EX.NO : 8 DATE :

REG NO:220701021

IMPLEMENTING ARTIFICIAL NEURAL NETWORKS FOR AN APPLICATION USING PYTHON – CLASSIFICATION

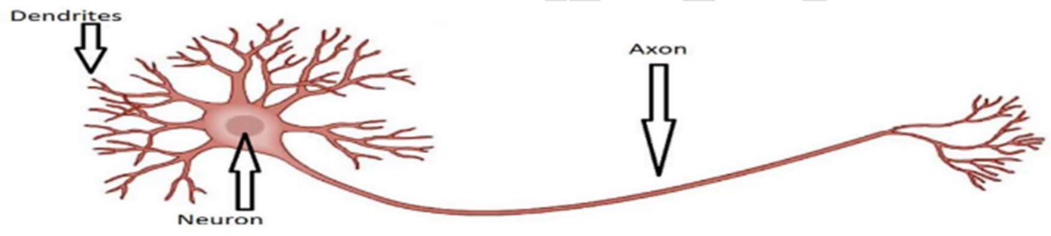
AIM :

To implementing artificial neural networks for an application in classification using python.

What is an Artificial Neural Network?

Artificial Neural Network is much similar to the human brain. The human Brain consist of neurons. These

neurons are connected. In the human brain, neuron looks something like this...

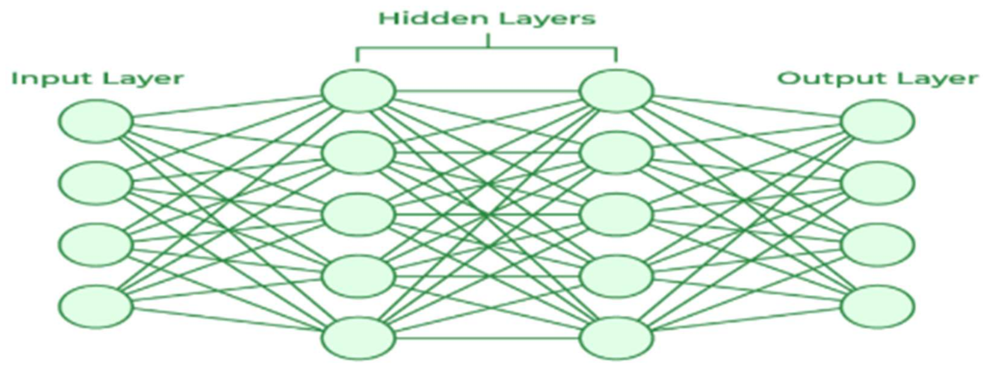


What do you think?

When you touch the hot surface, how you suddenly remove your hand?. This is the procedure that happens inside you.When you touch some hot surface. Then automatically your skin sends a signal to the neuron.And then the neuron takes a decision, “Remove your hand”. So that’s all about the Human Brain. In the same way, Artificial Neural Network works.

Artificial Neural Networks

Artificial Neural Networks contain artificial neurons which are called units. These units are arranged in a series of layers that together constitute the whole Artificial Neural Network in a system. A layer can have only a dozen units or millions of units as this depends on how the complex neural networks will be required to learn the hidden patterns in the dataset. Commonly, Artificial Neural Network has an input layer, an output layer as well as hidden layers. The input layer receives data from the outside world which the neural network needs to analyze or learn about. Then this data passes through one or multiple hidden layers that transform the input into data that is valuablefor the output layer. Finally, the output layer provides an output in the form of a response of the Artificial Neural Networks to input data provided. The structures and operations of human neurons serve as the basis for artificial neural networks. It is also known as neural networks or neural nets. The input layer of an artificial neural network is the first layer, and it receives input from external sources and releases it to the hidden layer, which is the second layer. In the hidden layer, each neuron receives input from the previous layer neurons, computes the weighted sum, and sends it to the neurons in the next layer. These connections are weighted means effects of the inputs from the previous layer are optimized more or less by assigning different-different weights to each input and it is adjusted during the training process by optimizing these weights for improved model performance.

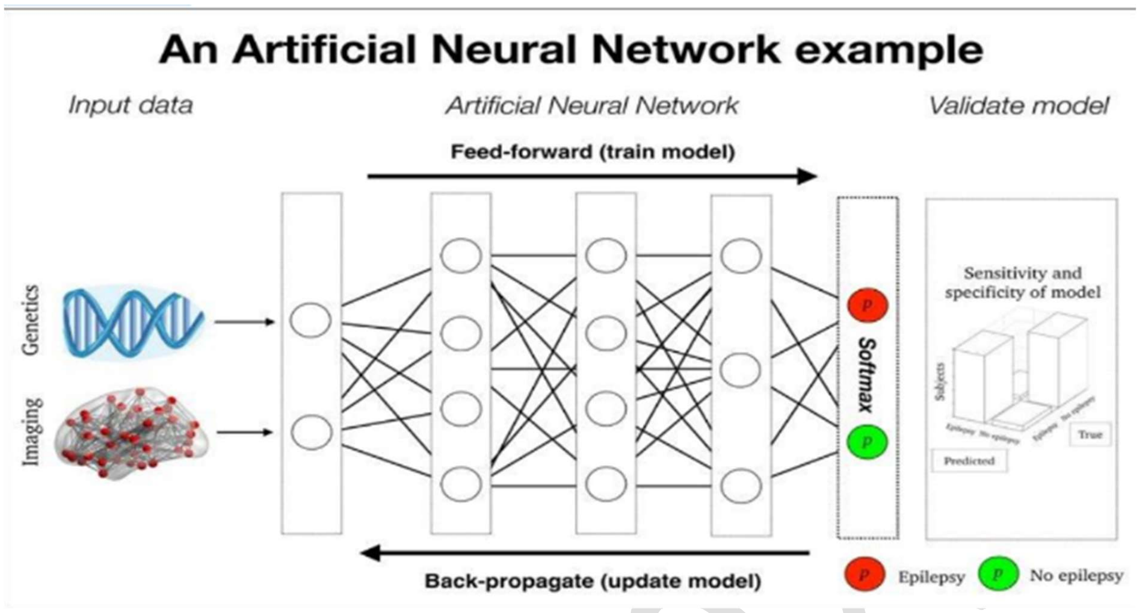


What are the types of Artificial Neural Networks?

1. Feedforward Neural Network
2. Convolutional Neural Network
3. Modular Neural Network
4. Radial basis function Neural Network
5. Recurrent Neural Network

Applications of Artificial Neural Networks

1. Social Media
2. Marketing and Sales
3. Healthcare
4. Personal Assistants

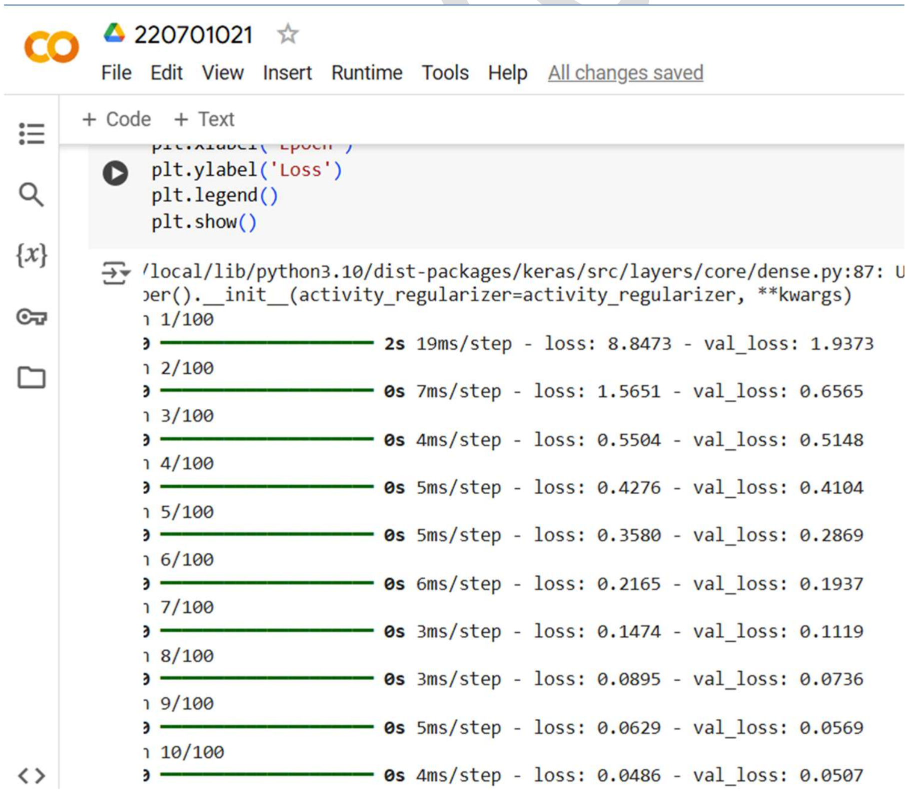

An Artificial Neural Network example
Input data
Artificial Neural Network
Validate model
Feed-forward (train model)
Genetics
Imaging
Softmax
Sensitivity and specificity of model
Subjects
Predicted
True
Back-propagate (update model)
Epilepsy
No epilepsy

