A feed-forward neural network is a type of artificial neural network where connections between nodes do not form a cycle, but the information moves in only one direction.

Imagine a feed-forward neural network (FFNN) as a team of specialists working together to solve a problem. Each member of the team (neuron) has a specific role and expertise. They pass information from one to another, refining it until they achieve the desired outcome. Here's how they work:

1. Input Layer: Think of this as the team receiving raw data. Each member focuses on a particular aspect of the data, like its shape, color, or size.

2. Hidden Layers: These are the team's brainstorming sessions. Each member combines information from the previous layer, applying their own unique perspective. They adjust the data to find patterns and connections.

3. Output Layer: Finally, the team presents their findings. Based on their collective analysis, they give a verdict or prediction about the data.

Throughout this process, each team member adjusts their approach based on feedback, gradually improving their accuracy.

Python Code:

```python
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 classification_report
from keras.models import Sequential
from keras.layers import Dense

# Load the data from the CSV file
data = pd.read_csv("data.csv")

# Handle missing values (if necessary)
# data = data.dropna()  # Or use other methods to handle missing values

# Encode categorical variables using one-hot encoding
data = pd.get_dummies(data, columns=['protocol_type', 'service', 'flag'],
drop_first=True)

# Separate features and target variable
```

~~~python
X = data.drop(columns=['class'])
y = data['class']

# Split 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)

# Scale the features for better training
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Design the neural network architecture
model = Sequential([
   Dense(64, input_dim=X_train_scaled.shape[1], activation='relu'),
   Dense(32, activation='relu'),
   Dense(len(y_train.unique()), activation='softmax')  # Use softmax for multiclass
classification
])

# Compile the model specifying optimizer and loss function
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model on the training data
model.fit(X_train_scaled, pd.get_dummies(y_train), epochs=10, batch_size=32,
verbose=1)

# Make predictions on unseen testing data
y_pred = model.predict(X_test_scaled)
y_pred_labels = np.argmax(y_pred, axis=1)  # Convert probabilities to class labels

# Map predicted labels to original class names
class_names = ['attack', 'normal']
y_pred_mapped = [class_names[label] for label in y_pred_labels]

# Evaluate the model's performance
print(classification_report(y_test, y_pred_mapped))
~~~

```
```

