# NEURAL NETWORKS

David Kauchak
CS158 – Spring 2019

## Admin

Assignment 4

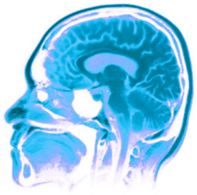
Assignment 5

Quiz #2 on Wednesday!

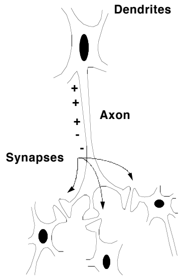
## Neural Networks

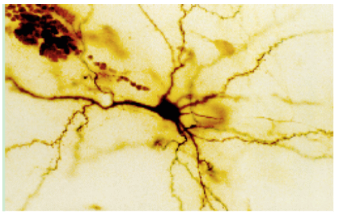
Neural Networks try to mimic the structure and function of our nervous system

***People like biologically motivated approaches***



## Our Nervous System

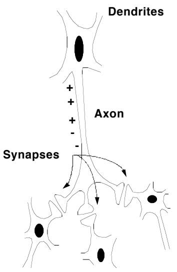




Neuron

What do you know?

## Our nervous system:
## the computer science view

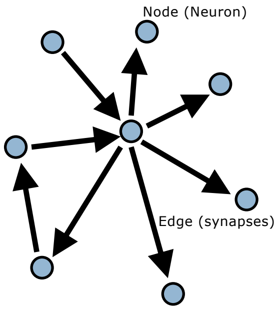


the human brain is a large collection of interconnected neurons

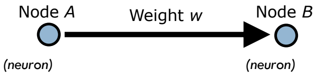
a NEURON is a brain cell

- they collect, process, and disseminate electrical signals
- they are connected via synapses
- they FIRE depending on the conditions of the neighboring neurons

## Artificial Neural Networks



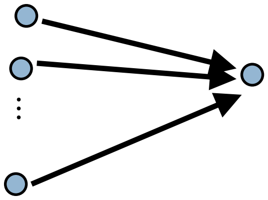
our approximation



*W* is the strength of signal sent between A and B.

If *A* fires and *w* is **positive**, then *A* **stimulates** *B*.

If *A fires* and *w* is **negative**, then *A* **inhibits** *B*.

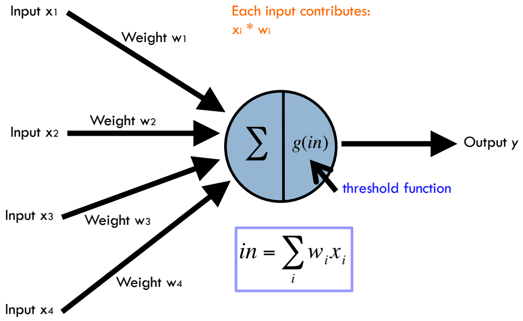


A given neuron has many, many connecting, input neurons

If a neuron is stimulated enough, then it also fires

How much stimulation is required is determined by its **threshold**

## A Single Neuron/Perceptron



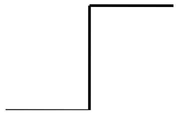
Each input contributes:
$x_i * w_i$

$$in = \sum_i w_i x_i$$

## Activation functions

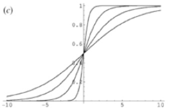
hard threshold:

$$g(in) = \begin{cases} 1 & if\ in \geq T \\ 0 & otherwise \end{cases}$$

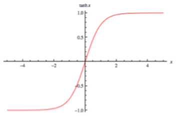


sigmoid

$$g(x) = \frac{1}{1 + e^{-ax}}$$

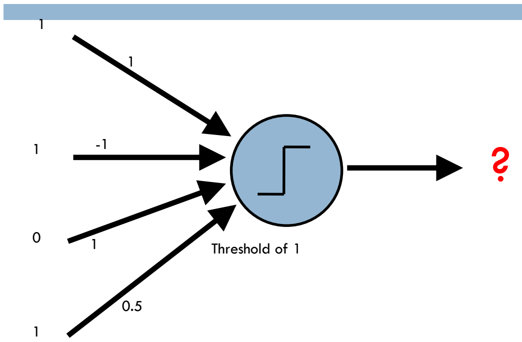


tanh x

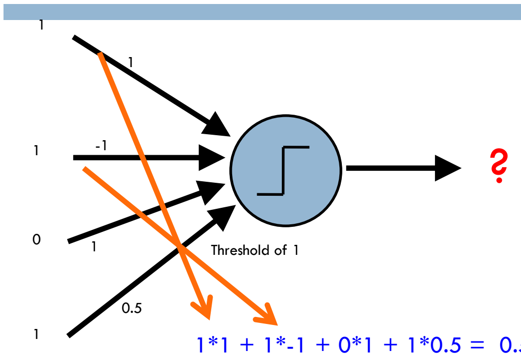


why other threshold functions?

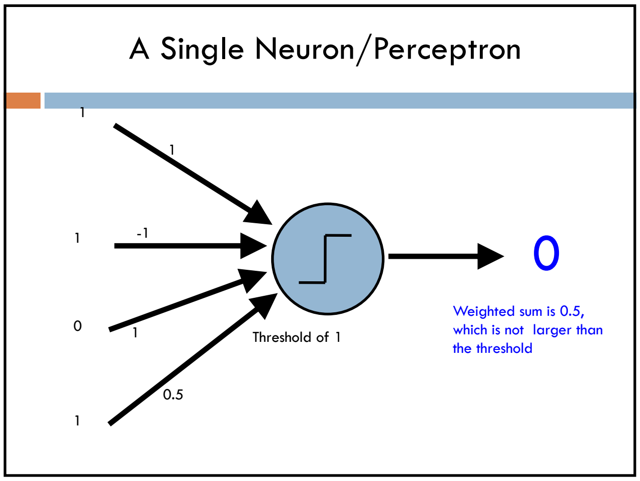
## A Single Neuron/Perceptron



## A Single Neuron/Perceptron



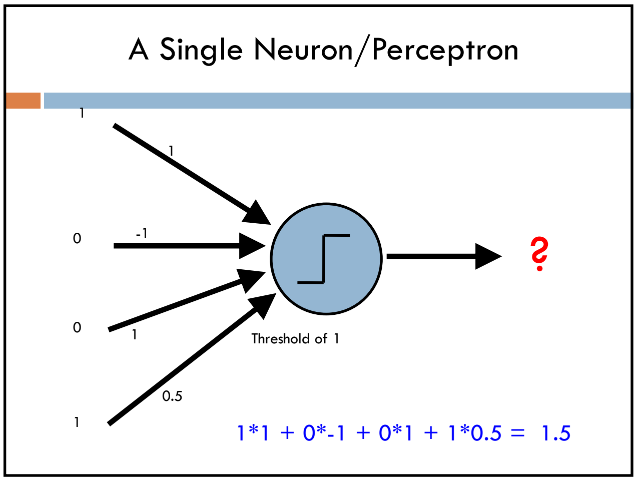
1*1 + 1*-1 + 0*1 + 1*0.5 = 0.5



## A Single Neuron/Perceptron

Threshold of 1

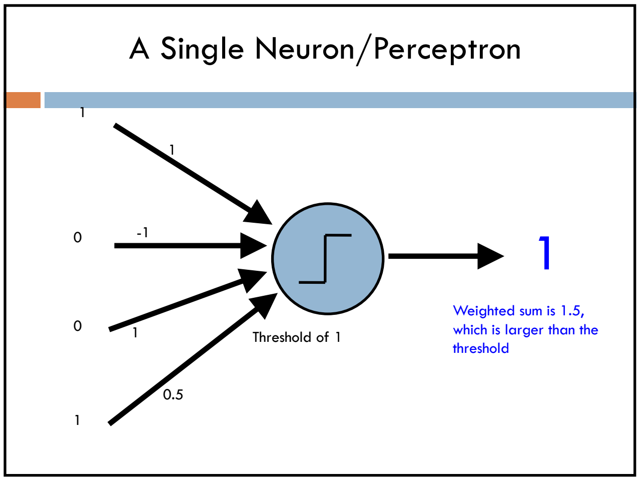
Weighted sum is 0.5, which is not larger than the threshold



