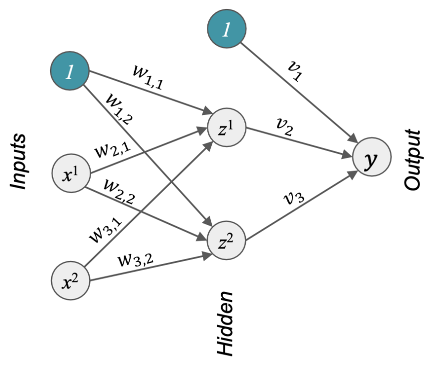
Week 8 - Neural networks and deep learning

- Neural networks
    - Multi-layer perceptron (MLP)
        - Lots of neuron units connected together into a directed acyclic graph
        - A feed-forward neural network
        - Fully connected layer: if all input units are connected with all output units
        - Fully connected networks: (every node in each layer connected to every node in the previous layer) means rapid growth in the number of weights

Neural networks: deep learning

- Example of a multilayer neural network

What is this '1'???



$$z^1 = f\left(w_{1,1}1 + w_{2,1}x^1 + w_{3,1}x^2\right)$$

$$z^2 = f\left(w_{1,2}1 + w_{2,2}x^1 + w_{3,2}x^2\right)$$

$$W^T = \begin{bmatrix} w_{1,1} & w_{2,1} & w_{3,1} \\ w_{1,2} & w_{2,2} & w_{3,2} \end{bmatrix}$$

$$z = f(W^T x)$$

$$y = f(v_1 1 + v_2 z^1 + v_3 z^2)$$

$$y = f(v^T z)$$

The constant nodes "1" are essentially not inputs so they have been moved out of the way of the inputs for clarity

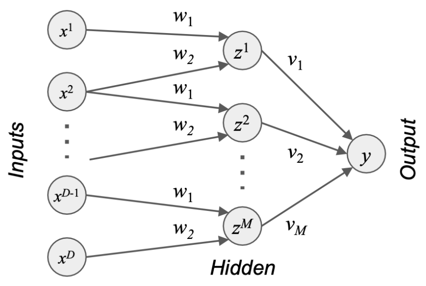
It represents bias

- **Weight-sharing in fully connected networks**

#Code

Neural networks: weights consideration

- **Fully connected networks** (every node in each layer connected to every node in the previous layer) means rapid growth in the number of weights
- **Weight-sharing**, forcing certain connections between nodes to have the same weight, is sensible for certain special applications
- Widely-used example (particularly suited to ordered data: images or time series) is **convolutional** sharing



$$z^1 = \max(0,\ w^T[x^1\ x^2]^T)$$
$$z^2 = \max(0,\ w^T[x^2\ x^3]^T)$$
$$\dots$$
$$z^M = \max(0,\ w^T[x^{D-1}\ x^D]^T)$$
$$y = \max(0,\ v^T z)$$

- In machine learning, the sign function is a mathematical function that maps the input to a specific output based on its sign. It is commonly used in binary classification problems or as an activation function in neural networks. The sign function is defined as follows:

```
sign(x) = -1 if w^t * x < 0
           0 if w^t * x = 0
           1 if w^t * x > 0
```

- Activation functions

Activation function	Expression	Derivative	Expression
ReLU (rectified linear unit)	$\max(0, x)$	Step function	$\mathbb{I}[x \geq 0]$
Softplus	$\ln(1 + e^x)$	Logistic (sigmoid)	$\frac{1}{1+e^{-x}}$
Hyperbolic tangent	$\tanh(x)$	Hyperbolic tangent gradient	$1 - \tanh(x)^2$

- ReLU

D

- Soft ReLU

$$y = log 1 + e^x$$

- Softplus

$$ln(1 + e^x)$$

- Hard Threshold

$$\begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

- Logistic

$$y = \frac{1}{1 + e^{-x}}$$

- Hyperbolic Tangent (tanh)

$$tanh(x) = y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

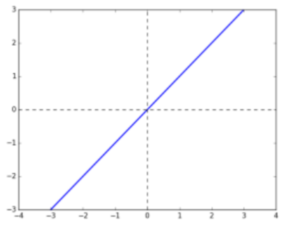
- tanh derivative (Hyperbolic Tangent Gradient)

$$1 - tanh(x)^2$$

- Step

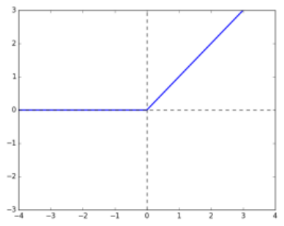
$$\mathbb{I}[x \geq 0]$$

Some activation functions:



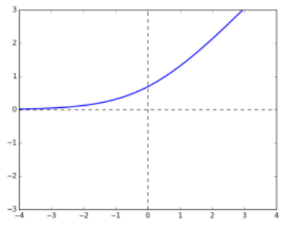
Linear

$$y = z$$



Rectified Linear Unit (ReLU)

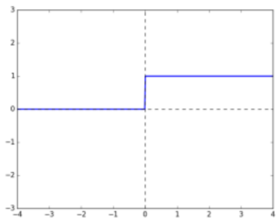
$$y = \max(0, z)$$



Soft ReLU

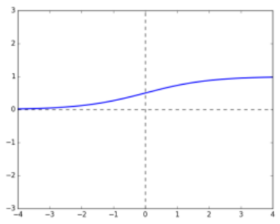
$$y = \log 1 + e^z$$

Some activation functions:



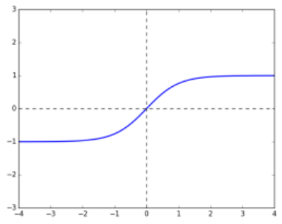
Hard Threshold

$$y = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$



Logistic

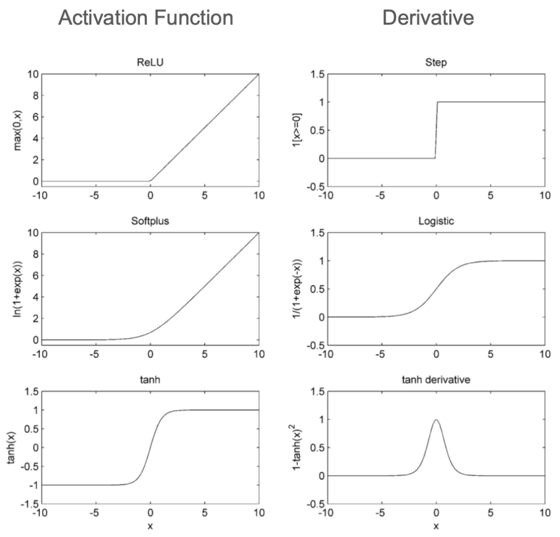
$$y = \frac{1}{1 + e^{-z}}$$



Hyperbolic Tangent (tanh)

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Activation nonlinearities



- Wide range of activation functions in use (**logistic**, **tanh**, **softplus**, **ReLU**): only criteria is that they must be **nonlinear** and should ideally be **differentiable** (almost everywhere)
- ReLU is perceptron loss, sigmoid is logistic regression loss
- ReLU most widely used activation; exactly zero for half of its input range (many outputs will be zero)

- Deep Neural Logic Networks

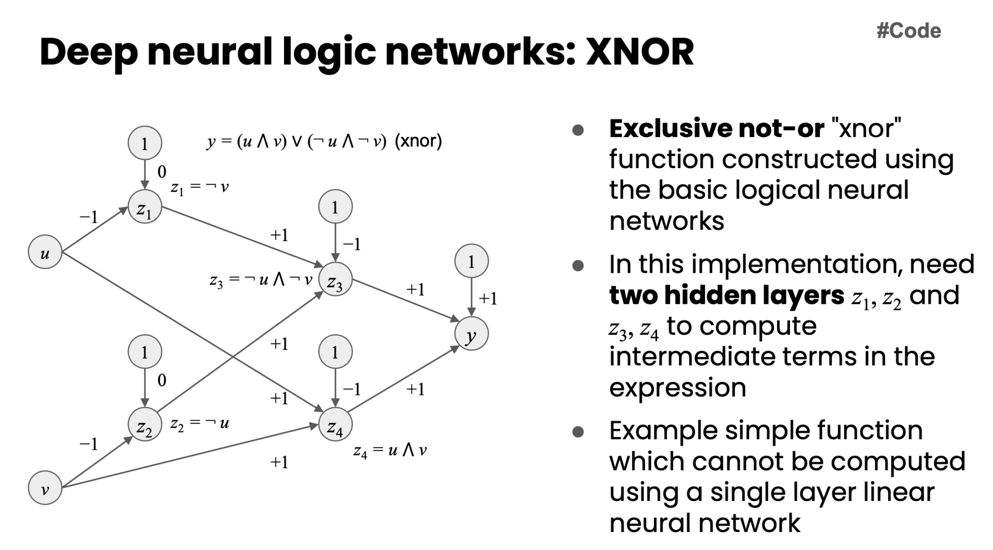
True. We will construct a system of logical computation based on the use of the neural network function $f_b(w, x) = \text{sign}(w_0 + w_1x^1 + w_2x^2)$[3] for the binary operators 'and' and 'or', and for the 'not' operator we will use the single input neural network function $f_u(w, x) = \text{sign}(w_0 + w_1x^1)$. For the 'and' function, under this encoding, $w_{\text{and}} = [-1, 1, 1]$ behaves as required. Similarly, for the 'or' function, weights $w_{\text{or}} = [1, 1, 1]$ work, and for the 'not' function, $w_{\text{not}} = [0, -1]$ suffices.[4] So, our single-layer logical operator neurons are given by the following very simple functions,

