Attendance Code:
xxxxx

Artificial Intelligence and Machine Learning (AIML)

2023–24

#Code

- **Last ML lectures**:
  - **Unsupervised learning**: clustering and the k-means algorithm
  - **Supervised learning**:
    - Linear regression using SGD
    - Classification using the perceptron algorithm
- **This lecture**: neural networks and deep learning

AIML

#Code

What we have accomplished so far

- Supervised Learning

Labeled training data

Data point (i)	Feature 1 (x^1)	Feature 2 (x^2)	…	Feature D (x^D)	Output (y)
1					
2					
3					
4					
…					
…					
N					

Linear models for regression/classification

#Code

What we have accomplished so far

- Supervised Learning

Labeled training data

Data point (i)	Feature 1 (x^1)	Feature 2 (x^2)	…	Feature D (x^D)	Output (y)
1					
2					
3					
4					
…					
…					
N					

Linear models for regression/classification

$$f(w, x) = \hat{y} = f\left(\sum_{j=1}^{D} w_j x^j\right)$$

#Code

What we have accomplished so far

- Supervised Learning

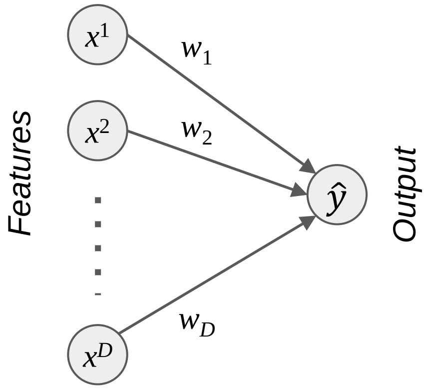
Labeled training data

Data point (i)	Feature 1 (x^1)	Feature 2 (x^2)	…	Feature D (x^D)	Output (y)
1					
2					
3					
4					
…					
…					
N					

Linear models for regression/classification

$$f(w, x) = \hat{y} = f\left(\sum_{j=1}^{D} w_j x^j\right)$$

Pictorial Representation



$$f(w, x) = \hat{y} = f(w_1 x^1 + w_2 x^2 + \cdots + w_D x^D)$$

What we have accomplished so far

- Supervised Learning

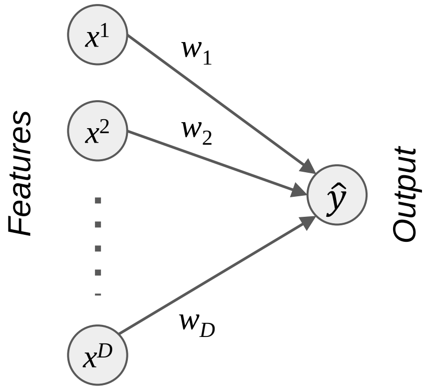
Labeled training data

Data point (i)	Feature 1 (x^1)	Feature 2 (x^2)	…	Feature D (x^D)	Output (y)
1					
2					
3					
4					
…					
…					
N					

Linear models for
regression/classification

$$f(w,x) = \hat{y} = f\left(\sum_{j=1}^{D} w_j x^j\right)$$

Pictorial Representation



$$f(w,x) = \hat{y} = f(w_1 x^1 + w_2 x^2 + \cdots + w_D x^D)$$

$$\hat{y} = f(w^T x)$$

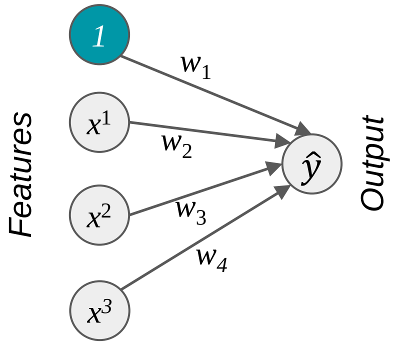
parameters vector

features vector

#Code

Classification Intuition: Housing Market

- Data on 3 features: square footage, # bedrooms, house age
- Classification label: location (UK vs. non-UK house)



$$f(w, x) = \hat{y} = \mathrm{sign}(w_1 1 + w_2 x^1 + w_3 x^2 + w_4 x^3)$$

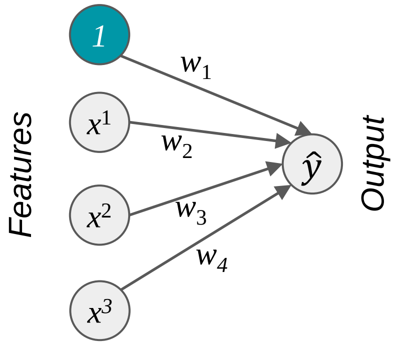
$$\hat{y} = \mathrm{sign}(w^T x)$$

AIML

#Code

How can we extend this to 3 classes?

