# Feedforward Neural Network (FFNN) - Beginner's Guide

## 1. What is a Feedforward Neural Network (FFNN)?

A Feedforward Neural Network is a type of artificial neural network where information flows in one direction: from input layer to hidden layers to output layer. There are no cycles or feedback loops.

It's used in classification, regression, and many other tasks.

## 2. Structure of FFNN

- Input Layer: Takes the initial data (e.g., image pixels, numeric features).
- Hidden Layers: Neurons apply weighted sums followed by an activation function.
- Output Layer: Produces the final prediction.

Each layer is fully connected to the next - meaning every neuron in one layer is connected to every neuron in the next.

## 3. Math Behind a Layer

Each neuron performs the following computation:

$$z = W * x + b$$

$$a = \text{activation}(z)$$

Where:

- W: weights
- x: inputs from the previous layer
- b: bias term

- activation: a non-linear function (e.g., ReLU, Sigmoid)

## 4. Activation Functions

- ReLU: max(0, x) - most common for hidden layers

- Sigmoid: 1 / (1 + e^-x) - for binary classification output

- Tanh: maps input to range [-1, 1]

- Softmax: used for multi-class classification

- Linear: used in regression tasks

## 5. Training a FFNN

Steps:

1. Forward pass: calculate predictions from inputs

2. Compute loss: compare predictions with actual targets

3. Backpropagation: calculate gradients using loss

4. Update weights: use optimizer to apply gradients

Loss functions:

- Binary Cross Entropy: for binary classification

- Cross Entropy: for multi-class

- MSE (Mean Squared Error): for regression

## 6. PyTorch Example

# Feedforward Neural Network (FFNN) - Beginner's Guide

```
import torch

import torch.nn as nn

import torch.optim as optim


# Sample XOR dataset

X = torch.tensor([[0.,0.],[0.,1.],[1.,0.],[1.,1.]])

y = torch.tensor([[0.],[1.],[1.],[0.]])


# Define model

class FFNN(nn.Module):

    def __init__(self):

        super().__init__()

        self.hidden = nn.Linear(2, 4)

        self.output = nn.Linear(4, 1)

        self.activation = nn.ReLU()

        self.sigmoid = nn.Sigmoid()


    def forward(self, x):

        x = self.activation(self.hidden(x))

        x = self.sigmoid(self.output(x))

        return x


model = FFNN()

criterion = nn.BCELoss()

optimizer = optim.Adam(model.parameters(), lr=0.01)
```

```
for epoch in range(1000):
    outputs = model(X)
    loss = criterion(outputs, y)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

## 7. Summary

FFNNs are basic yet powerful networks. Their structure is simple - fully connected layers with non-linear activations.

They are suitable for a wide variety of machine learning tasks, especially when data can be treated in a tabular or static form.