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

A Feed-forward Neural Network (FNN) is a type of artificial neural network wherein connections between the nodes do not form a cycle.
This kind of network architecture is one of the simplest forms of artificial neural networks and is particularly well-suited for approximating functions and classifying datasets with complex boundaries.
Here's a comprehensive breakdown of its key components and functionalities:

Structure and Layers:
Input Layer: This layer consists of input neurons that send data on to the next layer. The input layer directly receives the input features of the data and passes them on without any computation.
Hidden Layers: These are intermediate layers between input and output layers, containing neurons that perform computations and then transfer their outputs to the next layer. The actual number of hidden layers and the number of neurons in each layer can vary, impacting the network's complexity and ability to capture relationships in the data.

Output Layer: The final layer that produces the output of the network. For classification tasks, the output layer often uses a softmax function to output probabilities of the classes.

Activation Functions:

These functions are critical as they introduce non-linear properties to the network, which allow it to learn more complex patterns. Common activation functions include ReLU (Rectified Linear Unit), sigmoid, and tanh.
Forward Propagation:
This is the process by which inputs are passed through the network from the input layer to the output layer. Each neuron in a layer receives inputs, performs a dot product followed by an activation function, and passes the output to the next layer.
Backpropagation:
In order to learn, FNNs use a technique called backpropagation, where the output error (difference between predicted output and actual output) is propagated back through the network to update the weights. This updating is done using gradient descent or variations thereof to minimize the error. Learning Rate:
This is a hyperparameter that determines the step size at each iteration while moving toward a minimum of a loss function. It affects how quickly a network learns and converges to a low error.
Loss Functions:
The choice of loss function depends on the specific task (e.g., cross-entropy loss for

classification tasks). The loss function measures how well the network's predictions match the actual labels, and the network's goal is to minimize this loss.
Regularization Techniques:
Techniques such as L1 and L2 regularization are often used to prevent the network from overfitting, ensuring the model generalizes well to unseen data.
Advantages:
FNNs are straightforward to implement, capable of approximating any input-output relationship (given sufficient neurons and layers), and useful in a wide range of applications from regression to classification.
Limitations:

They can suffer from issues like vanishing or exploding gradients, especially with deep networks. They also might require a large number of neurons and layers to deal with complex data, leading to high computational costs.

In [4]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam

# Step 1: Load Data
df = pd.read_csv('urldata.csv', usecols=['url', 'result'])

# Step 2: Feature Extraction
# Convert URL text to features using TF-IDF
vectorizer = TfidfVectorizer(max_features=100)  # Limiting to the top 100 features
X = vectorizer.fit_transform(df['url']).toarray()
y = df['result']

# Step 3: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Step 4: Normalize the features (optional, typically not needed for TF-IDF)
# scaler = StandardScaler()
# X_train = scaler.fit_transform(X_train)
# X_test = scaler.transform(X_test)

# Step 5: Build the model
model = Sequential([
    Dense(64, input_dim=X_train.shape[1], activation='relu'),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

# Step 6: Compile the model
model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
```

```
# Step 7: Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32)

# Step 8: Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy*100:.2f}%')
```

```
C:\ProgramData\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:88: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequ
ential models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
11255/11255 ━━━━━━━━━━━━━━━━━━━━ 8s 557us/step - accuracy: 0.9892 - loss: 0.0393
Epoch 2/10
11255/11255 ━━━━━━━━━━━━━━━━━━━━ 6s 564us/step - accuracy: 0.9970 - loss: 0.0122
Epoch 3/10
11255/11255 ━━━━━━━━━━━━━━━━━━━━ 6s 563us/step - accuracy: 0.9973 - loss: 0.0114
Epoch 4/10
11255/11255 ━━━━━━━━━━━━━━━━━━━━ 6s 550us/step - accuracy: 0.9971 - loss: 0.0121
Epoch 5/10
11255/11255 ━━━━━━━━━━━━━━━━━━━━ 6s 555us/step - accuracy: 0.9972 - loss: 0.0114
Epoch 6/10
11255/11255 ━━━━━━━━━━━━━━━━━━━━ 6s 559us/step - accuracy: 0.9972 - loss: 0.0112
Epoch 7/10
11255/11255 ━━━━━━━━━━━━━━━━━━━━ 6s 553us/step - accuracy: 0.9974 - loss: 0.0106
Epoch 8/10
11255/11255 ━━━━━━━━━━━━━━━━━━━━ 6s 552us/step - accuracy: 0.9974 - loss: 0.0109
Epoch 9/10
11255/11255 ━━━━━━━━━━━━━━━━━━━━ 6s 555us/step - accuracy: 0.9974 - loss: 0.0107
Epoch 10/10
11255/11255 ━━━━━━━━━━━━━━━━━━━━ 6s 559us/step - accuracy: 0.9975 - loss: 0.0101
2814/2814 ━━━━━━━━━━━━━━━━━━━━ 1s 433us/step - accuracy: 0.9975 - loss: 0.0104
Test Accuracy: 99.75%
```

In [5]:

```
# Display the first few rows of the dataframe to understand what the data looks lik
print("First few rows of the dataset:")
print(df.head())

# Display a concise summary of the dataframe, including the types of index/columns,
print("\nDataset Info:")
print(df.info())

# Generate descriptive statistics that summarize the central tendency, dispersion,
print("\nDescriptive Statistics:")
print(df.describe())
```