CODE:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
import matplotlib.pyplot as plt

# Generating synthetic dataset for demonstration
# Replace this with your own dataset
np.random.seed(42)
X = np.random.rand(1000, 3)  # 1000 samples, 3 features
y = 3 * X[:, 0] + 2 * X[:, 1] ** 2 + 1.5 * np.sin(X[:, 2] * np.pi) + np.random.normal(0, 0.1, 1000)  # Non-linear relationship

# 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)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Building the ANN model
model = Sequential()
```

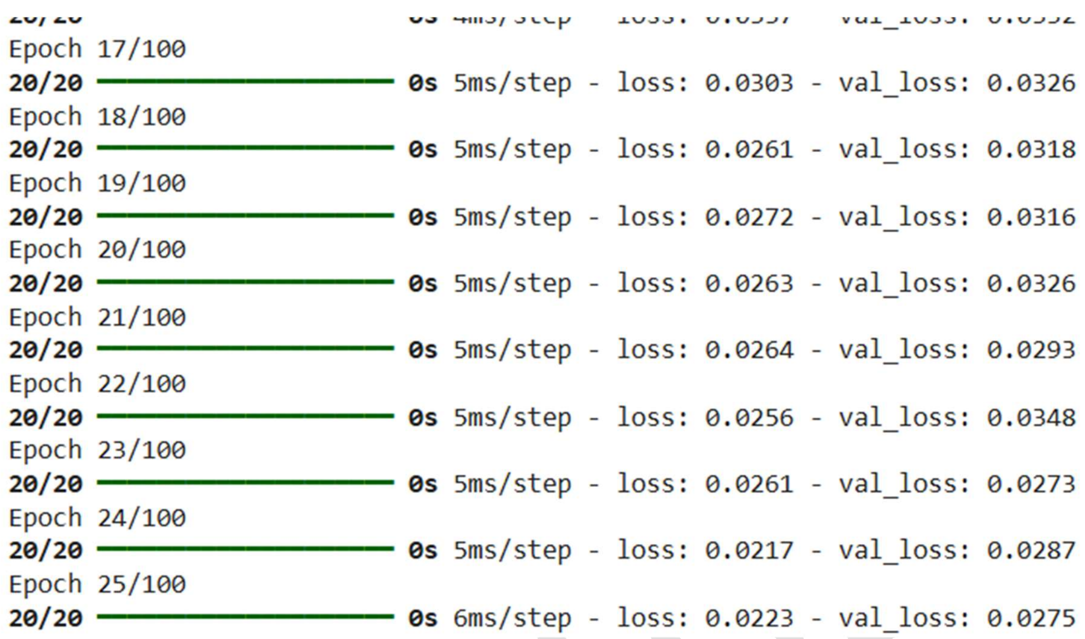
```python
# Second hidden layer with 10 neurons and ReLU activation
model.add(Dense(10, activation='relu'))

# Output layer with linear activation for regression
model.add(Dense(1, activation='linear'))

# Compiling the model
model.compile(optimizer=Adam(learning_rate=0.01), loss='mean_squared_error')

# Training the model
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=1)

# Making predictions
y_pred = model.predict(X_test)

# Evaluating the model
mse = np.mean((y_test - y_pred.flatten()) ** 2)
print(f'Mean Squared Error: {mse:.4f}')

# Plotting training history
plt.figure(figsize=(12, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

OUTPUT:



```python
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: U
per().__init__(activity_regularizer=activity_regularizer, **kwargs)
1 1/100
━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 8.8473 - val_loss: 1.9373
1 2/100
━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 1.5651 - val_loss: 0.6565
1 3/100
━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.5504 - val_loss: 0.5148
1 4/100
━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.4276 - val_loss: 0.4104
1 5/100
━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.3580 - val_loss: 0.2869
1 6/100
━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.2165 - val_loss: 0.1937
1 7/100
━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.1474 - val_loss: 0.1119
1 8/100
━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0895 - val_loss: 0.0736
1 9/100
━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0629 - val_loss: 0.0569
1 10/100
━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0486 - val_loss: 0.0507
```



```
Epoch 17/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0303 - val_loss: 0.0326
Epoch 18/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0261 - val_loss: 0.0318
Epoch 19/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0272 - val_loss: 0.0316
Epoch 20/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0263 - val_loss: 0.0326
Epoch 21/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0264 - val_loss: 0.0293
Epoch 22/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0256 - val_loss: 0.0348
Epoch 23/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0261 - val_loss: 0.0273
Epoch 24/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0217 - val_loss: 0.0287
Epoch 25/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0223 - val_loss: 0.0275
```

RESULT:

The above ANN program is successfully implemented.