- Data on 3 features: square footage, # bedrooms, house age
- Classification label: location (UK vs. non-UK house)

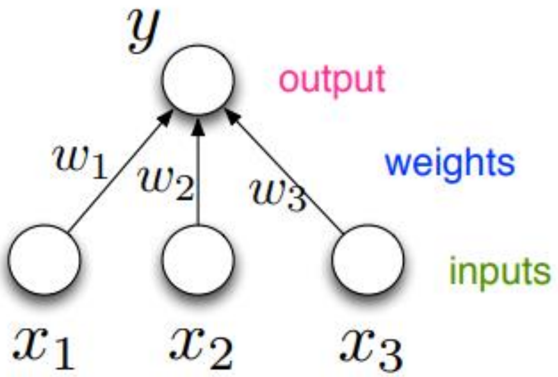


- Extra data labeled as Dubai houses
- How would you build a classification model to classify between the UK, Dubai, and other locations using the same features?

$$f(w, x) = \hat{y} = \text{sign}(w_1 1 + w_2 x^1 + w_3 x^2 + w_4 x^3)$$

$$\hat{y} = \text{sign}(w^T x)$$

- Recall the simple neuron-like unit:



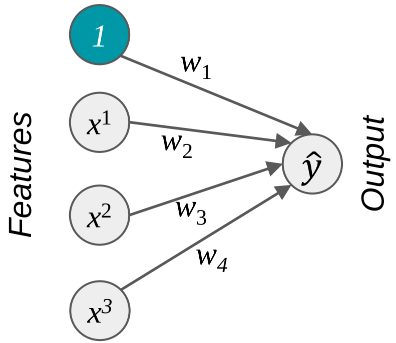
$$\overset{\text{output}}{y} = \underset{\text{nonlinearity}}{g}\left(\overset{\text{bias}}{b} + \sum_i \underset{\text{i'th input}}{x_i}\,\overset{\text{i'th weight}}{w_i}\right)$$

- These units are much more powerful if we connect many of them into a neural network.

How can we extend this to 3 classes?

#Code

- Data on 3 features: square footage, # bedrooms, house age
- Classification label: location (UK vs. non-UK house)



- Extra data labeled as Dubai houses
- How would you build a classification model to classify between the UK, Dubai, and other locations using the same features?

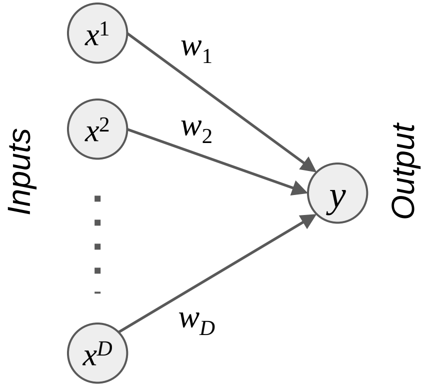
$$f(w, x) = \hat{y} = \mathrm{sign}(w_1 1 + w_2 x^1 + w_3 x^2 + w_4 x^3)$$

$$\hat{y} = \mathrm{sign}(w^T x)$$

#Code

Neural networks: single layer perceptron

- Can view the simple linear perceptron with its loss function, in the form of a **weighted linear combination** with nonlinear **activation function**