$$\begin{aligned} f_{\text{and}}\left(x^1, x^2\right) &= \text{sign}\left(x^1 + x^2 - 1\right) \\ f_{\text{or}}\left(x^1, x^2\right) &= \text{sign}\left(x^1 + x^2 + 1\right) \\ f_{\text{not}}(x) &= \text{sign}(-x). \end{aligned} \tag{13.4}$$

xnor: 1 if two values are the same, 0 if they are different. $y = (T \wedge T) \vee (F \wedge F) = T \vee T = 1,\ y = (T \wedge F) \vee (F \wedge T) = F \vee F = 0$

xor: 0 if two values are the same, 1 if they are different

  - XNOR

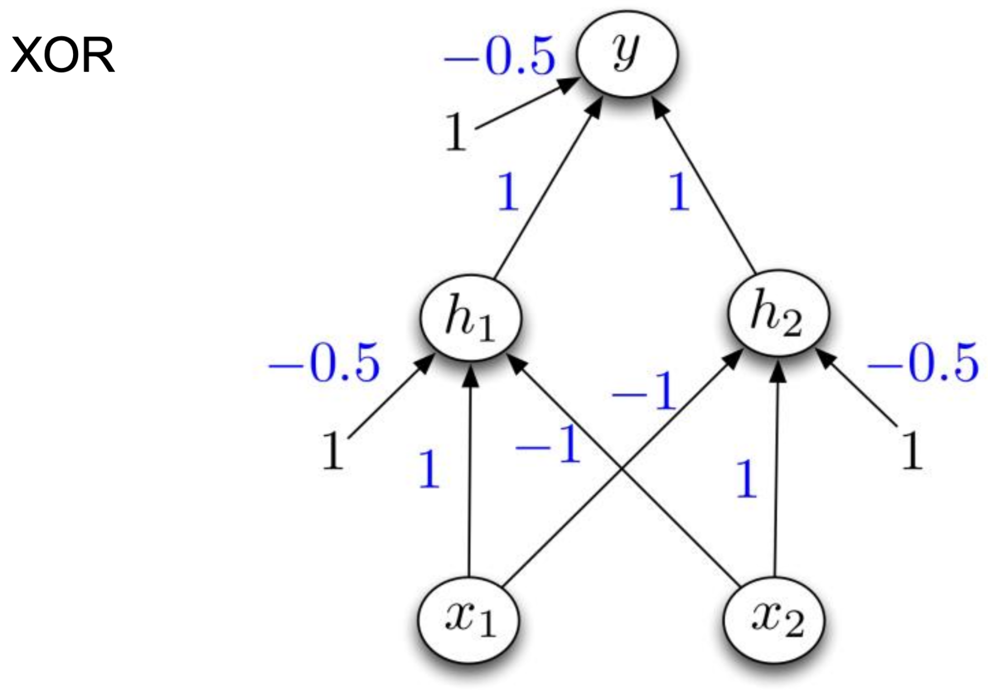

$y = (u \land v) \lor (\neg u \land \neg v)$ (xnor)
$z_1 = \neg v$
$z_2 = \neg u$
$z_3 = \neg u \land \neg v$
$z_4 = u \land v$
u
v
y
1
0
−1
+1

Deep neural logic networks: XNOR

#Code

- **Exclusive not-or** "xnor" function constructed using the basic logical neural networks
- In this implementation, need **two hidden layers** z_1, z_2 and z_3, z_4 to compute intermediate terms in the expression
- Example simple function which cannot be computed using a single layer linear neural network

- XOR



- **A convolutional neural network (CNN) has** weights **shared** between connections.
  - well-suited to ordered data such as **images** and **time series**