## A Single Neuron/Perceptron

Threshold of 1

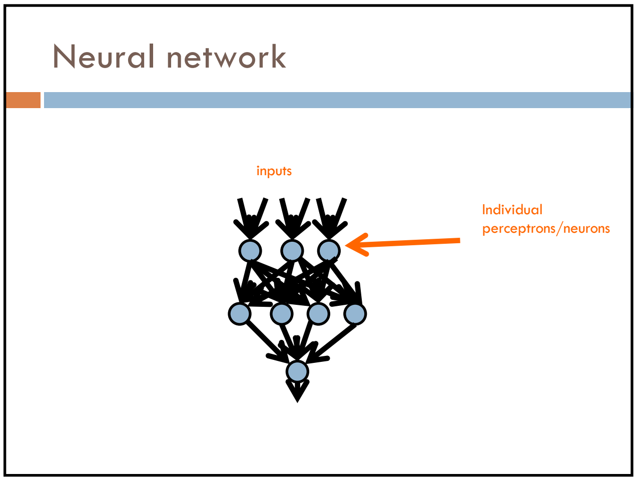
1*1 + 0*-1 + 0*1 + 1*0.5 = 1.5



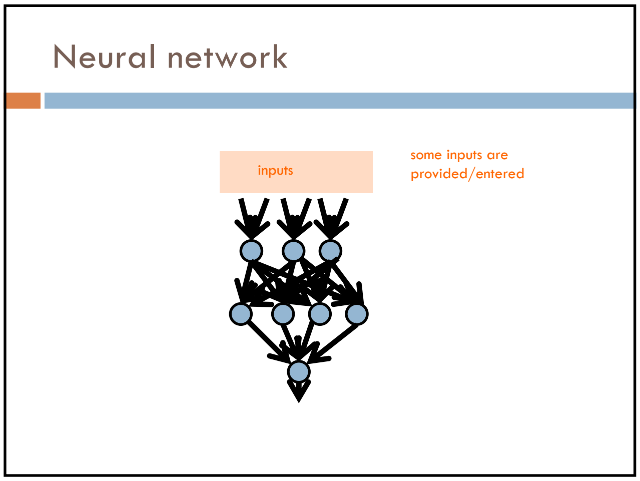
## A Single Neuron/Perceptron

Threshold of 1

Weighted sum is 1.5, which is larger than the threshold



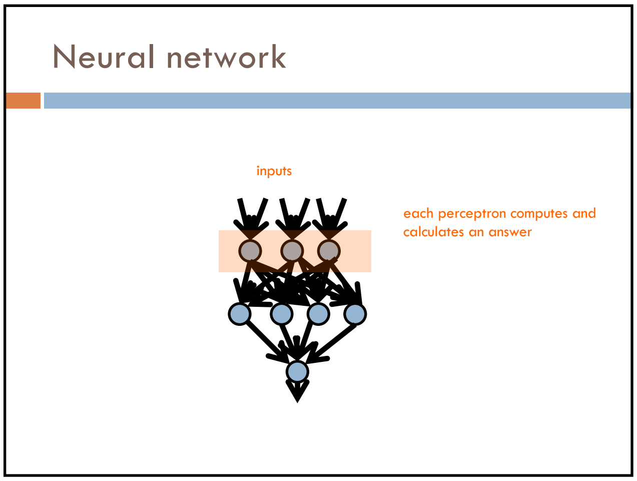
## Neural network



## Neural network

inputs

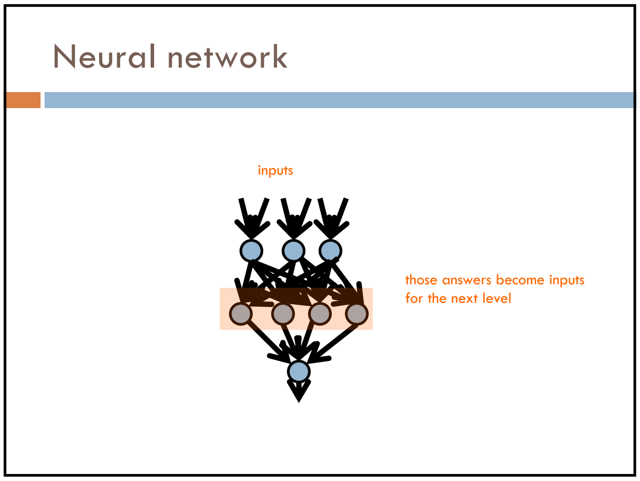
some inputs are provided/entered



## Neural network

inputs

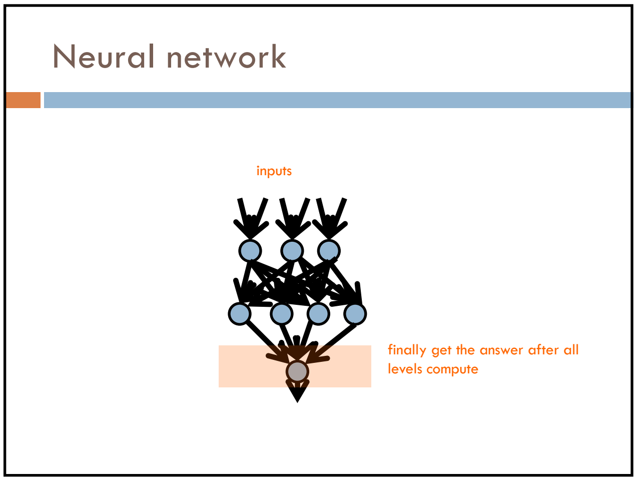
each perceptron computes and calculates an answer



## Neural network

inputs

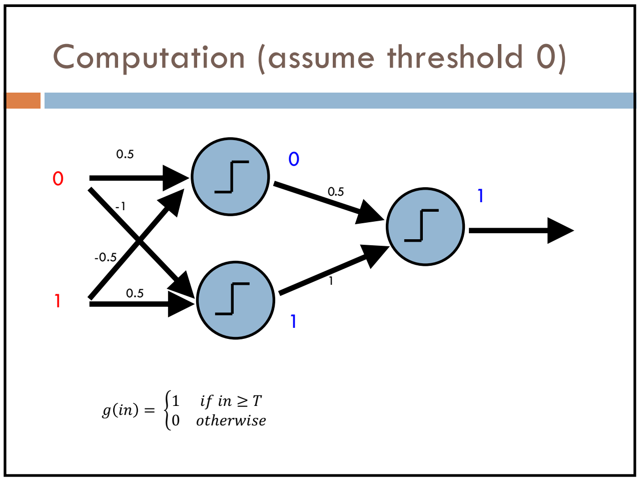
those answers become inputs for the next level



