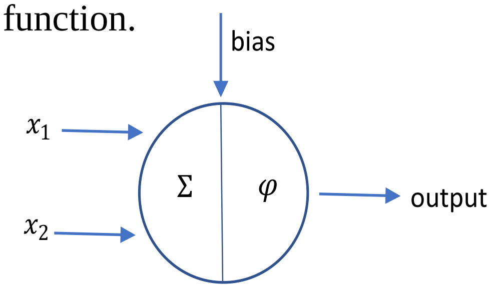
Lecture 3

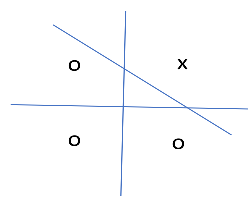
In previous lectures we talk about single unit neural network "single perceptron" where:

1. Can represent linear function.
2. Its form:



Limitation of single perceptron:

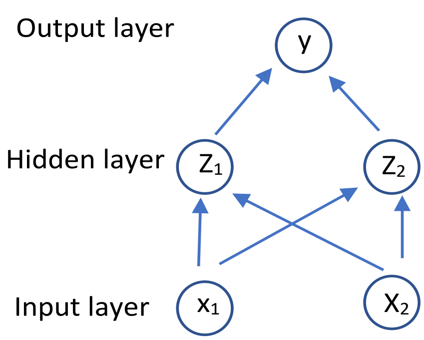
1. Monotonicity property → The activation can increase as corresponding input value, where the link has positive value.
2. Input interaction can cancel one another effect.
3. Represent only linearly separable functions, like:



Note:

We could represent **non-linear** function with neural network by stacking perceptron layers into different architecture, so we use **multilayer neural network**.

Multilayer form →



Our neural network has 2 layers
- **1 hidden layer**
- **1 output layer**

Note:

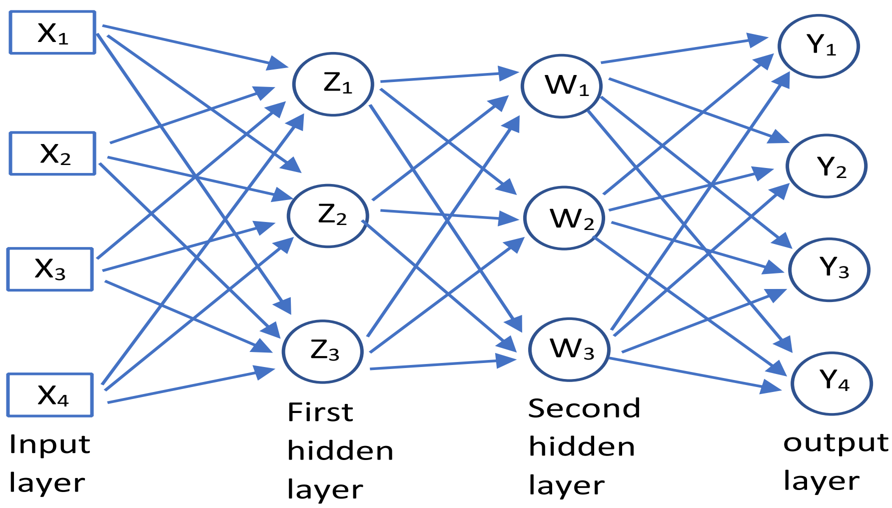
The hidden layer named like this because they are not observed in training example.

Multilayer NN advantages:

1. Represent interaction among inputs.
2. Can represent any Boolean and continuous functions as the number of hidden units is sufficient.

Note:

Representation exists to represent function does not mean that you can learn it well, but NN learning algorithm exist with weaker guarantee general structure of multilayer network.



Our neural network has 3 layers → **2 hidden layer**, **1 output layer**

Why has it called free forward?

1. It transfers from input layer to hidden layer to output layer.
2. All edges are single directional going forward from input to output.

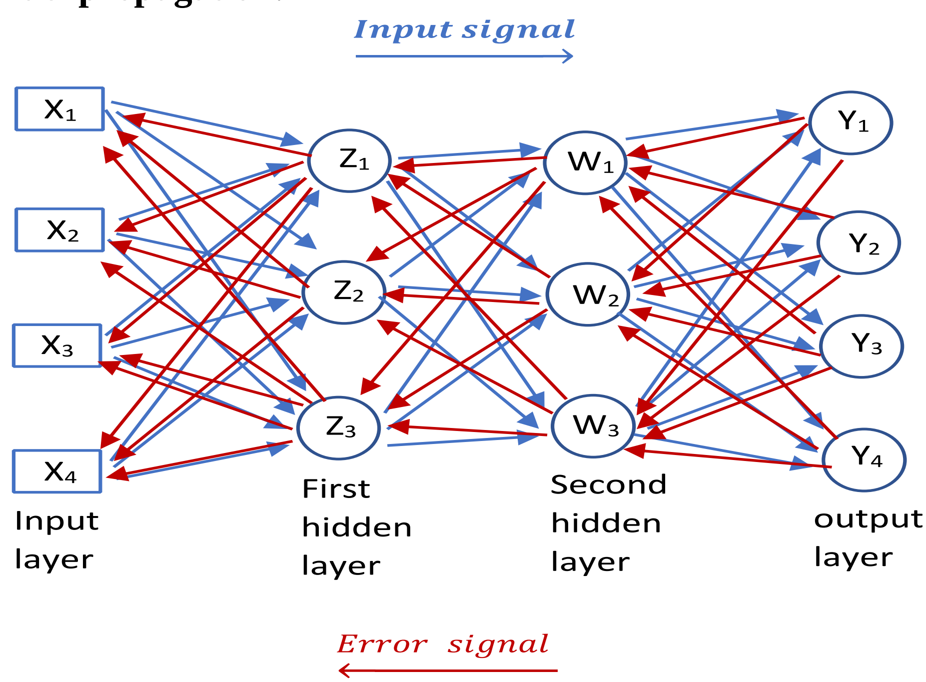
Note:

- In single perceptron the small error occurs when we change connection weights where the training perceptron based on output layer error.
- In multilayer neural network we change in hidden layer weights because we don't know the optimal output "target".

When we change hidden layer weights the error propagates from:

Output layer→Second hidden layer→First hidden layer→Input layer

That called Backpropagation:



Its algorithm:

1. **Initialization** → set all weights and threshold levels randomly uniformly in small range.
2. **Calculate forward computation:**
   a) **Apply input vector 'x'.**
   b) **Compute activation vector 'z' in hidden layer** → $z_j = \varphi(\sum_i v_{ij} x_i)$ "v is weights from input to hidden layer".
   c) **Compute output vector 'y'** → $y_k = \varphi(\sum_j w_{jk} z_j)$ "w is weights from hidden to output layer".
3. **Propagate error by updating weights from output to hidden layer by "delta rule".**

The back forward code:

https://github.com/Abdel-rahim/preceptron_test/blob/main/neuralnetwork.py