, 0,600,1,40,3,60000,2,2,2,50000]])))
print(ann_c.predict(sc.transform([[1, 0, 0,600,1,40,3,60000,2,2,2,50000]]))>0.5)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 190ms/step
[[0.02300457]]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
[[False]]
```

```
y_pred=ann_c.predict(x_test)
y_pred=(y_pred>0.5)
print(np.concatenate((y_pred.reshape(len(y_pred),1),y_test.reshape(len(y_test),1)),1))
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━
[[0 0]
 [0 0]
 [0 0]
 ...
 [1 1]
 [0 1]
 [0 1]]
```

from sklearn.metrics import confusion_matrix,accuracy_score,classification_report

import seaborn as sns

cm=confusion_matrix(y_test,y_pred)

sns.heatmap(cm,annot=True,fmt='g')

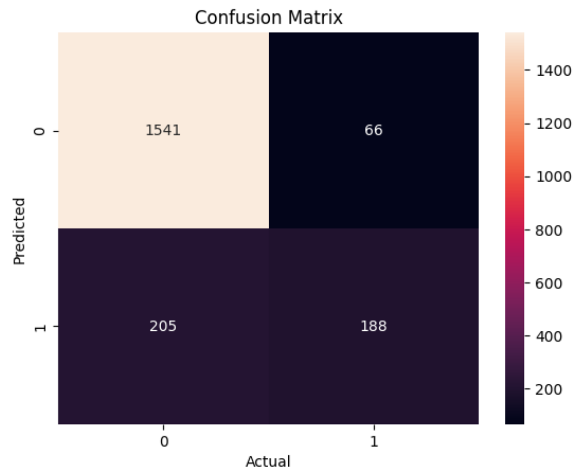
plt.title("Confusion Matrix")

plt.xlabel("Actual")

plt.ylabel("Predicted")

plt.savefig("confusion_matrix1.png")

plt.show()



print(accuracy_score(y_test,y_pred))

```
0.8645
```

print(classification_report(y_test,y_pred))

```
              precision    recall  f1-score   support

           0       0.88      0.96      0.92      1607
           1       0.74      0.48      0.58       393

    accuracy                           0.86      2000
   macro avg       0.81      0.72      0.75      2000
weighted avg       0.85      0.86      0.85      2000
```

Viva Voice-

1. **What is an Artificial Neural Network (ANN)?**

- **Answer**: An Artificial Neural Network (ANN) is a computational model inspired by the structure and functioning of the human brain. It consists of interconnected layers of neurons (nodes) that process data by applying linear transformations and non-linear activation functions to solve complex problems like classification and regression.

2. How does an ANN differ when used for classification versus regression?

- **Answer**: In classification, the ANN's output layer typically uses activation functions like softmax or sigmoid to produce categorical outputs (classes). In regression, the output layer usually has a linear activation function to predict continuous values. The loss functions also differ: classification often uses cross-entropy loss, while regression uses mean squared error (MSE).

3. What is the purpose of activation functions in an ANN?

- **Answer**: Activation functions introduce non-linearity into the network, allowing it to model complex relationships between the input and output. Without activation functions, the network would only be able to model linear functions, severely limiting its capability to solve real-world problems.

4. What is the role of the loss function in training an ANN?

- **Answer**: The loss function measures how far the network's predictions are from the actual values. During training, the objective is to minimize this loss function by adjusting the network's weights through optimization techniques like gradient descent. This process is crucial for improving the model's accuracy.

5. Explain the concept of backpropagation in ANN.

- **Answer**: Backpropagation is the algorithm used to train ANNs by adjusting weights to minimize the loss function. It works by calculating the gradient of the loss function with respect to each weight in the network, then updating the weights in the direction that reduces the loss. This is done iteratively for each layer in the network.

6. What are some common activation functions used in classification and regression tasks?

- **Answer**: For classification tasks, common activation functions include sigmoid (for binary classification) and softmax (for multi-class classification). For regression tasks, a linear activation function is often used in the output layer to produce continuous values.