## Neural network

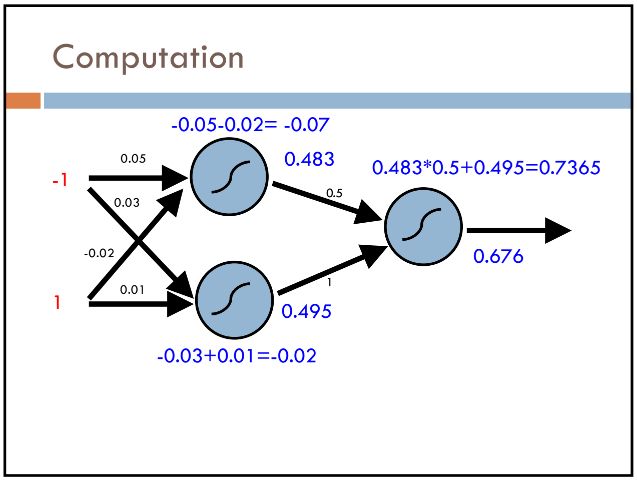
inputs

finally get the answer after all levels compute

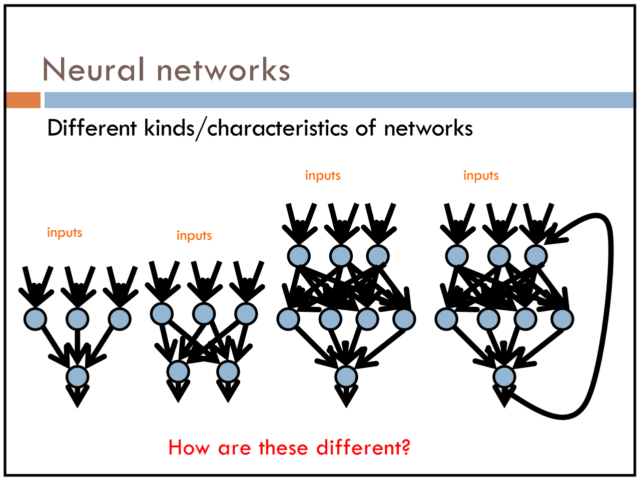


## Computation (assume threshold 0)

$$g(in) = \begin{cases} 1 & if\ in \geq T \\ 0 & otherwise \end{cases}$$



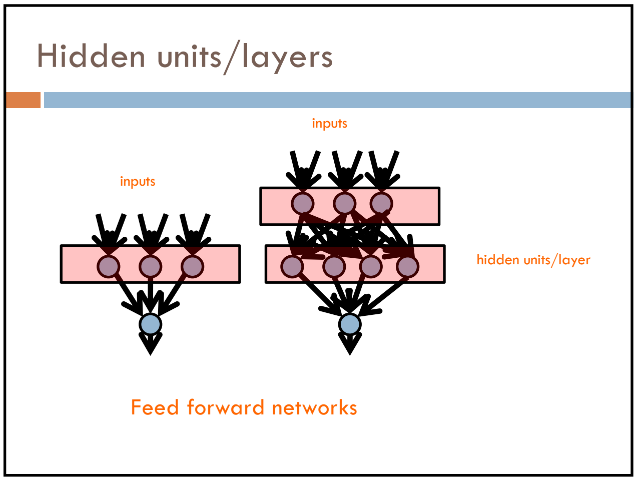
## Computation



## Neural networks

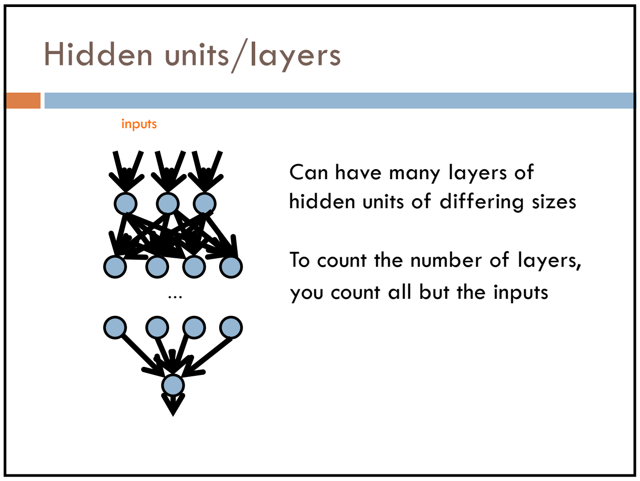
Different kinds/characteristics of networks

How are these different?



## Hidden units/layers

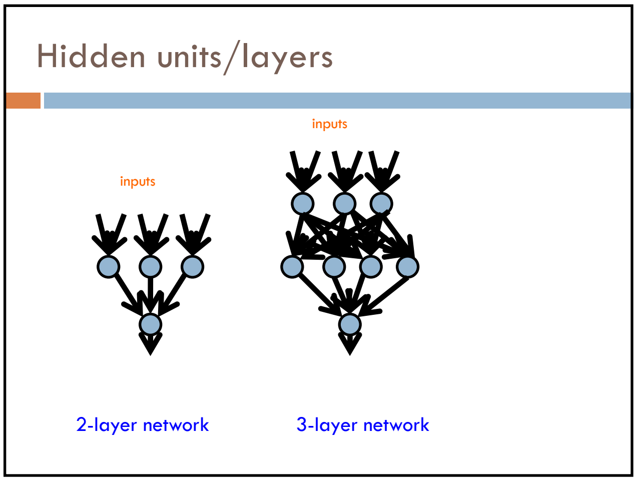
Feed forward networks



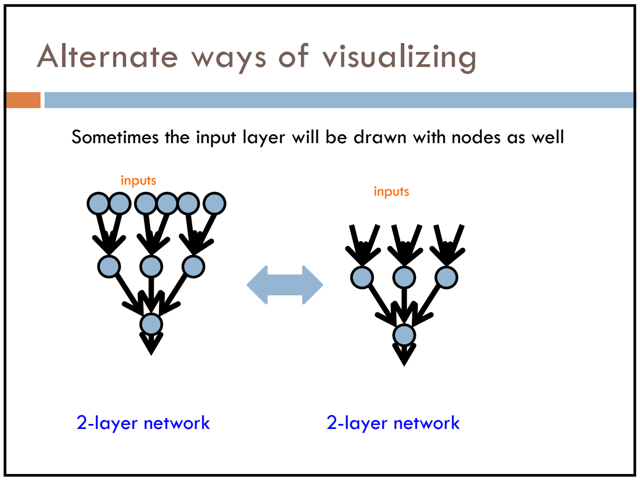
## Hidden units/layers

Can have many layers of hidden units of differing sizes

To count the number of layers, you count all but the inputs

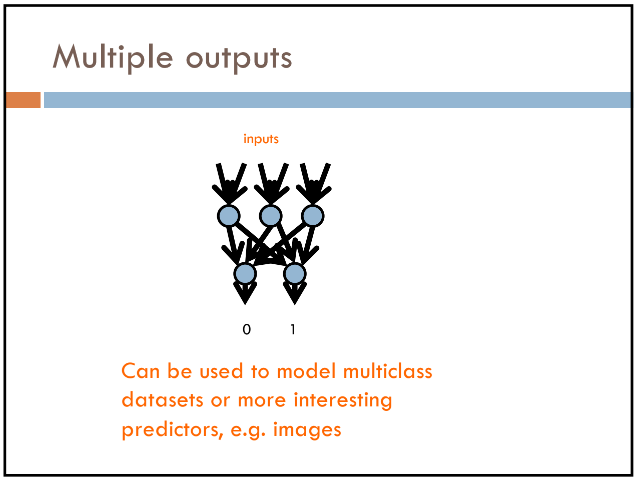


## Hidden units/layers



## Alternate ways of visualizing

Sometimes the input layer will be drawn with nodes as well



## Multiple outputs

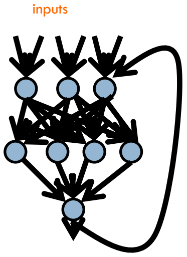
Can be used to model multiclass datasets or more interesting predictors, e.g. images

## Multiple outputs

input

output
(edge detection)

## Neural networks



Recurrent network

Output is fed back to input

Can support memory!

