Task 1.

A feed-forward neural network (FFNN) is a fundamental type of artificial neural network where connections between the nodes do not form cycles. It's called "feed-forward" because the data flows from the input layer through one or more hidden layers to the output layer without any feedback loops. Each layer in the network consists of nodes (neurons), and each node is connected to every node in the subsequent layer.

The following is the comprehensive breakdown of the components and functioning of a feed-forward neural network:

Input Layer: The input layer consists of neurons that receive the initial data. Each neuron represents a feature of the input data, and the number of neurons in this layer is determined by the dimensionality of the input data.

Hidden Layers: Between the input and output layers, there can be one or more hidden layers. These layers are responsible for learning complex patterns in the data. Each neuron in a hidden layer receives inputs from all neurons in the previous layer and produces an output using an activation function.

Output Layer: The output layer produces the final output of the network. The number of neurons in the output layer depends on the nature of the task. For example, for binary classification, there would be one neuron for each class, while for multi-class classification, there would be one neuron per class.

Weights and Biases: Each connection between neurons in adjacent layers has a weight associated with it. These weights determine the strength of the connection. Additionally, each neuron has a bias term which allows the network to capture non-linear relationships between inputs and outputs.

Activation Functions: Activation functions introduce non-linearities into the network, enabling it to learn complex mappings between inputs and outputs. Common activation functions include sigmoid, tanh, ReLU (Rectified Linear Unit), and softmax (for the output layer in classification tasks).

Forward Propagation: During the forward pass, the input data is passed through the network, and computations are performed layer by layer until the output is generated. Each neuron in a layer computes a weighted sum of its inputs, adds a bias term, and applies an activation function to produce the output.

Loss Function: The output of the network is compared with the actual target values using a loss function, which measures the difference between the predicted and actual values. Common loss functions include mean squared error for regression tasks and cross-entropy loss for classification tasks.

Backpropagation: Backpropagation is the process of updating the weights and biases of the network to minimize the loss function. It involves computing the gradients of the loss function with respect to the weights and biases using the chain rule and then adjusting the weights and biases using gradient descent or its variants.

Now, let's consider a practical example of applying a feed-forward neural network in cybersecurity, specifically for detecting network intrusions using synthetic cybersecurity data. We'll implement a simple FFNN using Python and TensorFlow/Keras. For this example we use the data given in the synthetic_cybersecurity_data.csv

protocol_type	service	flag	src_bytes	dst_bytes	label
tcp	dns	SF	41658	84356	0
tcp	http	S1	11827	42960	0
tcp	dns	REJ	76393	88720	0
udp	ftp	RSTR	25845	54441	0
icmp	dns	RSTR	28097	86161	1
icmp	ftp	SF	95998	20444	0
icmp	dns	RSTO	74737	48279	1
udp	http	S1	80488	85687	0
udp	http	RSTO	16746	90620	0
icmp	http	SF	56705	45707	0
udp	ftp	SF	68284	80481	0
icmp	http	RSTO	514	22042	0
tcp	ftp	RSTR	57196	77190	1
icmp	http	RSTR	91201	91727	0
udp	dns	REJ	4845	41842	0
udp	http	RSTR	73582	31815	0
udp	http	RSTO	73314	45768	0
icmp	ftp	REJ	88007	51196	1
udp	dns	S1	56988	21387	0

The following is the Python prediction code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

# Load the synthetic cybersecurity dataset
df = pd.read_csv('synthetic_cybersecurity_data.csv')

# Preprocessing
X = df.drop('label', axis=1)
y = df['label']

# One-hot encode categorical features
X = pd.get_dummies(X)

# Split data into train and test sets
```

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

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the FFNN model
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

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

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

# Predict probabilities
y_pred_probs = model.predict(X_test)

# Convert probabilities to binary predictions based on a threshold (e.g., 0.5)
y_pred = (y_pred_probs > 0.5).astype(int)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```