Data:

```
duration,protocol_type,service,flag,src_bytes,dst_bytes,class
215,tcp,http,SF,184,0,normal
162,tcp,ftp,SF,2607,1977,normal
90,icmp,dns,SF,146,146,attack
171,udp,ssh,SF,146,146,normal
191,tcp,http,SF,697,528,normal
92,udp,dns,S0,146,0,normal
103,tcp,http,REJ,146,0,attack
82,tcp,ftp,S0,0,0,normal
183,tcp,ssh,REJ,105,0,attack
96,icmp,ftp,SF,111,25,attack
201,udp,http,S0,0,0,normal
121,icmp,ftp,RSTO,0,0,normal
136,tcp,ssh,REJ,146,0,attack
117,icmp,ssh,S0,0,0,normal
171,tcp,ftp,S0,0,0,normal
102,tcp,http,SF,485,289,normal
153,icmp,ftp,SF,146,146,attack
157,udp,http,S0,0,0,normal
112,icmp,dns,S0,0,0,normal
208,udp,ftp,SF,146,146,attack
131,tcp,ssh,SF,217,203,attack
82,tcp,ssh,REJ,0,0,normal
176,tcp,http,S0,0,0,normal
109,tcp,ftp,S0,0,0,normal
176,udp,http,SF,146,146,normal
141,tcp,http,S0,0,0,normal
98,icmp,ftp,RSTO,0,0,normal
160,udp,ftp,SF,146,146,attack
134,tcp,ssh,REJ,0,0,normal
105,tcp,http,S0,0,0,normal
194,tcp,ftp,SF,3232,1867,normal
98,icmp,ssh,RSTO,0,0,normal
162,tcp,http,REJ,0,0,normal
172,udp,ftp,SF,146,146,normal
134,icmp,ftp,SF,146,146,attack
173,tcp,http,SF,3621,1543,normal
125,udp,ssh,SF,146,146,attack
116,tcp,ftp,S0,0,0,normal
148,tcp,http,SF,244,257,normal
173,udp,http,S0,0,0,normal
136,tcp,ssh,SF,146,146,attack
```

```
169,icmp,ftp,SF,146,146,attack
153,tcp,http,SF,299,201,normal
170,tcp,http,S0,0,0,normal
99,tcp,ssh,REJ,0,0,normal
167,tcp,http,SF,586,315,normal
129,udp,ftp,SF,146,146,attack
175,icmp,ftp,SF,146,146,attack
124,udp,ssh,S0,0,0,normal
176,tcp,ftp,S0,0,0,normal
134,tcp,http,S0,0,0,normal
98,icmp,http,RSTO,0,0,normal
140,tcp,ftp,SF,303,304,normal
113,tcp,http,S0,0,0,normal
162,udp,http,S0,0,0,normal
111,tcp,ssh,SF,204,201,attack
139,tcp,ftp,S0,0,0,normal
173,icmp,http,S0,0,0,normal
128,udp,ftp,SF,146,146,attack
180,tcp,ssh,S0,0,0,normal
160,icmp,ftp,SF,146,146,attack
132,udp,http,S0,0,0,normal
104,tcp,ftp,S0,0,0,normal
95,tcp,http,SF,367,233,normal
187,udp,ftp,SF,146,146,attack
92,icmp,ftp,SF,146,146,attack
184,tcp,http,REJ,0,0,normal
132,udp,ssh,S0,0,0,normal
185,tcp,ftp,SF,2980,2243,normal
137,tcp,http,S0,0,0,normal
90,icmp,http,RSTO,0,0,normal
174,udp,http,S0,0,0,normal
148,icmp,ftp,S0,0,0,normal
95,udp,ssh,S0,0,0,normal
125,tcp,ftp,S0,0,0,normal
155,tcp,http,S0,0,0,normal
112,icmp,http,RSTO,0,0,normal
168,udp,ftp,SF,146,146,attack
145,tcp,http,S0,0,0,normal
99,tcp,ftp,SF,259,199,normal
127,icmp,ftp,S0,0,0,normal
176,udp,ssh,S0,0,0,normal
149,tcp,ftp,S0,0,0,normal
181,tcp,http,S0,0,0,normal
97,icmp,http,S0,0,0,normal
```

```
169,udp,ftp,SF,146,146,attack
146,tcp,http,SF,255,247,normal
91,icmp,ftp,S0,0,0,normal
180,udp,ssh,S0,0,0,normal
156,tcp,ftp,S0,0,0,normal
171,tcp,http,S0,0,0,normal
99,icmp,http,S0,0,0,normal
138,udp,ftp,SF,146,146,attack
150,tcp,http,S0,0,0,normal
183,tcp,ftp,SF,2327,1582,normal
```

Result:

```
Epoch 1/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.1310 - loss: 1.0752
Epoch 2/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 643us/step - accuracy: 0.1297 - loss: 0.9337
Epoch 3/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 612us/step - accuracy: 0.1963 - loss: 0.8280
Epoch 4/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 593us/step - accuracy: 0.4058 - loss: 0.7561
Epoch 5/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 689us/step - accuracy: 0.5944 - loss: 0.6637
Epoch 6/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 672us/step - accuracy: 0.7747 - loss: 0.6000
Epoch 7/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 625us/step - accuracy: 0.7514 - loss: 0.5463
Epoch 8/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 576us/step - accuracy: 0.7919 - loss: 0.4906
Epoch 9/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 709us/step - accuracy: 0.7998 - loss: 0.4466
Epoch 10/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 566us/step - accuracy: 0.8154 - loss: 0.4160
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
              precision    recall  f1-score   support

      attack       1.00      0.25      0.40         4
      normal       0.83      1.00      0.91        15

    accuracy                           0.84        19
   macro avg       0.92      0.62      0.65        19
weighted avg       0.87      0.84      0.80        19
```