Feed Forward Neural Network

A feed-forward neural network (FDD), is a type of artificial neural network that is widely used for a variety of tasks, including classification, regression, and pattern recognition. It is called "feed-forward" because the information flow is unidirectional, from the input layer through the hidden layers to the output layer, without any feedback connections.

Architecture of FDD:

Input Layer: This layer receives the input data, which can be numerical values, categorical data, or any other form of information relevant to the task at hand.

Hidden Layers: Between the input and output layers, there can be one or more hidden layers. Each hidden layer consists of a set of neurons (or nodes) that apply a non-linear transformation to the input received from the previous layer.

Output Layer: This layer produces the final output of the neural network, which can be a single value (for regression tasks) or a set of probabilities (for classification tasks).

Operation:

Forward Propagation: During the forward propagation phase, the input data is fed into the input layer, and the outputs of each layer are computed by applying a non-linear activation function (e.g., sigmoid, ReLU) to the weighted sum of the inputs from the previous layer.

Activation Functions: Activation functions introduce non-linearity into the neural network, allowing it to model complex relationships between the input and output data. Common activation functions include the sigmoid function, the rectified linear unit (ReLU), and the hyperbolic tangent (tanh).

Weights and Biases: Each connection between neurons in adjacent layers has an associated weight, which determines the strength of the connection. Additionally, each neuron has a bias term that helps the neuron to shift its activation function left or right.

Backpropagation: During the training phase, the neural network's weights and biases are adjusted using an optimization algorithm, such as gradient descent or one of its variants (e.g., Adam, RMSProp). This process, known as backpropagation, involves computing the gradients of the loss function with respect to the weights and biases, and updating the weights and biases in the direction that minimizes the loss function.

Loss Function: The loss function measures the difference between the neural network's predicted output and the true output (target) for a given input sample. Common loss functions include mean squared error (MSE) for regression tasks and cross-entropy loss for classification tasks.

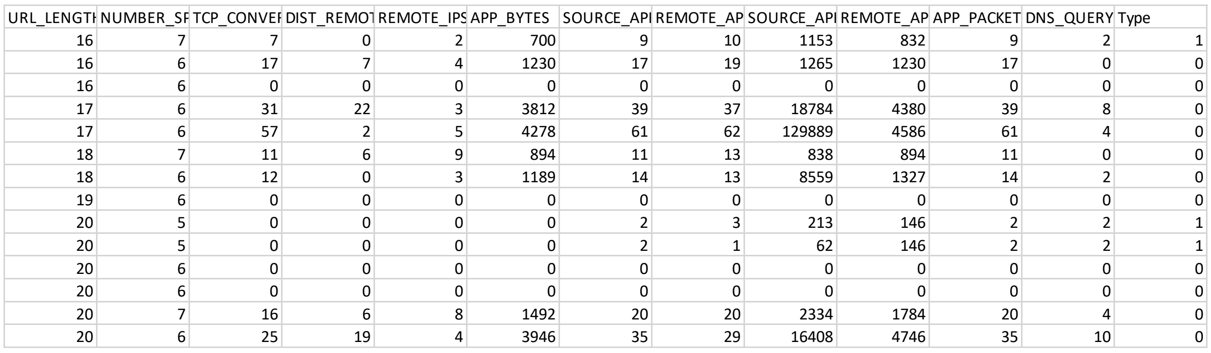
Regularization: To prevent overfitting, which occurs when the neural network models the training data too closely and fails to generalize well to new data, various regularization techniques can be applied, such as L1 or L2 regularization (weight decay), dropout, or early stopping.

FDDs are widely used in various applications, including image recognition, natural language processing, speech recognition, and many other domains where patterns need to be learned from data. Despite their simplicity compared to more complex neural network architectures like recurrent neural networks (RNNs) or convolutional neural networks (CNNs), feed-forward neural networks can still achieve remarkable performance on many tasks, especially when combined with techniques like transfer learning or ensemble methods.

david_dataset1:

Dataset is downloaded from: https://www.kaggle.com/datasets/xwolf12/malicious-and-benign-websites

Data set is modified (some columns and rows are removed)



URL_LENGTI	NUMBER_SI	TCP_CONVEI	DIST_REMOT	REMOTE_IPS	APP_BYTES	SOURCE_API	REMOTE_AP	SOURCE_API	REMOTE_AP	APP_PACKET	DNS_QUERY	Type
16	7	7	0	2	700	9	10	1153	832	9	2	1
16	6	17	7	4	1230	17	19	1265	1230	17	0	0
16	6	0	0	0	0	0	0	0	0	0	0	0
17	6	31	22	3	3812	39	37	18784	4380	39	8	0
17	6	57	2	5	4278	61	62	129889	4586	61	4	0
18	7	11	6	9	894	11	13	838	894	11	0	0
18	6	12	0	3	1189	14	13	8559	1327	14	2	0
19	6	0	0	0	0	0	0	0	0	0	0	0
20	5	0	0	0	0	2	3	213	146	2	2	1
20	5	0	0	0	0	2	1	62	146	2	2	1
20	6	0	0	0	0	0	0	0	0	0	0	0
20	6	0	0	0	0	0	0	0	0	0	0	0
20	7	16	6	8	1492	20	20	2334	1784	20	4	0
20	6	25	19	4	3946	35	29	16408	4746	35	10	0

Python code:

```
import pandas as pd

import tensorflow as tf

import matplotlib.pyplot as plt

from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

from tensorflow.keras.layers import Dense

from tensorflow.keras.models import Sequential


#Import and review dataset

data = pd.read_csv("david_dataset1.csv")

print(data.head())


#Fill in the missing values

data.fillna(data.median(), inplace=True)


#Normalize data

from sklearn.preprocessing import StandardScaler


scaler = StandardScaler()

data[data.columns[:-1]] = scaler.fit_transform(data[data.columns[:-1]])


#Split dataset into training and test datasets

from sklearn.model_selection import train_test_split


X = data.drop('Type', axis=1)

y = data['Type']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
#Define model architecture
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(16, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(8, activation='relu'),
    Dense(1, activation='sigmoid')
])

#Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

#Train the model on training dataset
history = model.fit(X_train, y_train, epochs=5, batch_size=32, validation_split=0.2)

#Use model to validate test dataset
y_pred = model.predict(X_test)

#Evaluate model
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Loss: {loss}, Test Accuracy: {accuracy}')

#Analyze test dataset using ROC/AUC, confusion matrix, predicting probability for test set,
classification report

# ROC/AUC
roc_auc = roc_auc_score(y_test, y_pred)
print(f'ROC AUC: {roc_auc}')
```

```
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred.round())
print('Confusion Matrix:')
print(conf_matrix)

# Classification report
class_report = classification_report(y_test, y_pred.round())
print('Classification Report:')
print(class_report)
```

Results:

```
Classification Report:
           precision   recall  f1-score   support


        0      0.83      1.00      0.91        83

        1      0.00      0.00      0.00        17


   accuracy                        0.83       100

  macro avg      0.41      0.50      0.45       100

weighted avg      0.69      0.83      0.75       100
```