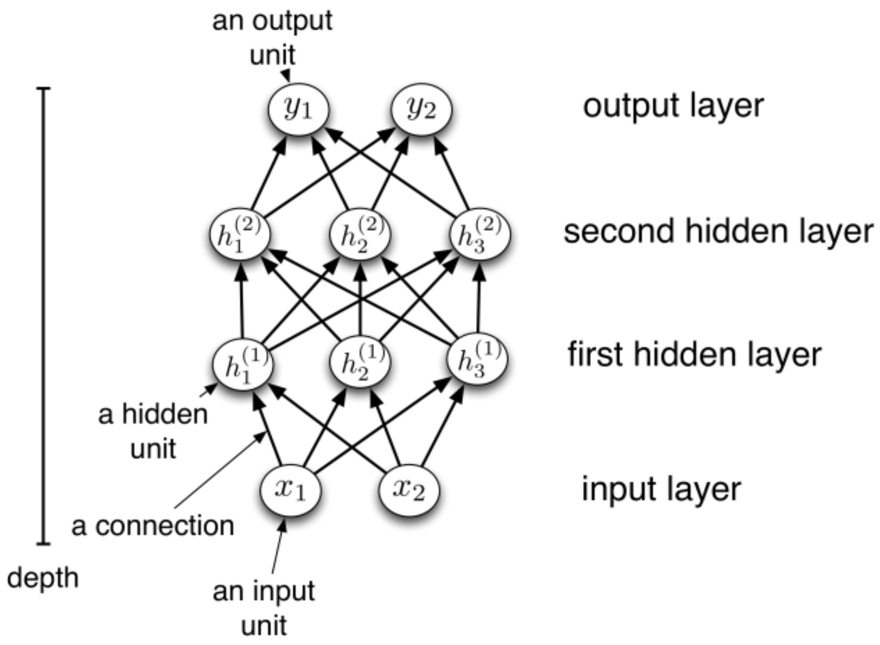
$$y = \max(0, w^T x)$$

- Inspired by a **biological neuron**, which has axons, dendrites and a nucleus, which fires when a certain threshold is crossed

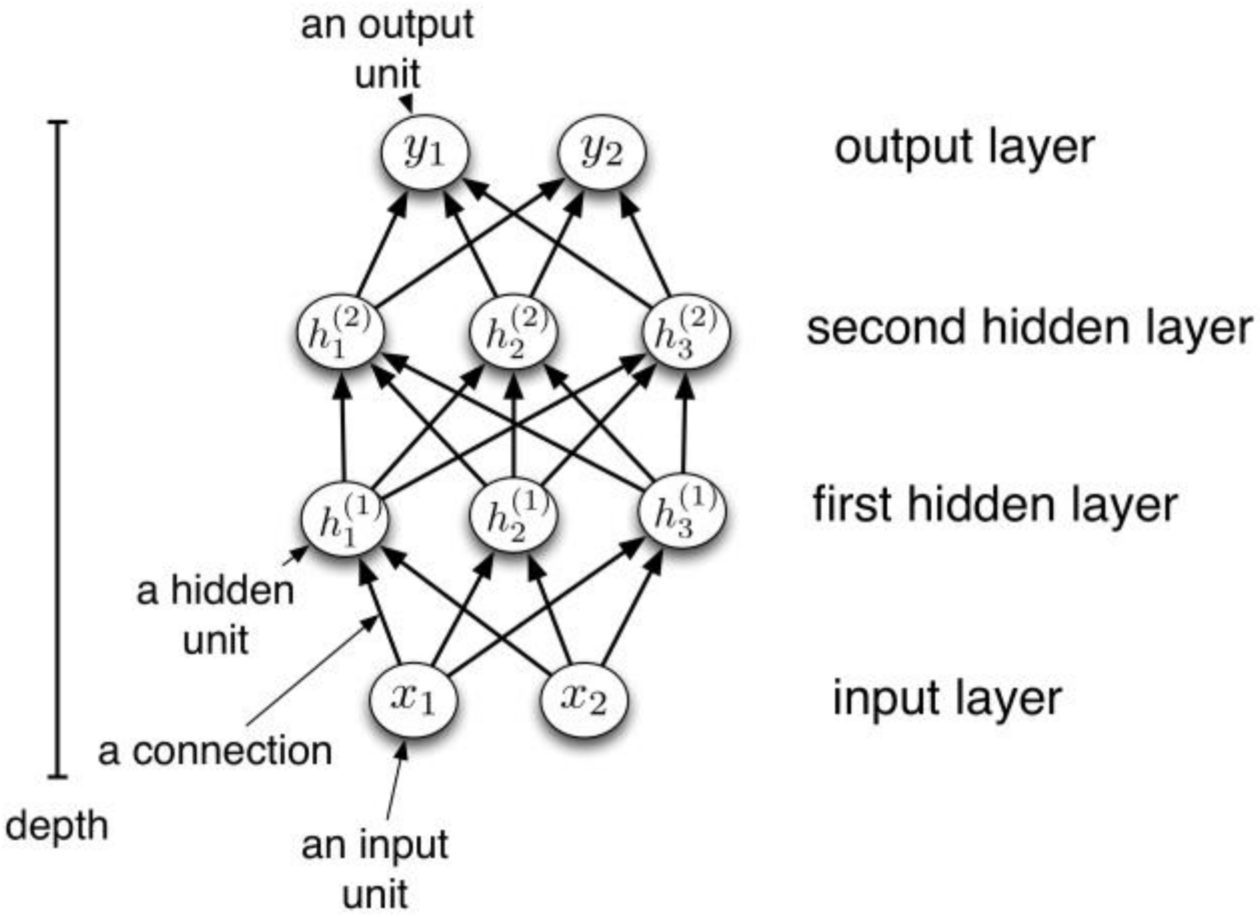
Multi-layer perceptron

#Code

- We can connect lots of units together into a directed acyclic graph.
- This gives a feed-forward neural network. That's in contrast to recurrent neural networks, which can have cycles. (We'll talk about those later.)
- Typically, units are grouped together into layers.