```
First few rows of the dataset:
                         url  result
0     https://www.google.com       0
1    https://www.youtube.com       0
2   https://www.facebook.com       0
3      https://www.baidu.com       0
4  https://www.wikipedia.org       0

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 450176 entries, 0 to 450175
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype 
---  ------  --------------   ----- 
 0   url     450176 non-null  object
 1   result  450176 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 6.9+ MB
None

Descriptive Statistics:
              result
count  450176.000000
mean        0.231994
std         0.422105
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000
```

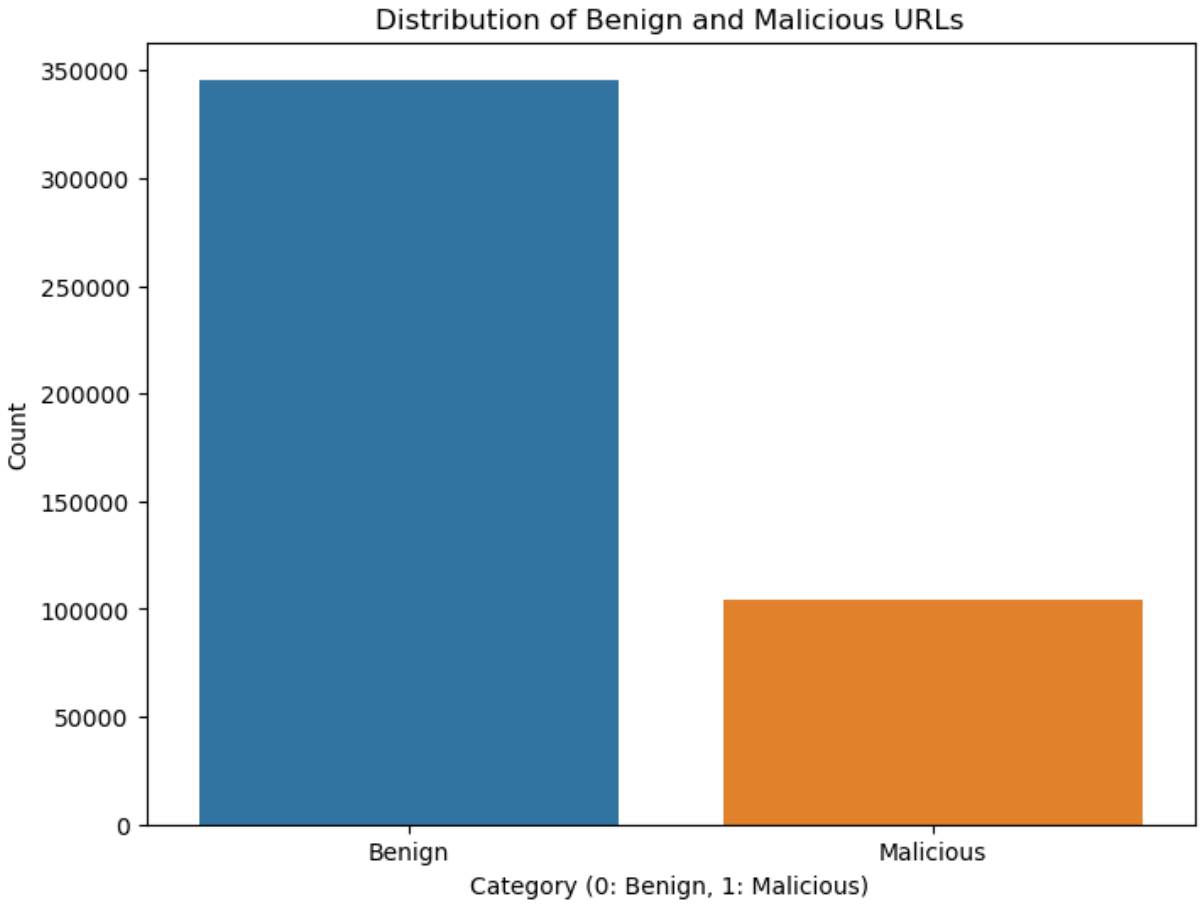
In [6]:
```python
# Print the first 10 rows of the dataframe
print("First 20 rows of the dataset:")
print(df.head(20))
```

```
First 20 rows of the dataset:
                          url  result
0      https://www.google.com       0
1     https://www.youtube.com       0
2    https://www.facebook.com       0
3       https://www.baidu.com       0
4   https://www.wikipedia.org       0
5      https://www.reddit.com       0
6       https://www.yahoo.com       0
7    https://www.google.co.in       0
8          https://www.qq.com       0
9      https://www.amazon.com       0
10     https://www.taobao.com       0
11    https://www.twitter.com       0
12      https://www.tmall.com       0
13   https://www.google.co.jp       0
14         https://www.vk.com       0
15       https://www.live.com       0
16  https://www.instagram.com       0
17       https://www.sohu.com       0
18    https://www.sina.com.cn       0
19         https://www.jd.com       0
```

In [8]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Plot the distribution of benign (0) and malicious (1) URLs
plt.figure(figsize=(8, 6))
sns.countplot(x='result', data=df)
plt.title('Distribution of Benign and Malicious URLs')
plt.xlabel('Category (0: Benign, 1: Malicious)')
plt.ylabel('Count')
plt.xticks([0, 1], ['Benign', 'Malicious'])
plt.show()
```


Distribution of Benign and Malicious URLs
350000
300000
250000
200000
150000
100000
50000
0
Count
Benign
Malicious
Category (0: Benign, 1: Malicious)

In [ ]: