Machine Learning for Robotics: **Fully Connected Neural Networks**

Prof. Navid Dadkhah Tehrani

WPI

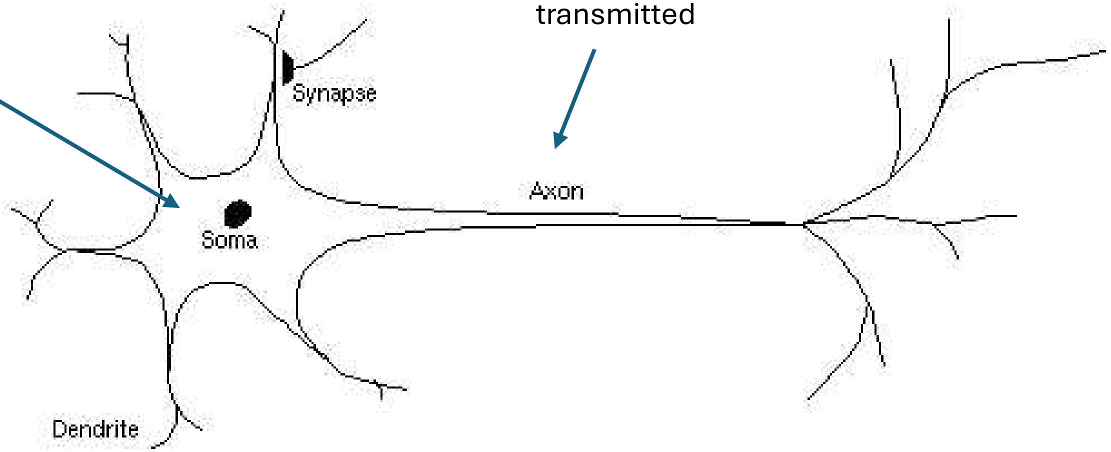
Perceptron:

Mathematical model of biological neuron. It can solve AND, OR, and NOT problem.

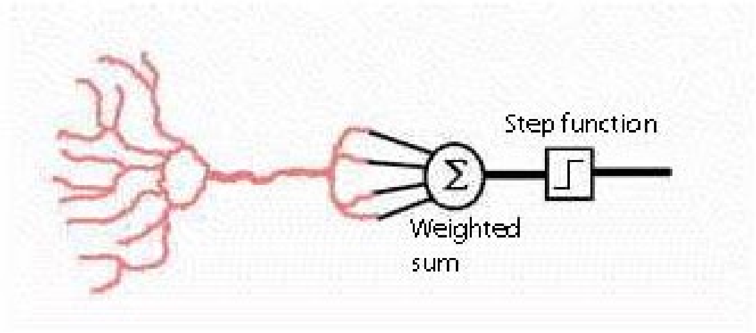
Cell core

Where signal get transmitted

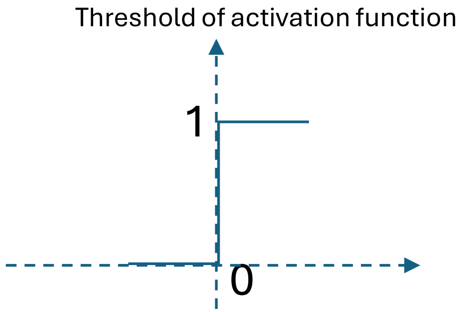
Where the neurons are connected to



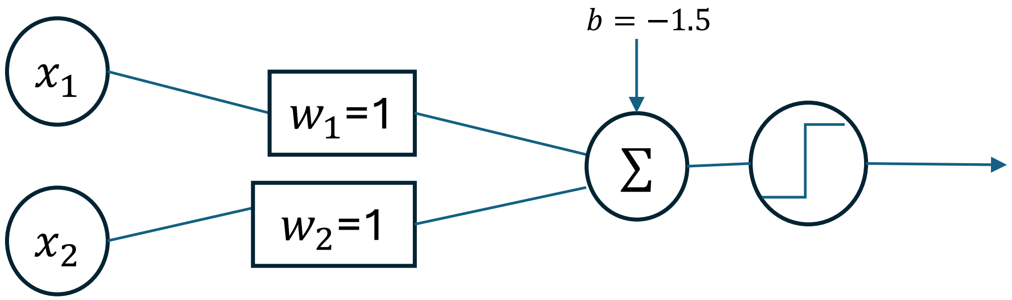
Mathematical model:



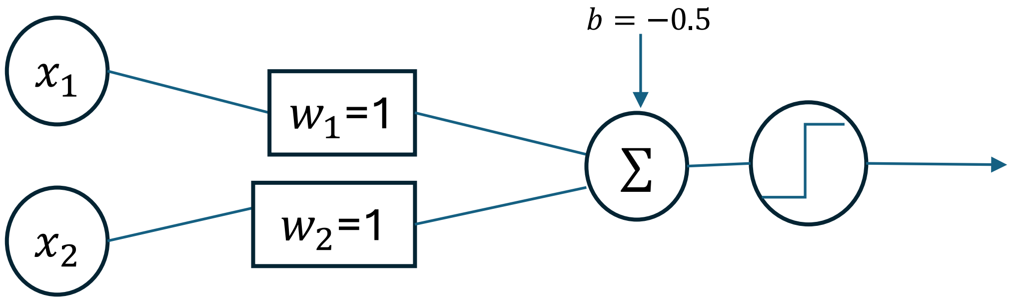
How to generate logical AND, OR for binary variables with perceptron:
(McCulloch and Pitts, 1943)



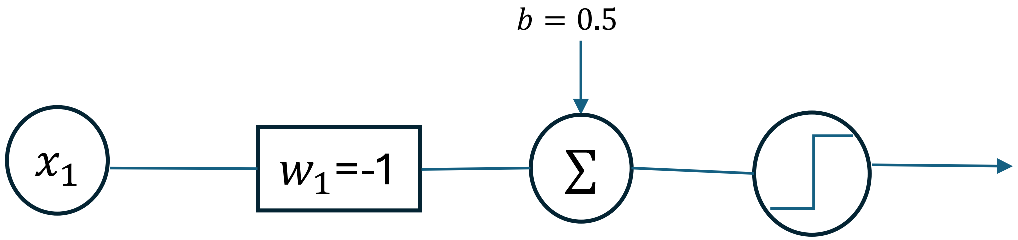
AND:



OR:



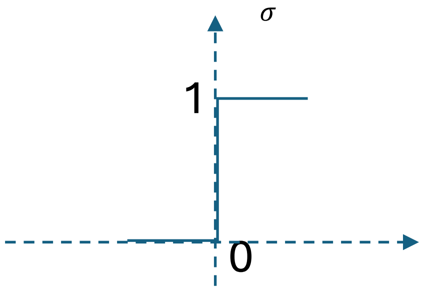
NOT:



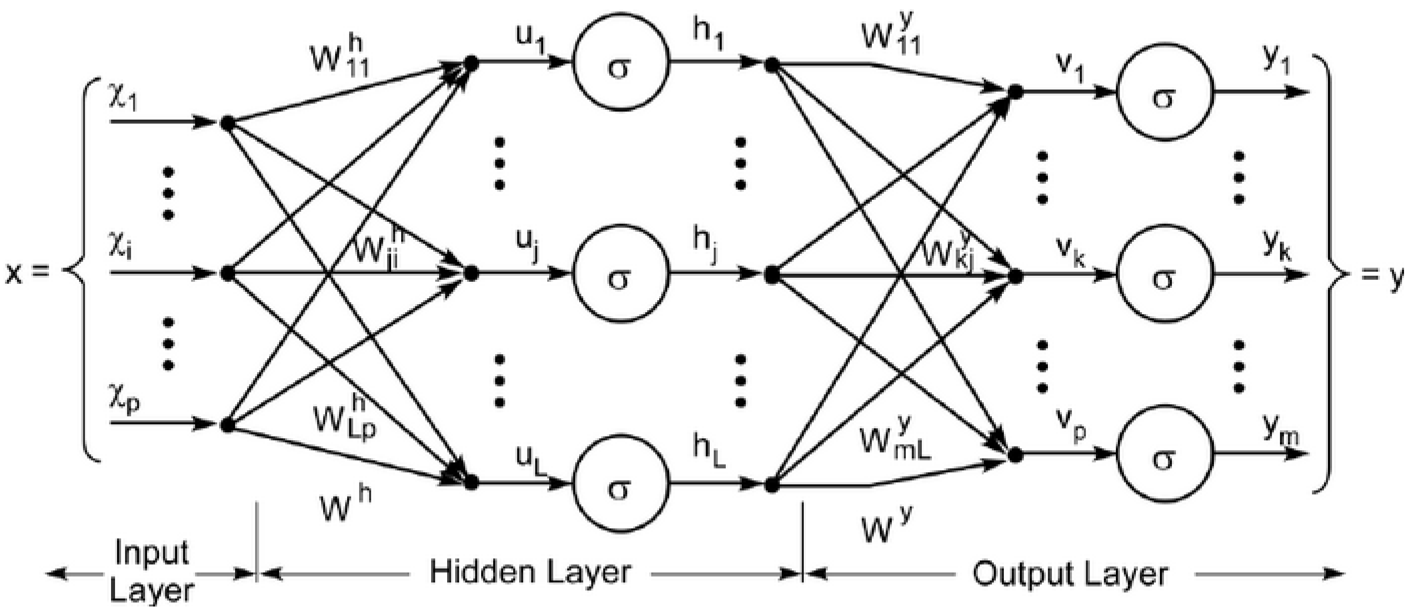
In general, the value of w_1, w_2, and b are learned from the data via training.

$$\hat{y} = \sigma\left(\sum_{i=1}^{m} x_i w_i + b\right)$$
$$= \sigma(x^T w + b)$$

$\hat{y}$: predicted output
σ: activation function



We can now extend this concept to a fully connected layers (or Multi-layer perceptron -MLP), where we have more than one neuron and more than one layer.



$$h = \sigma\left(W^h x + b^h\right)$$

$$W \in R^{L\times P}$$
$$x \in R^{P\times 1}$$
$$h \in R^{L\times 1}$$
$$b \in R^{L\times 1}$$

If we take batch on size n as input:

$$h = \sigma\left(W^h x + b^h\right)$$

$$W \in R^{L\times P}$$
$$x \in R^{P\times n}$$
$$h \in R^{L\times n}$$
$$b \in R^{L}$$

In essence, this is a linear transformation of input and then passing it through a non-linear activation function.

The output can be calculated similarly:

$$y = \sigma(W^y h + b^y)$$

Q: can we just apply the non-activation function at the last layer?

A: in general we can have as many layers as we want. If we remove the non-linear activation function the resulting neural network is equivalent to having only one layer:

$$y = \sigma(W^y h + b^y) = \sigma\left(W^y (W^h x + b^h) + b^y\right) = \sigma(Wx + b)$$

The non-linear activation function is what enables the neural network to have many layers and learns richer information.

The only issue is that the notation $y = \sigma(Wx + b)$ is mathematically correct but can be confusing when dealing we NN with many layers; Because when we draw the NN, x enters from left but in the math enters from right.

That's why Pytorch always uses this notation:

$$h = \sigma\left(xW^h + b^h\right)$$

$$W \in R^{P\times L}$$
$$x \in R^{1\times P}$$
$$h \in R^{1\times L}$$
$$b \in R^{1\times L}$$

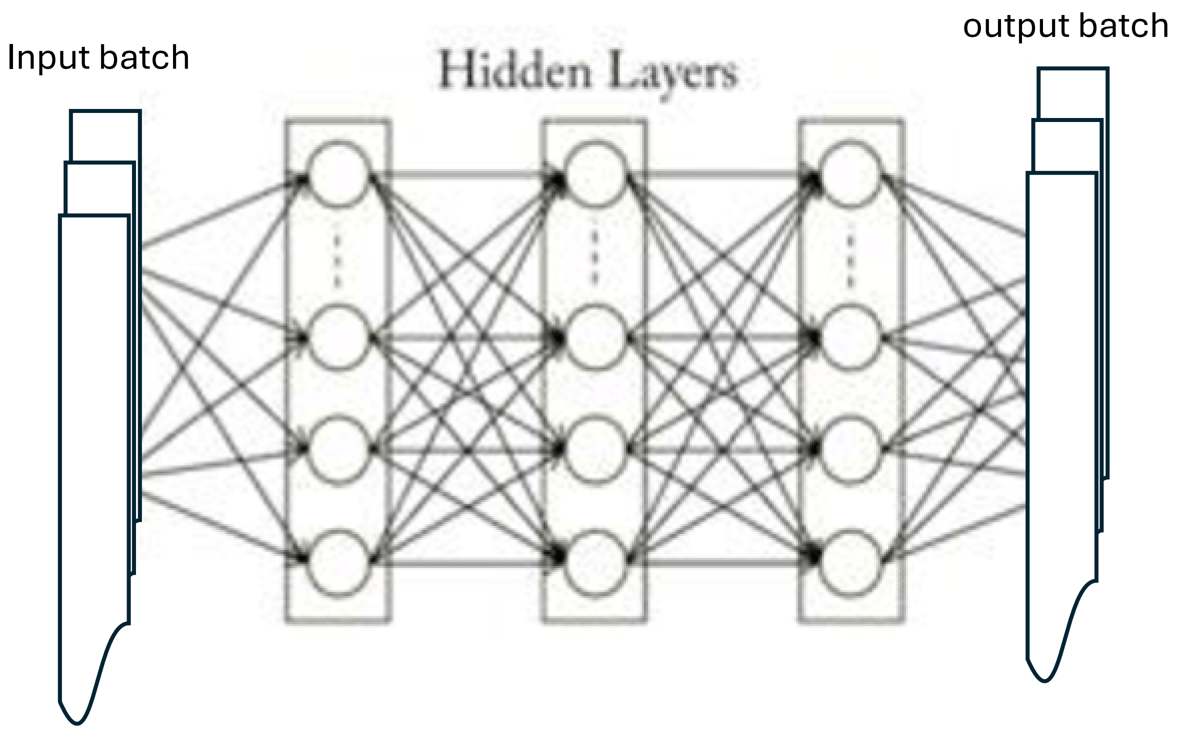
If we take batch of size n as input:

$$h = \sigma\left(xW^h + b^h\right)$$

$$W \in R^{P\times L}$$
$$x \in R^{n\times P}$$
$$h \in R^{n\times L}$$
$$b \in R^{1\times L}$$

In other words, in Pytorch the training examples as stacked in the rows on x.

Note that pytroch can process a batch of inputs in parallel and we don’t need to give inputs one by one.
For example, if we give a neural network that classifies images, you don’t have to pass 1 image at a time to get classification. You can pass many images and get classification for all of them.

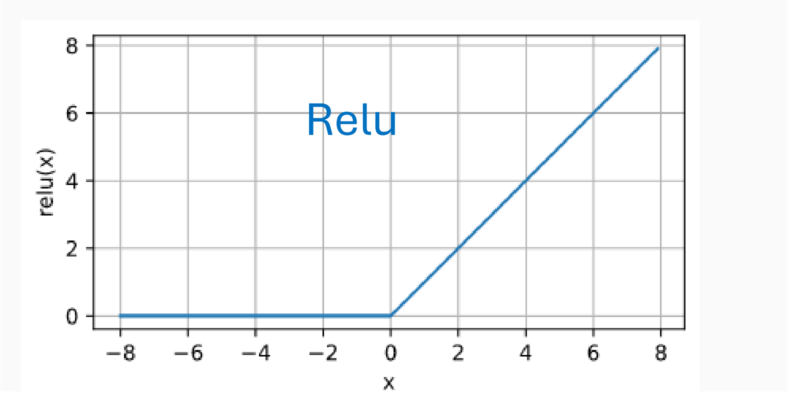


Q: how big the batch can be?
A: we are limited by the size of our GPU memory. But also larger batch is not necessarily better (will discuss in future).

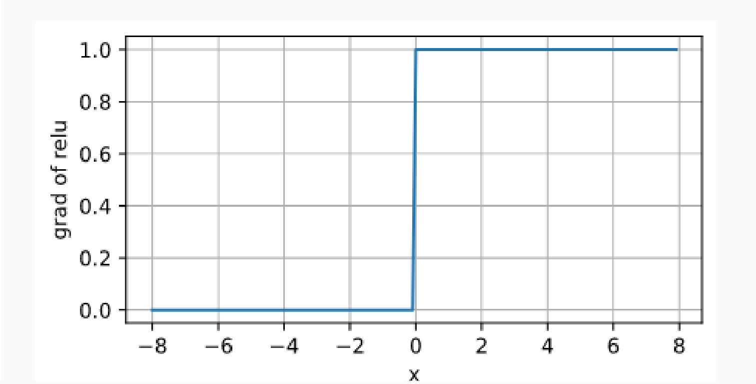
Different Activation Functions

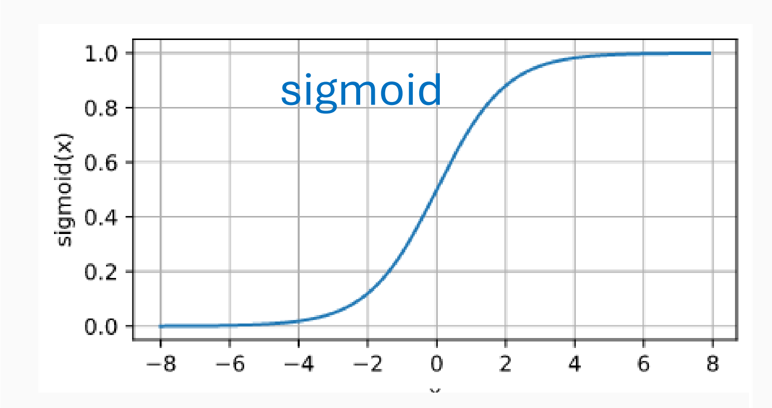
Math equation

gradient

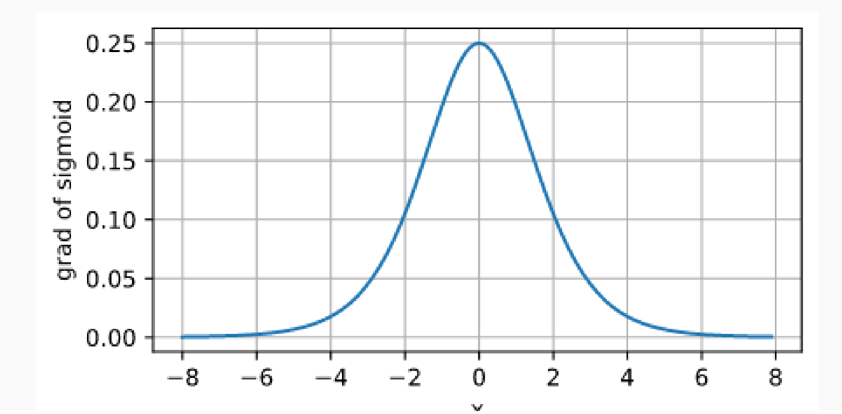


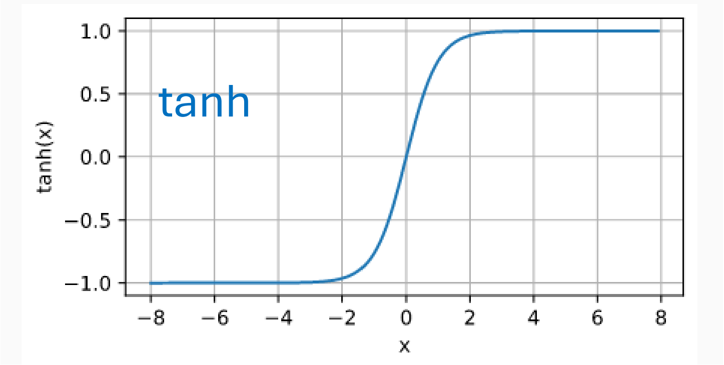
$$\mathrm{ReLU}(x) = \max(x, 0).$$





$$\mathrm{sigmoid}(x) = \frac{1}{1 + \exp(-x)}.$$





$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}.$$