# Neural Networks and Deep Learning - Assignment 1

Neta Ben Mordechai
Itay Chabra

## Project Overview

In this project, we have built a simple feedforward neural network using PyTorch.
The network consists of a custom Linear layer, which implements a basic linear transformation, and a BTU layer that applies the sigmoid activation function with a temperature parameter.

The NeuronNet class is used to define the structure of the neural network, where the number of neurons in the hidden layer (k) and whether a bypass mechanism is used can be customized.
The forward function processes input through the hidden layer, applies the BTU function, and computes the final output.
The set_weights function allows manual adjustment of weights for both the hidden and output layers.

The loss is computed using the sum of squared errors between the predicted output and the expected output.

In tests, the network is evaluated using the XOR dataset with different configurations of hidden layer neurons (k = 1, 2, 4) and bypass options (True/False). For each configuration, weights and biases for both the hidden and output layers are manually set.
The network's performance is assessed by computing the output truth table and calculating the loss (sum of squared errors).
The variations in configurations help observe the network's behavior with different complexity levels and architecture choices, providing insights into how these changes impact the model's ability to approximate the XOR function.

## Result Analysis

- **Input matrix** - represents all possible combinations of two binary inputs for the XOR problem.
  tensor([[0., 0.], [0., 1.], [1., 0.], [1., 1.]]).
- **Expected output** - defines the XOR truth table.
  tensor([[0.], [1.], [1.], [0.]]).

Configuration: k = 1, bypass = True:

- **Hidden layer weights and biases** - Hidden Layer Weights (w): [[1. 1.]] , Hidden Layer Biases (b): [-1.5].
- **Output layer weights and biases** - Output Layer Weights (w): [[ 1.  1. -2.]] , Output Layer Biases (b): [-0.5].
- **Output matrix** - Matches the expected XOR truth table exactly, achieving perfect predictions.
  Output Matrix (truth table):
  Input: tensor([0., 0.]) => Output: 0.0000
  Input: tensor([0., 1.]) => Output: 1.0000
  Input: tensor([1., 0.]) => Output: 1.0000
  Input: tensor([1., 1.]) => Output: 0.0000
- **Loss** - 0.0, signifying no errors in predictions.

**Interpretation:** With only one hidden neuron and the bypass mechanism, the network effectively learns XOR by leveraging direct input contributions alongside the hidden layer output.

Configuration: k = 2, bypass = False:

- **Hidden layer weights and biases** - Hidden Layer Weights (w): [[ 1.  1.] [-1. -1.]] , Hidden Layer Biases (b): [-0.5  1.5].
- **Output layer weights and biases** - Output Layer Weights (w): [[1. 1.]] , Output Layer Biases (b): [-1.5].
- **Output matrix** - Matches the expected XOR truth table exactly, achieving perfect predictions.
  Output Matrix (truth table):
  Input: tensor([0., 0.]) => Output: 0.0000
  Input: tensor([0., 1.]) => Output: 1.0000
  Input: tensor([1., 0.]) => Output: 1.0000
  Input: tensor([1., 1.]) => Output: 0.0000
- **Loss** - 0.0, signifying no errors in predictions.

**Interpretation:** The absence of a bypass forces the network to rely entirely on the hidden layer, demonstrating that two neurons are sufficient to solve XOR without additional input connections.

Configuration: k = 4, bypass = False:

- **Hidden layer weights and biases** - Hidden Layer Weights (w): [[-1. -1.] [-1.  1.] [ 1. -1.] [ 1.  1.]] , Hidden Layer Biases (b): [ 0.5 -0.5 -0.5 -1.5].
- **Output layer weights and biases** - Output Layer Weights (w): [[0. 1. 1. 0.]] , Output Layer Biases (b): [-0.5].
- **Output matrix** - Matches the expected XOR truth table exactly, achieving perfect predictions.
  Output Matrix (truth table):
  Input: tensor([0., 0.]) => Output: 0.0000
  Input: tensor([0., 1.]) => Output: 1.0000
  Input: tensor([1., 0.]) => Output: 1.0000
  Input: tensor([1., 1.]) => Output: 0.0000
- **Loss** - 0.0, signifying no errors in predictions.

**Interpretation:** The larger hidden layer introduces redundancy, showcasing the network's capacity to model more complex problems while still solving XOR efficiently.

## Conclusions

The experiments demonstrate that the XOR problem can be effectively solved using a feedforward neural network with minimal configurations. The bypass mechanism allows simpler architectures to learn faster, while larger hidden layers introduce redundancy that may be beneficial for more complex tasks.