Task 1 (10 points)

Describe the Feed-forward neural network. The answer should be comprehensive. (5 points)

The practical example of application in cybersecurity with data and the python code are mandatory. (5 points)

Place your answer into the task1.pdf file. The source code and the data should be included in the resulting pdf file. No additional files are required.

Feedforward neural networks are a basic type of artificial neural network in which the connections between units do not form cycles. It is one of the simplest forms of neural networks in which information flows in one direction, from the input node through hidden nodes (if any) to the output node. Here's the full breakdown:

Input level:

The input layer consists of nodes representing input features. Each node represents a feature of the input data.

Hidden level:

The hidden layer is the layer of nodes between the input layer and the output layer. Each node in a hidden layer takes input from the node in the previous layer, applies weights and biases, and passes the output to the next layer.

Each hidden layer can have multiple nodes (neurons) that perform calculations on input data.

Output layer:

The output layer produces the final result of the network. The number of nodes in the output layer depends on the type of problem the network is designed to solve. For example, for binary classification, there will be one output node, while for multi-class classification, there will be multiple output nodes.

Activation function:

Activation functions introduce nonlinearity in the output of each neuron. Common activation functions include ReLU (rectified linear unit), Sigmoid, and Tanh. These characteristics enable the network to approximate complex functions.

Weights and Bias:

Each connection between neurons in adjacent layers is assigned a weight, which determines the strength of the connection.

The bias is added to the weighted sum of the inputs to each neuron. They allow the network to represent functions that do not pass through the origin.

Here's some additional detail on feed-forward neural networks:

Training Process:

1. Forward Propagation:
   - Input data is fed into the input layer.
   - The inputs are multiplied by their corresponding weights and summed at each node in the hidden layer, along with the bias.
   - The summed value is passed through an activation function to produce the output of that node.
   - This process is repeated for each subsequent hidden layer, until the output layer is reached.
2. Backward Propagation (Backpropagation):

- The output of the network is compared to the expected output (ground truth), and an error is calculated.
- This error is propagated backwards through the network, and the weights are adjusted to minimize the error using an optimization algorithm like gradient descent.
- This process is repeated for many iterations (epochs) with different subsets of the training data (batches) until the network converges and the error is minimized.

Network Architecture:

- The number of hidden layers and the number of nodes in each hidden layer can vary depending on the complexity of the problem.
- Adding more hidden layers and nodes can increase the network's capacity to learn complex patterns, but can also lead to overfitting if not properly regularized.
- Techniques like dropout, early stopping, and weight regularization are used to prevent overfitting.

Applications:

- Feed-forward neural networks are widely used for various tasks, including image recognition, natural language processing, and regression problems.
- They form the backbone of many deep learning models, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), which are used for more specialized tasks.

While feed-forward neural networks are relatively simple in structure compared to more advanced architectures, they are powerful function approximators and have been instrumental in the development of deep learning and its widespread applications.

Practical Example in Cybersecurity:

Prorlem: Building an FFNN for detecting malicious network traffic.

Data:

source_ip	destination_ip	packet_count	bytes_transferred	label
0.096357	0.4109014675052411	1	0.637289	0
0.4876615746180964	0.8218029350104823	0.5815677966101696	0.5587031790997797	1
0.9870740305522914	0.38155136268343814	0.913136	0.7279404049942294	0
0.2937720329024677	0.17819706498951782	0.5858050847457628	0.9976917427342357	1
0.10105757931844887	0.9790356394129981	0.24364406779661016	0.064421	0
0.059929494712103404	0.2672955974842767	0.8665254237288136	0.8461861294722485	0
0.79906	0.14570230607966456	0.5201271186440678	0.3138180673591438	1
0	0.4591194968553459	0.12394067796610168	0.14940719756583778	1
0.6980023501762631	0.3060796645702306	0.4989406779661017	0.39681040814185287	0
0.11868390129259694	0	0.8527542372881356	0.7757842828664359	1
0.5240893066980022	0.24213836477987422	0.6705508474576272	0.3055293253593537	1
0.2279670975323149	0.7610062893081762	0.007415	0.090232	1
0.36427732079905994	0.8752620545073376	0.8760593220338984	0.38631832966110585	0
0.5146886016451233	0.5649895178197065	0.1620762711864407	0.82132	0
0.078731	0.4748427672955975	0.2754237288135593	1	0
0.4136310223266745	0.038784	0.3961864406779661	0.7042283076277411	0
0.092832	0.5125786163522014	0.6218220338983051	0.5785332074283915	0

	1	0.6918238993710693	0.3199152542372881		0	0
0.7555816686251469		0.4758909853249476		0	0.4339523659636974	0
0.12925969447708577		0.7106918238993711	0.24152542372881355		0.087504	0
0.7532314923619271			1	0.8082627118644068	0.7835484209421887	0
0.3384253819036428		0.7976939203354299	0.3516949152542373		0.9763928234183192	1
0.8801410105757931		0.17610062893081763	0.5836864406779662		0.089707	0
0.3795534665099883		0.9811320754716982	0.9364406779661018		0.8087294092959816	1
0.5534665099882491		0.6970649895178198	0.3453389830508474		0.6586926870212989	1

Input: Network traffic data (e.g., packets, flow features).
Output: Binary label indicating whether the traffic is malicious or benign.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from keras.models import Sequential
from keras.layers import Dense

# Load the data
data = pd.read_csv('networkdata.csv')
# Assuming your data has features and labels, adjust this according to your data
X = data.drop('label', axis=1).values  # Features
y = data['label'].values  # Labels
# Splitting 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)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Building the FFNN model
model = Sequential()
model.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))  # Input layer with 128 neurons and ReLU activation
model.add(Dense(64, activation='relu'))  # Hidden layer with 64 neurons and ReLU activation
model.add(Dense(1, activation='sigmoid'))  # Output layer with 1 neuron and Sigmoid activation
# Compiling the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Training the model
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)
```

```
# Making predictions
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)
# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```