#Code


an output unit
y_1
y_2
output layer
$h_1^{(2)}$
$h_2^{(2)}$
$h_3^{(2)}$
second hidden layer
$h_1^{(1)}$
$h_2^{(1)}$
$h_3^{(1)}$
first hidden layer
a hidden unit
x_1
x_2
input layer
a connection
depth
an input unit

Multi-layer perceptron

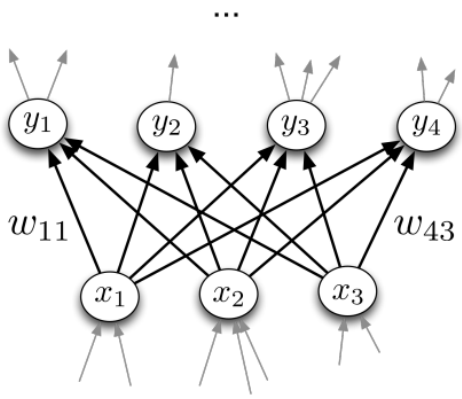
#Code

- Each layer connects N input units to M output units.
- In the simplest case, all input units are connected to all output units. We call this a fully connected layer. We'll consider other layer types later.
- Note: the inputs and outputs for a layer are distinct from the inputs and outputs to the network.

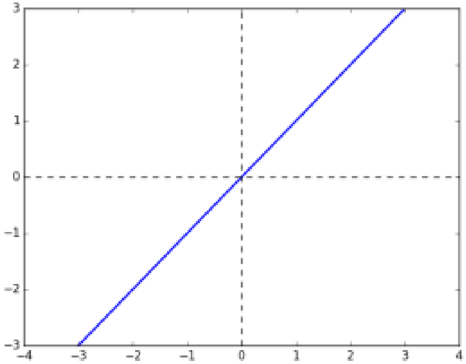
- Recall from multiway logistic regression: this means we need an $M \times N$ weight matrix.
- The output units are a function of the input units:

$$\mathbf{y} = f(\mathbf{x}) = \phi(\mathbf{W}\mathbf{x} + \mathbf{b})$$

- A multilayer network consisting of fully connected layers is called a multilayer perceptron. Despite the name, it has nothing to do with perceptrons!

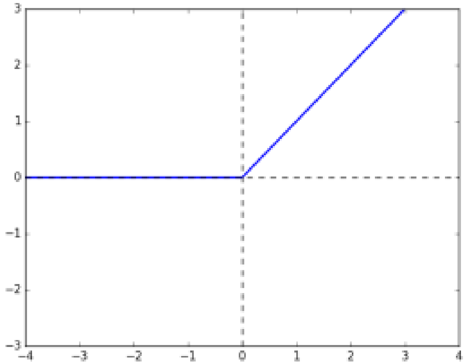


Some activation functions:



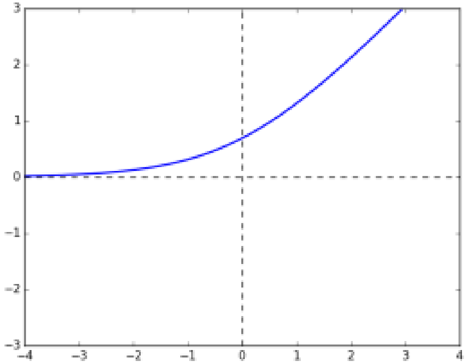
Linear

$$y = z$$



Rectified Linear Unit (ReLU)

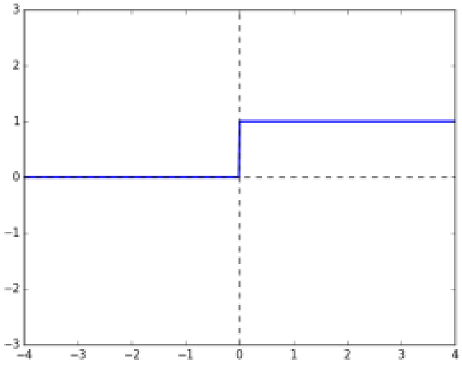
$$y = \max(0, z)$$



Soft ReLU

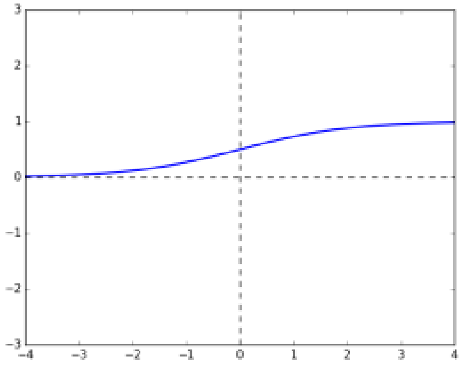
$$y = \log 1 + e^{z}$$

Some activation functions:



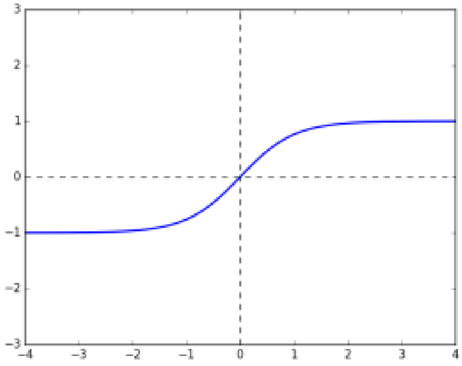
Hard Threshold

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



Logistic

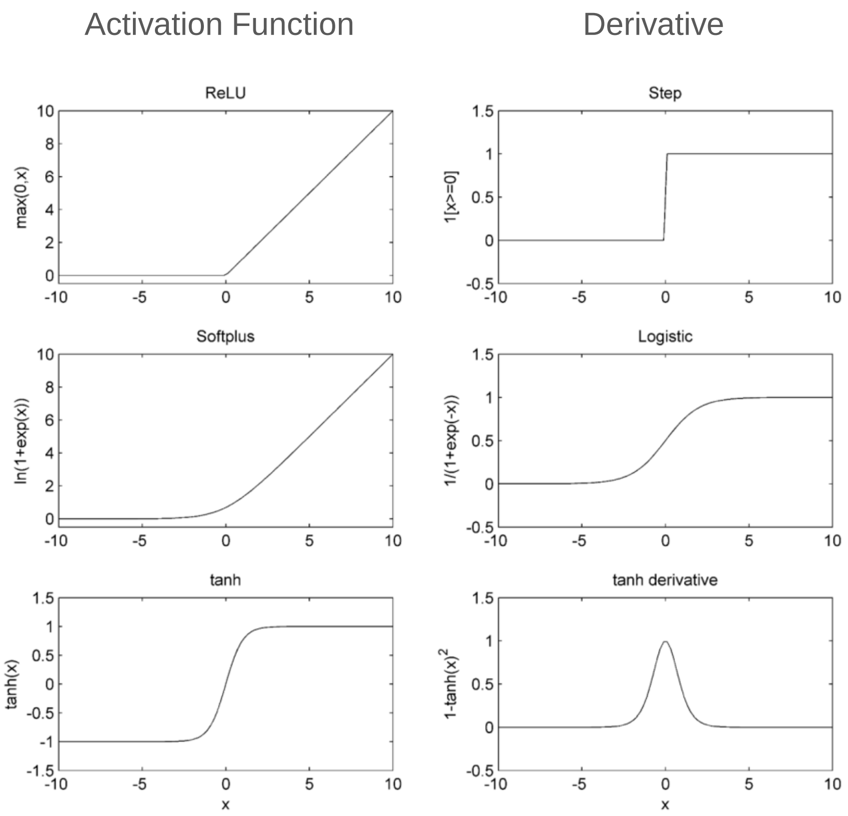
$$y = \frac{1}{1 + e^{-z}}$$



Hyperbolic Tangent (tanh)

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

Activation nonlinearities



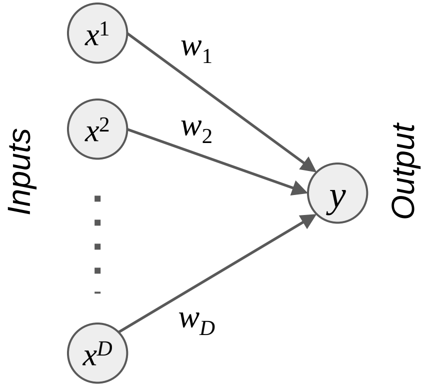
- Wide range of activation functions in use (**logistic**, **tanh**, **softplus**, **ReLU**): only criteria is that they must be **nonlinear** and should ideally be **differentiable** (almost everywhere)
- ReLU is perceptron loss, sigmoid is logistic regression loss
- ReLU most widely used activation;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

AIML

#Code

Neural networks: single layer perceptron

- Can view the simple linear perceptron with its loss function, in the form of a **weighted linear combination** with nonlinear **activation function**



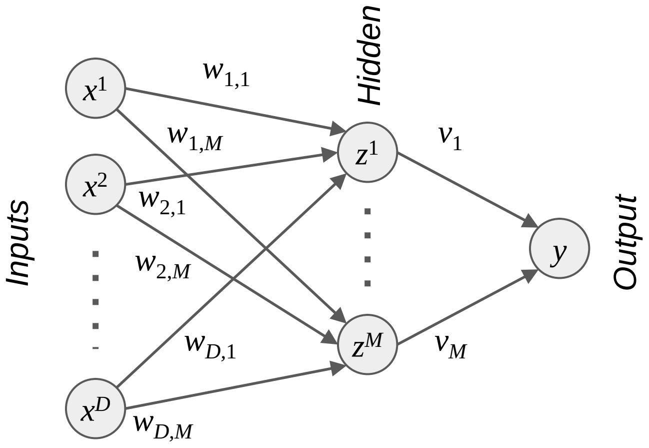
$$y = \max(0, w^T x)$$

- Inspired by a **biological neuron**, which has axons, dendrites and a nucleus, which fires when a certain threshold is crossed
- Perceptron is very limited, and can only model linear decision boundaries
- More complex nonlinear boundaries are required in practical ML

#Code

Neural networks: deep learning

- Extend the perceptron to two or more layers of weighted combinations (**linear layers**) with **nonlinear activations** connecting them



$$z = \max(0, W^T x),\ y = \max(0, v^T z)$$

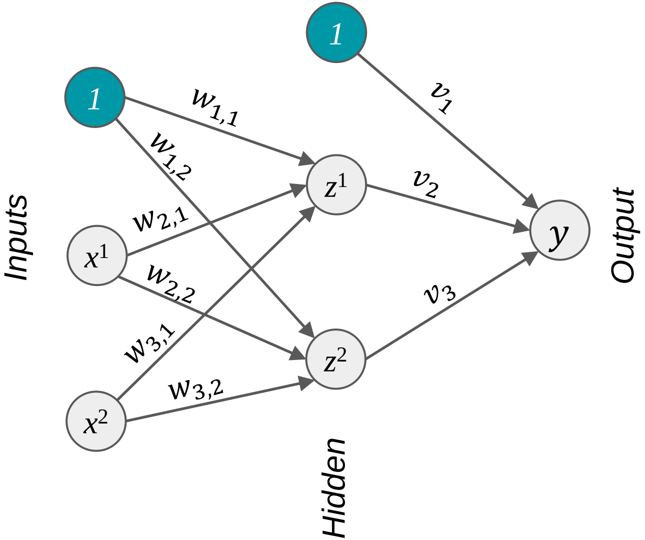
- Intermediate nodes are known as **hidden neurons**, whose output *z* is fed into the **output layer** which produces the final output *y*
- Modern **deep learning algorithms** usually have multiple hidden neurons, in multiple additional (hidden) layers
- Greatly extends the complexity of decision boundaries (**piecewise linear** boundaries)

AIML

#Code

Neural networks: deep learning

- Example of a multilayer neural network



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

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

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

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

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

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

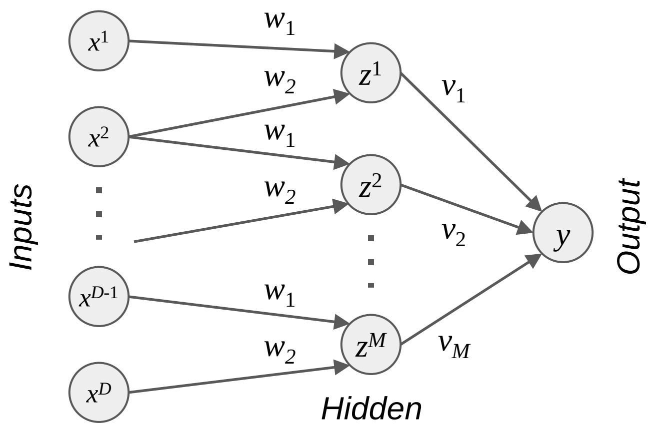
Neural networks: weights consideration

- **Fully connected networks** (every node in each layer connected to every node in the previous layer)

#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



$$z^1 = \max(0, w^T[x^1\ x^2]^T)$$

$$z^2 = \max(0, w^T[x^2\ x^3]^T)$$

$$\ldots$$

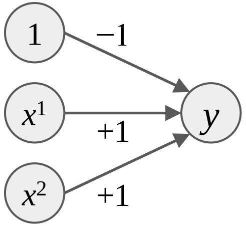
$$z^M = \max(0, w^T[x^{D-1}\ x^D]^T)$$

$$y = \max(0, v^T z)$$

Deep neural logic networks: example

- With sign activation function (similar to step activation), **logical neural networks** have simple weights
- Use these to implement basic logical functions "and", "or" and "not", encoding $True$ as $+1$, *False* as -1

$y = x^1 \wedge x^2$ (and)



$$True = +1, False = -1$$

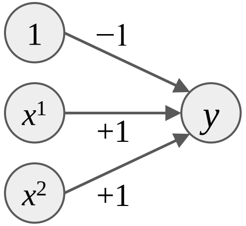
$$f_{\text{and}}(x_1, x_2) = \text{sign}(x_1 + x_2 - 1)$$

#Code

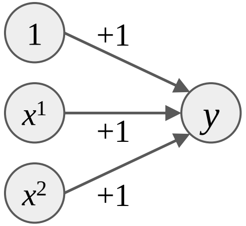
Deep neural logic networks: example

- With sign activation function (similar to step activation), **logical neural networks** have simple weights
- Use these to implement basic logical functions "and", "or" and "not", encoding *True* as $+1$, *False* as -1
- Any complex logical function can be implemented by composing these basic neurons together

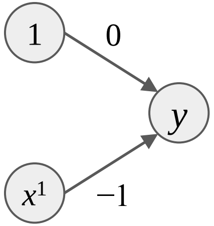
$y = x^1 \wedge x^2$ (and)



$y = x^1 \vee x^2$ (or)



$y = \neg\, x^1$ (not)



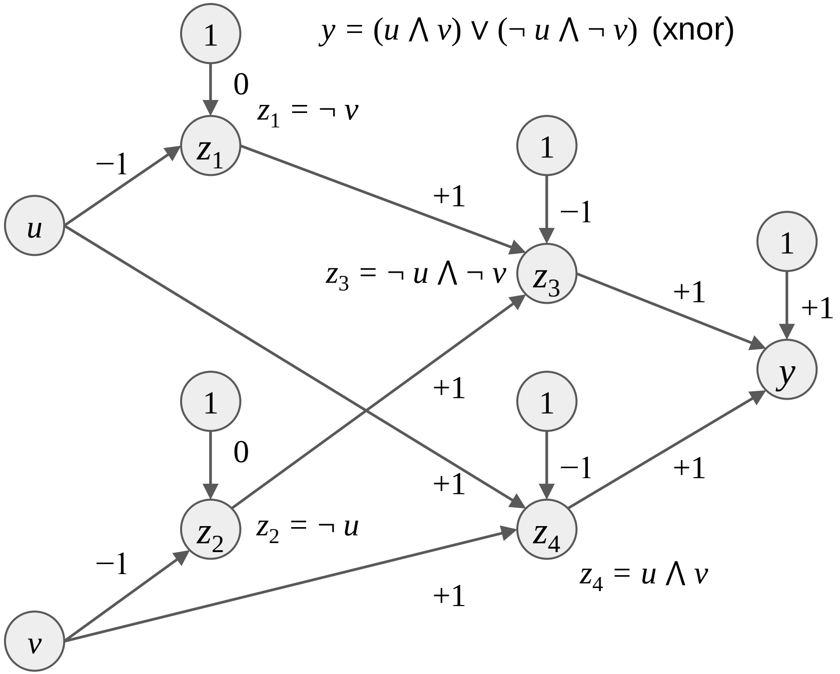
$$True = +1, False = -1$$

$$f_{\mathrm{and}}(x_1, x_2) = \mathrm{sign}(x_1 + x_2 - 1)$$

$$f_{\mathrm{or}}(x_1, x_2) = \mathrm{sign}(x_1 + x_2 + 1)$$

$$f_{\mathrm{not}}(x) = \mathrm{sign}(-x)$$

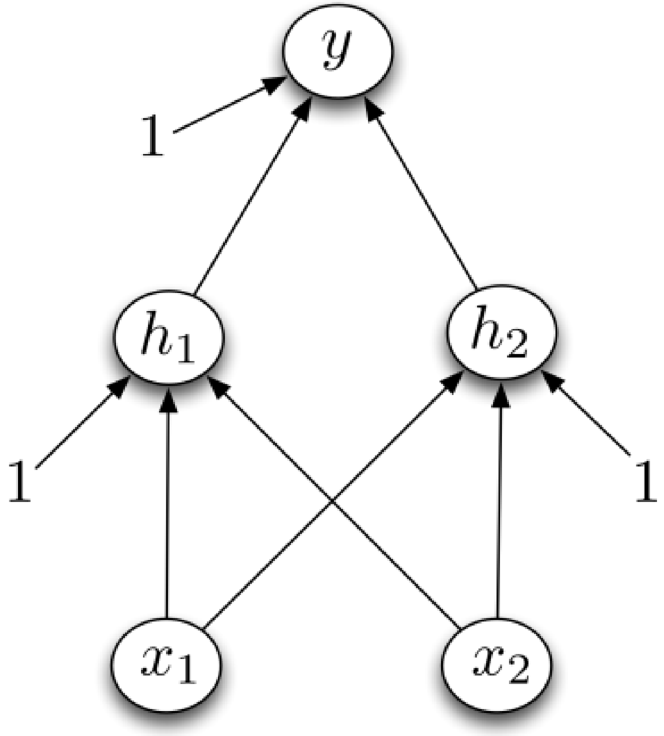
Deep neural logic networks: XNOR



- **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

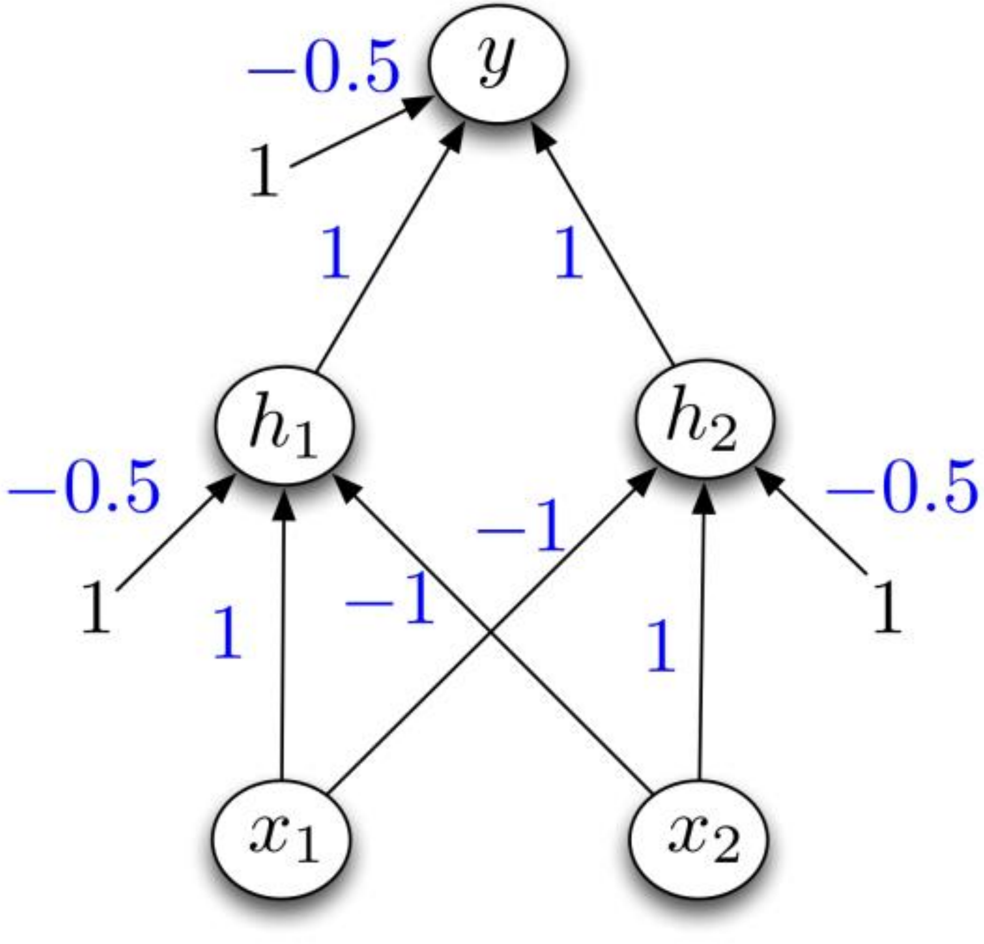
Designing a network to compute XOR:

Assume hard threshold activation function



XOR

#Code



To recap

- We learned the basic concepts of a **neural network**
  - Extended the concept to multiple (hidden) layers: **deep learning**
  - Problem of the number of weights & weight sharing: **convolutional neural network**
- **Next**: How to optimize the weights of a neural network
  - Pre-Reading: **Lecture Notes**, Section 14

Further Reading

- **PRML**, Section 5.1
- **H&T**, Section 11.3