Good for temporal data

## History of Neural Networks

McCulloch and Pitts (1943) – introduced model of artificial neurons and suggested they could learn

Hebb (1949) – Simple updating rule for learning

Rosenblatt (1962) - the *perceptron* model

Minsky and Papert (1969) – wrote *Perceptrons*

Bryson and Ho (1969, but largely ignored until 1980s--Rosenblatt) – invented back-propagation learning for multilayer networks

## Training the perceptron

First wave in neural networks in the 1960's

Single neuron

Trainable: its threshold and input weights can be modified

If the neuron doesn't give the desired output, then it has made a mistake

Input weights and threshold can be changed according to a learning algorithm

## Examples - Logical operators

**AND** – if all inputs are 1, return 1, otherwise return 0

**OR** – if at least one input is 1, return 1, otherwise return 0

**NOT** – return the opposite of the input

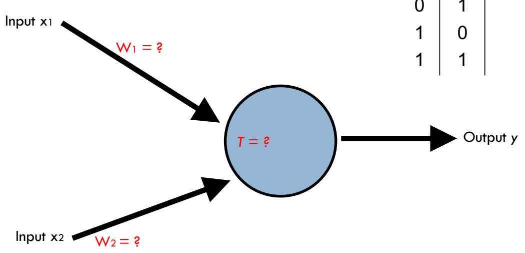
**XOR** – if exactly one input is 1, then return 1, otherwise return 0

## AND

| $x_1$ | $x_2$ | $x_1$ **and** $x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

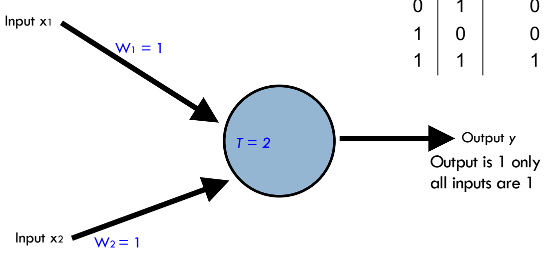
## AND

| $x_1$ | $x_2$ | $x_1$ **and** $x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



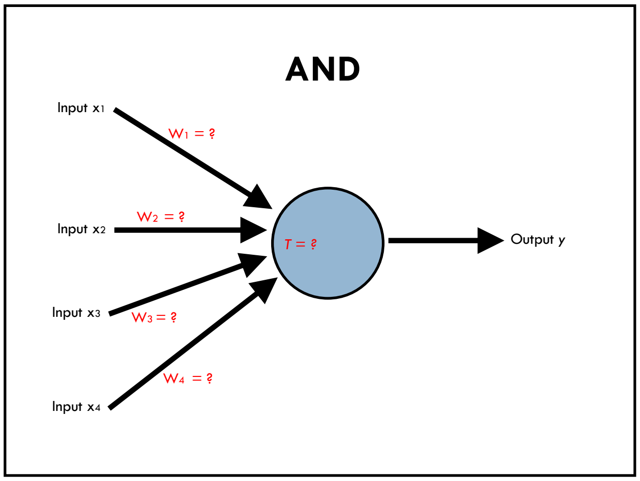
## AND

| $x_1$ | $x_2$ | $x_1$ **and** $x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

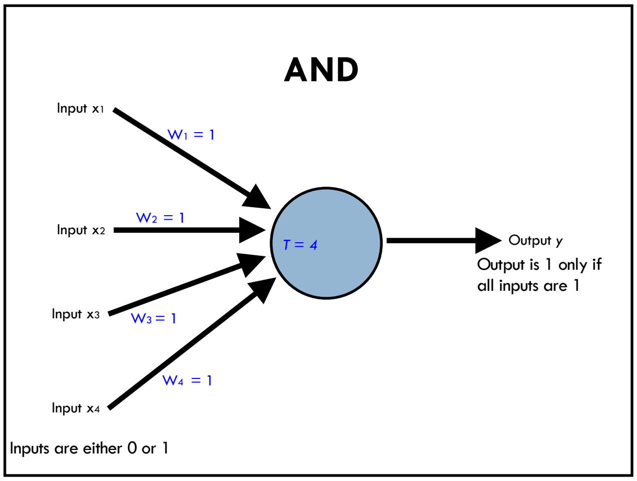


Output is 1 only if all inputs are 1

Inputs are either 0 or 1



## AND



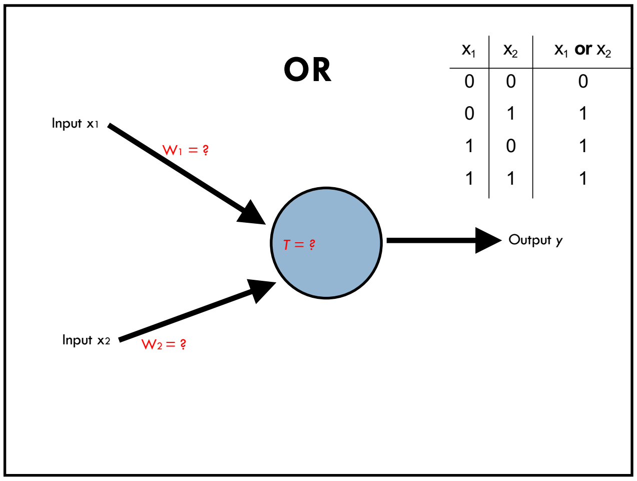
## AND

Output is 1 only if all inputs are 1

Inputs are either 0 or 1

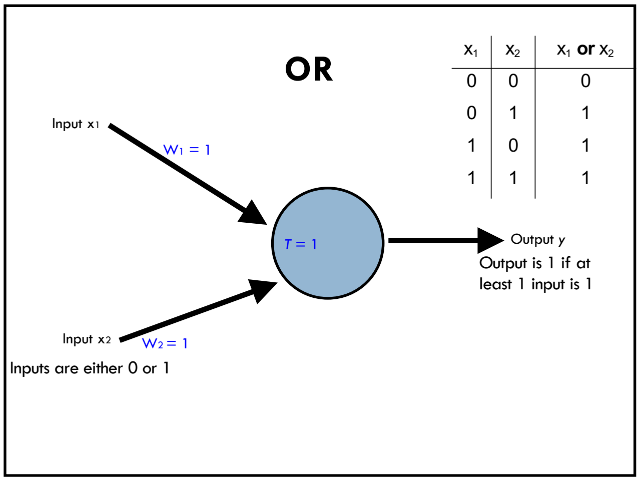
## OR

| $x_1$ | $x_2$ | $x_1$ **or** $x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |



## OR

| $x_1$ | $x_2$ | $x_1$ **or** $x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |



## OR

| $x_1$ | $x_2$ | $x_1$ **or** $x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Input $x_1$

$W_1 = 1$

$T = 1$

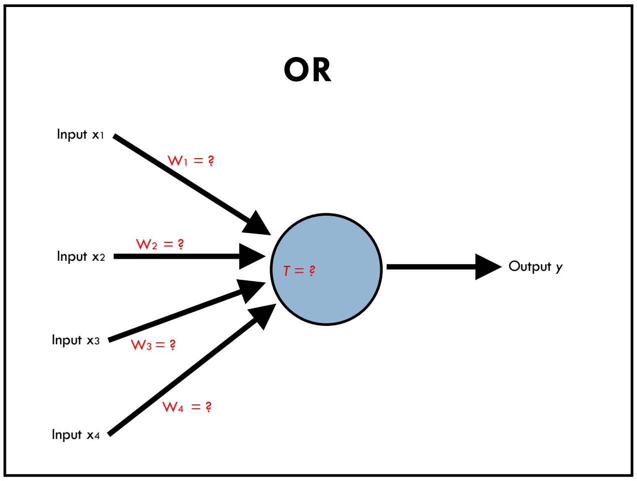
Output $y$

Output is 1 if at least 1 input is 1

Input $x_2$

$W_2 = 1$

Inputs are either 0 or 1



## OR

Input $x_1$

$W_1 = ?$

Input $x_2$

$W_2 = ?$

$T = ?$

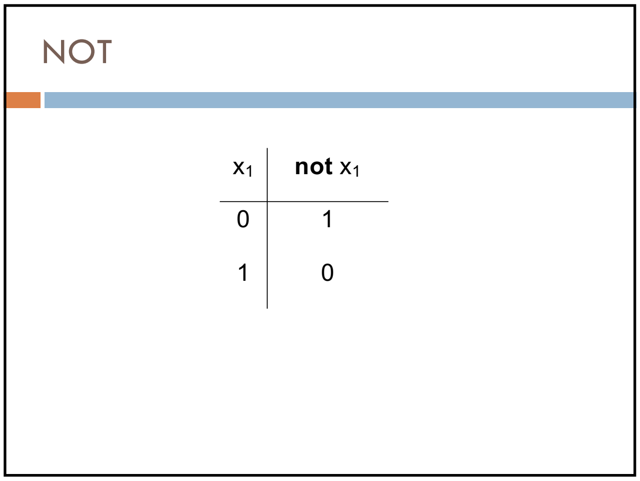
Output $y$

Input $x_3$

$W_3 = ?$

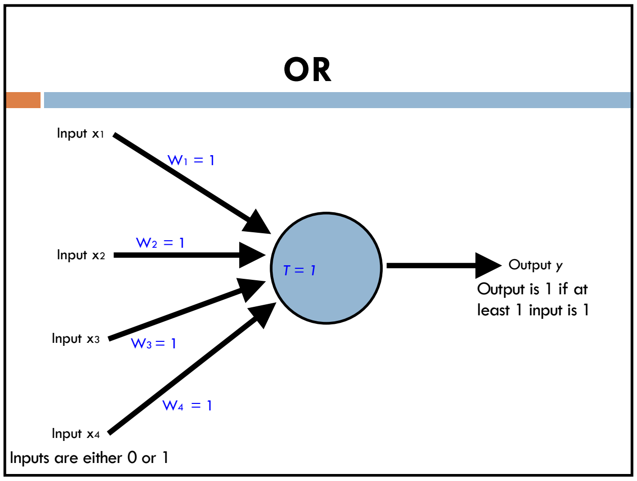
$W_4 = ?$

Input $x_4$



## NOT

| $x_1$ | **not** $x_1$ |
|---|---|
| 0 | 1 |
| 1 | 0 |



## OR

Input $x_1$

$W_1 = 1$

$W_2 = 1$

Input $x_2$

$T = 1$

Output $y$

Output is 1 if at least 1 input is 1

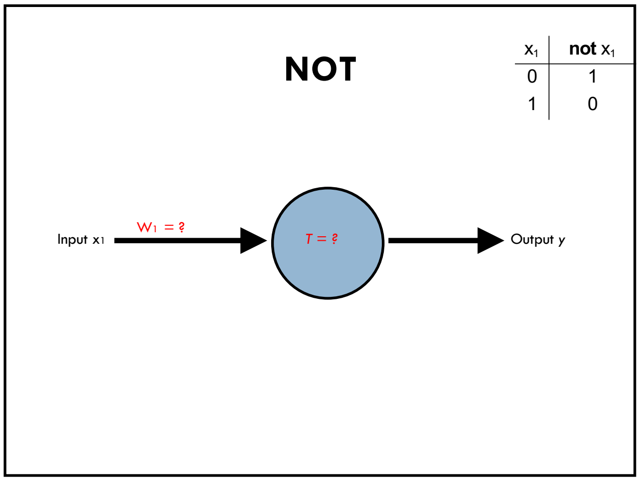
Input $x_3$

$W_3 = 1$

$W_4 = 1$

Input $x_4$

Inputs are either 0 or 1



# NOT

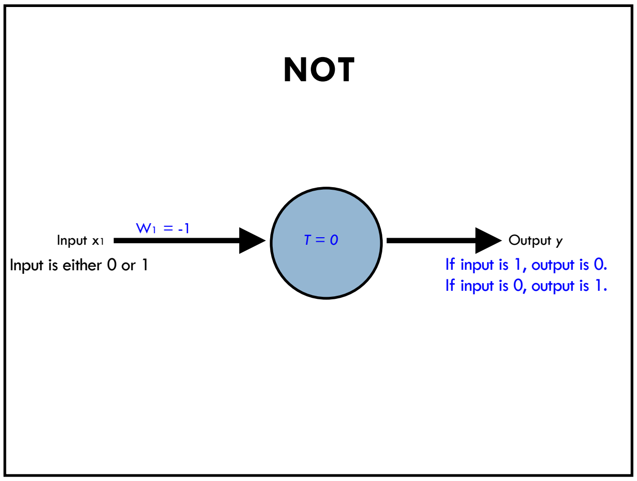
| $x_1$ | not $x_1$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

Input $x_1$

$W_1$ = ?

*T* = ?

Output *y*



# NOT

Input $x_1$

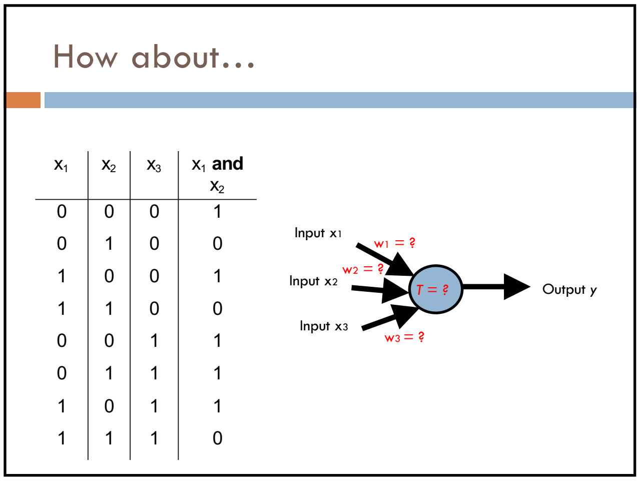
$W_1$ = -1

*T* = 0

Output *y*

Input is either 0 or 1

If input is 1, output is 0.
If input is 0, output is 1.



## How about…

| $x_1$ | $x_2$ | $x_3$ | $x_1$ and $x_2$ |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 |

Input $x_1$

$w_1$ = ?

Input $x_2$

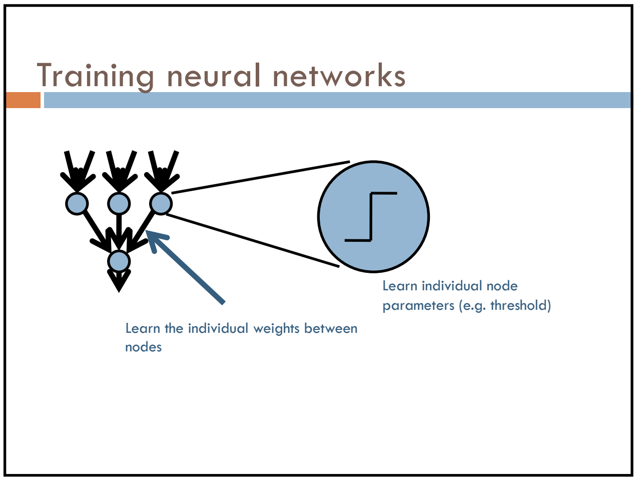
$w_2$ = ?

Input $x_3$

$w_3$ = ?

*T* = ?

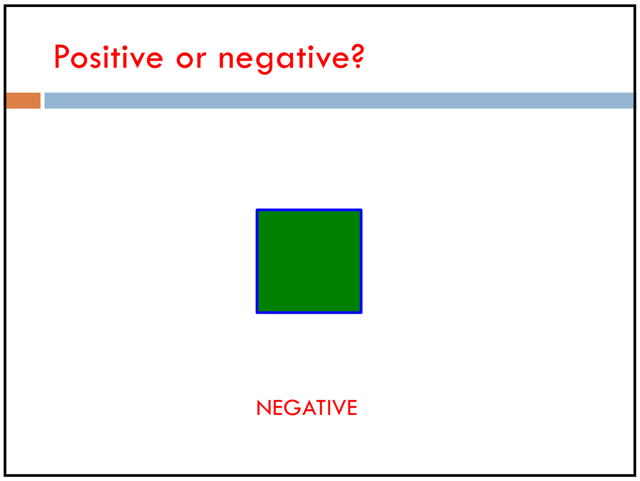
Output *y*



## Training neural networks

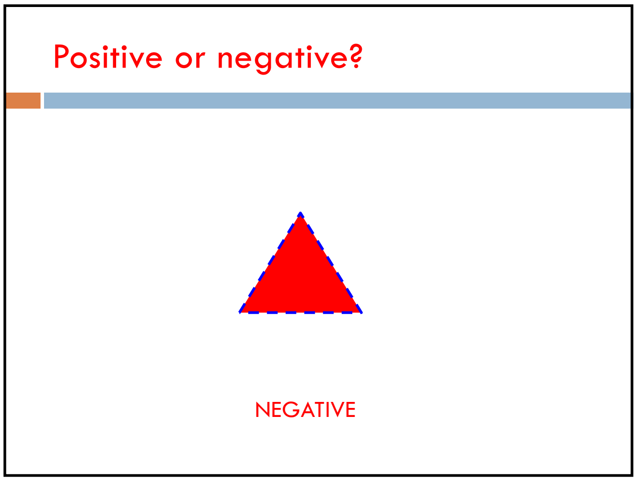
Learn the individual weights between nodes

Learn individual node parameters (e.g. threshold)



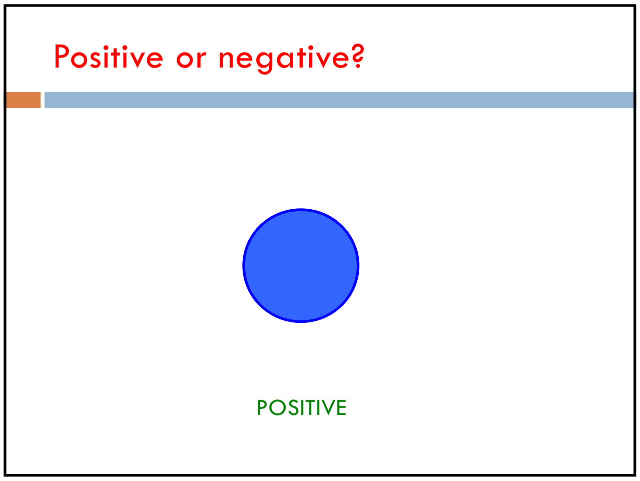
## Positive or negative?

NEGATIVE



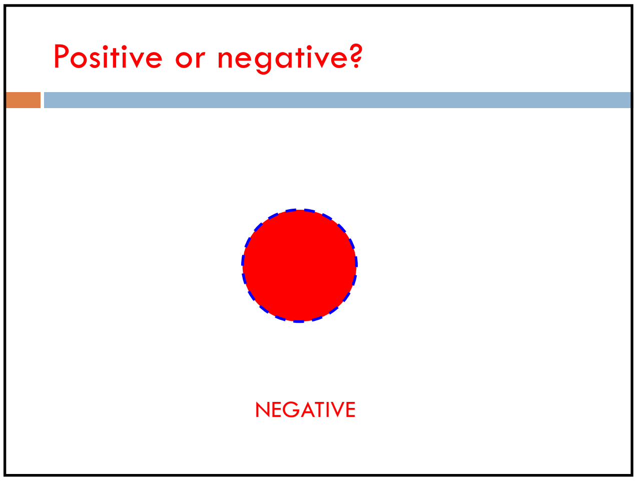
## Positive or negative?

NEGATIVE



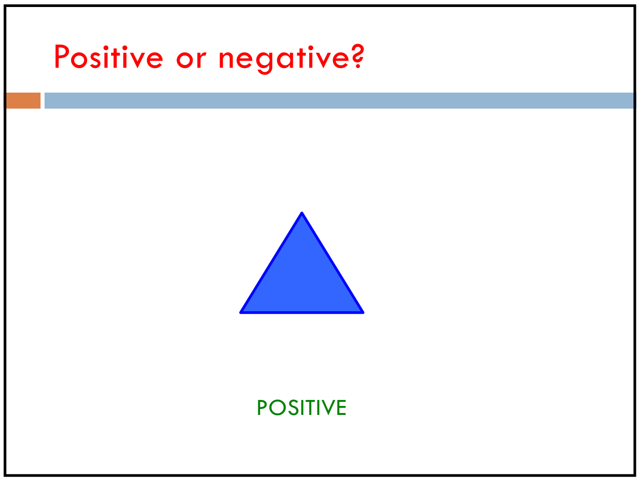
## Positive or negative?

POSITIVE



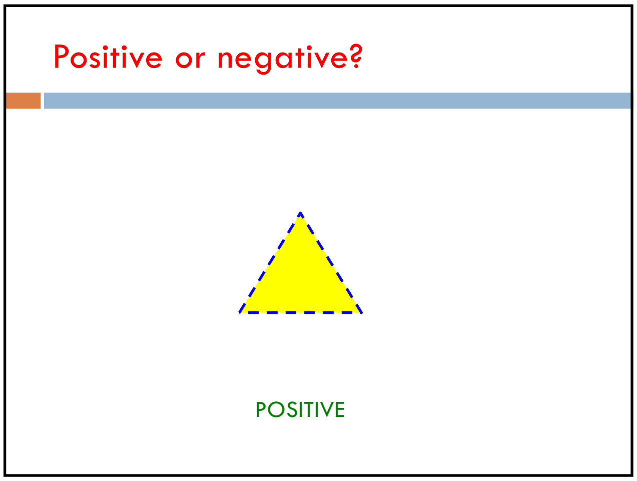
## Positive or negative?

NEGATIVE



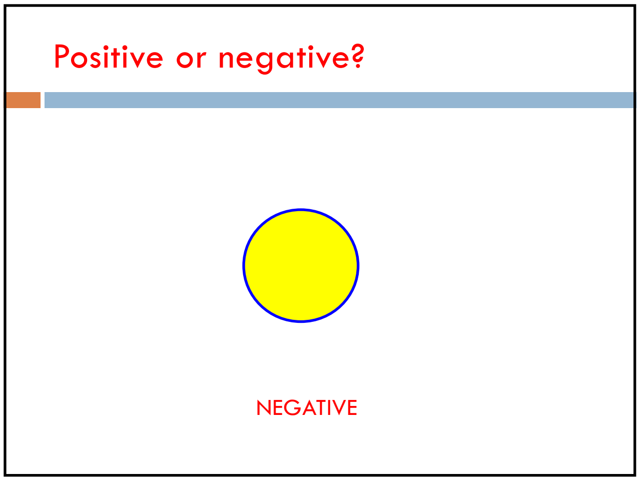
## Positive or negative?

POSITIVE



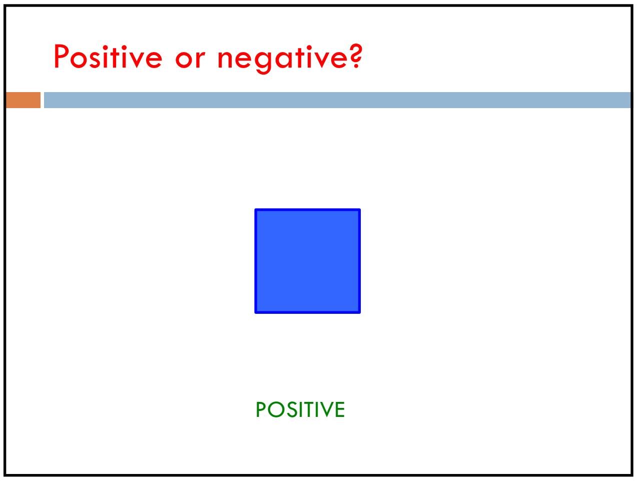
## Positive or negative?

POSITIVE



## Positive or negative?

NEGATIVE



## Positive or negative?

POSITIVE

## A method to the madness

blue = positive

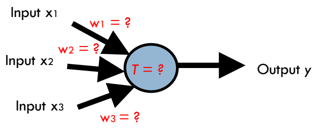
yellow triangles = positive

all others negative

How did you figure this out (or some of it)?

## Training a neuron (perceptron)

| $x_1$ | $x_2$ | $x_3$ | $x_1$ and $x_2$ |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 |



1. start with some initial weights and thresholds
2. show examples repeatedly to NN
3. update weights/thresholds by comparing NN output to actual output

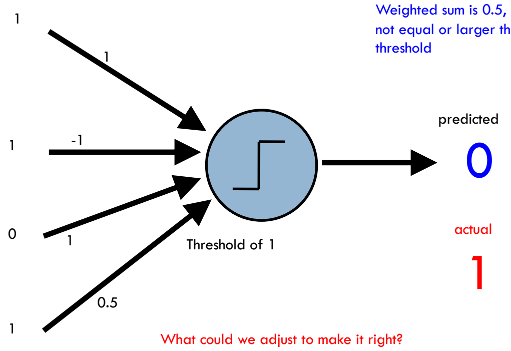
## Perceptron learning algorithm

repeat until you get all examples right:

- for each "training" example:
  - calculate current prediction on example
  - if wrong:
    - update weights and threshold towards getting this example correct

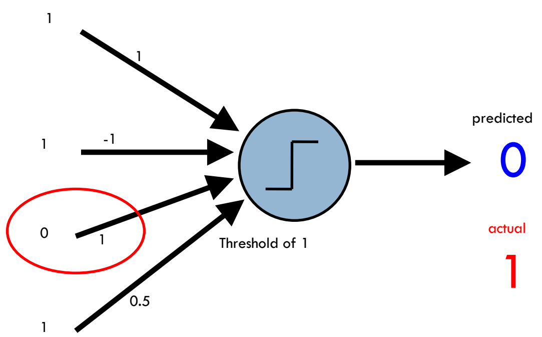
## Perceptron learning

Weighted sum is 0.5, which is not equal or larger than the threshold



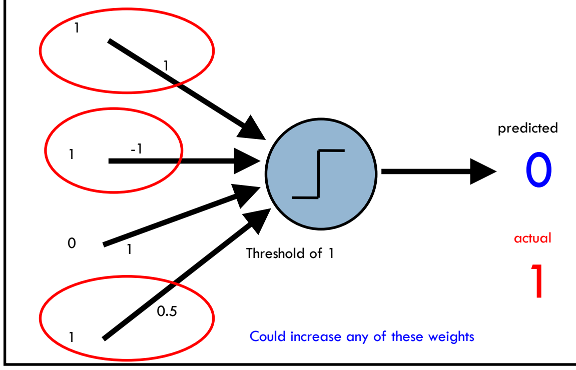
What could we adjust to make it right?

## Perceptron learning



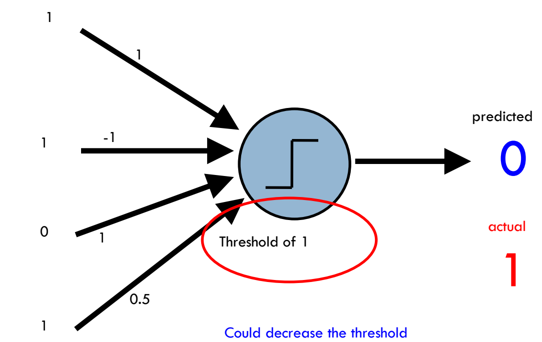
This weight doesn't matter, so don't change

## Perceptron learning



Could increase any of these weights

## Perceptron learning



Could decrease the threshold

## Perceptron learning

A few missing details, but not much more than this

Keeps adjusting weights as long as it makes mistakes

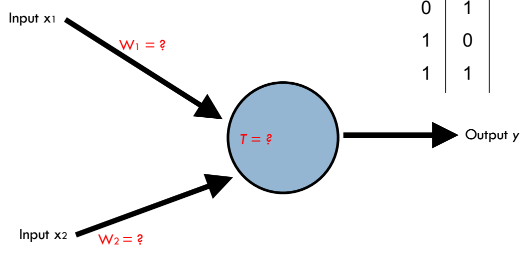
Run through the training data multiple times until convergence, some number of iterations, or until weights don't change (much)

## XOR

| $x_1$ | $x_2$ | $x_1$ **or** $x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

## XOR

| $x_1$ | $x_2$ | $x_1$ **xor** $x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



## Perceptron learning

A few missing details, but not much more than this

Keeps adjusting weights as long as it makes mistakes

Run through the training data multiple times until convergence, some number of iterations, or until weights don't change (much)

If the training data is linearly separable the perceptron learning algorithm is guaranteed to converge to the "correct" solution (where it gets all examples right)

## Linearly Separable

| $x_1$ | $x_2$ | $x_1$ **and** $x_2$ | |
|---|---|---|---|
| 0 | 0 | 0 | 🔵 |
| 0 | 1 | 0 | 🔵 |
| 1 | 0 | 0 | 🔵 |
| 1 | 1 | 1 | 🔴 |

| $x_1$ | $x_2$ | $x_1$ **or** $x_2$ | |
|---|---|---|---|
| 0 | 0 | 0 | 🔵 |
| 0 | 1 | 1 | 🔴 |
| 1 | 0 | 1 | 🔴 |
| 1 | 1 | 1 | 🔴 |

| $x_1$ | $x_2$ | $x_1$ **xor** $x_2$ | |
|---|---|---|---|
| 0 | 0 | 0 | 🔵 |
| 0 | 1 | 1 | 🔴 |
| 1 | 0 | 1 | 🔴 |
| 1 | 1 | 0 | 🔵 |

A data set is linearly separable if you can separate one example type from the other

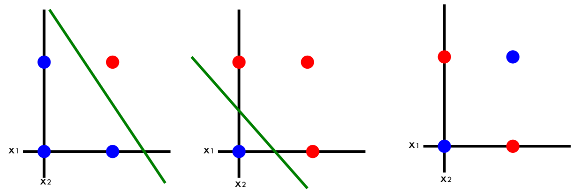
Which of these are linearly separable?

Which of these are linearly separable?

| $x_1$ | $x_2$ | $x_1$ **and** $x_2$ | |
|---|---|---|---|
| 0 | 0 | 0 | 🔵 |
| 0 | 1 | 0 | 🔵 |
| 1 | 0 | 0 | 🔵 |
| 1 | 1 | 1 | 🔴 |

| $x_1$ | $x_2$ | $x_1$ **or** $x_2$ | |
|---|---|---|---|
| 0 | 0 | 0 | 🔵 |
| 0 | 1 | 1 | 🔴 |
| 1 | 0 | 1 | 🔴 |
| 1 | 1 | 1 | 🔴 |

| $x_1$ | $x_2$ | $x_1$ **xor** $x_2$ | |
|---|---|---|---|
| 0 | 0 | 0 | 🔵 |
| 0 | 1 | 1 | 🔴 |
| 1 | 0 | 1 | 🔴 |
| 1 | 1 | 0 | 🔵 |



## Perceptrons

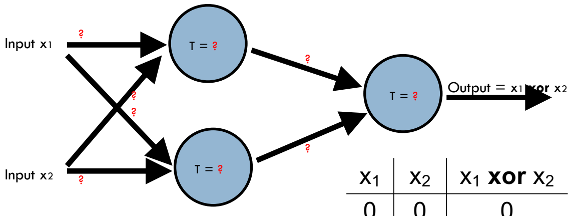
1969 book by Marvin Minsky and Seymour Papert

The problem is that they can only work for classification problems that are linearly separable

Insufficiently expressive

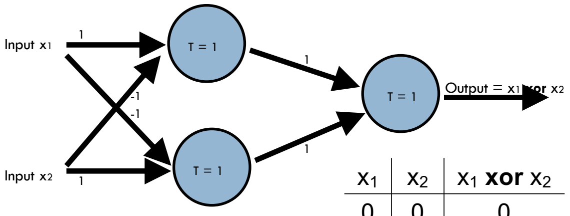
"Important research problem" to investigate multilayer networks although they were pessimistic about their value

## XOR



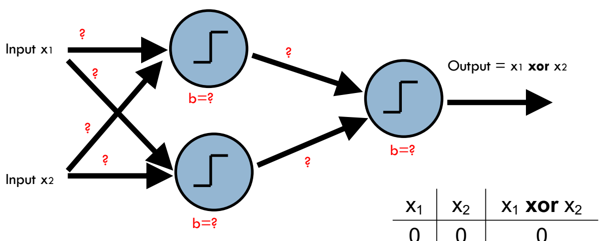
| $x_1$ | $x_2$ | $x_1$ **xor** $x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

## XOR



| $x_1$ | $x_2$ | $x_1$ **xor** $x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

## Training



| $x_1$ | $x_2$ | $x_1$ **xor** $x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

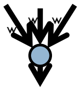
How do we learn the weights?

## Training multilayer networks

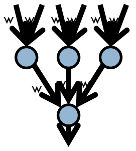
perceptron learning: if the perceptron's output is different than the expected output, update the weights

gradient descent: compare output to label and adjust based on loss function

Any other problem with these for general NNs?



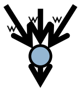
perceptron/
linear model



neural network

## Learning in multilayer networks

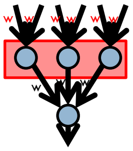
**Challenge:** for multilayer networks, we don't know what the expected output/error is for the internal nodes!



perceptron/
linear model

how do we learn these weights?

expected output?

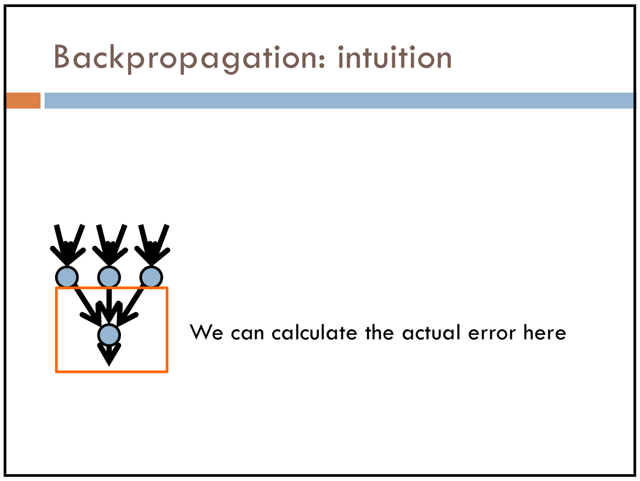


neural network

## Backpropagation: intuition

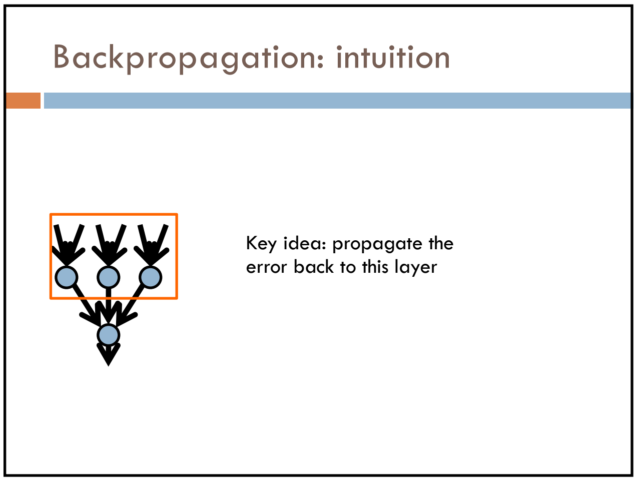
Gradient descent method for learning weights by optimizing a loss function

1. calculate output of all nodes
2. calculate the weights for the output layer based on the error
3. "backpropagate" errors through hidden layers



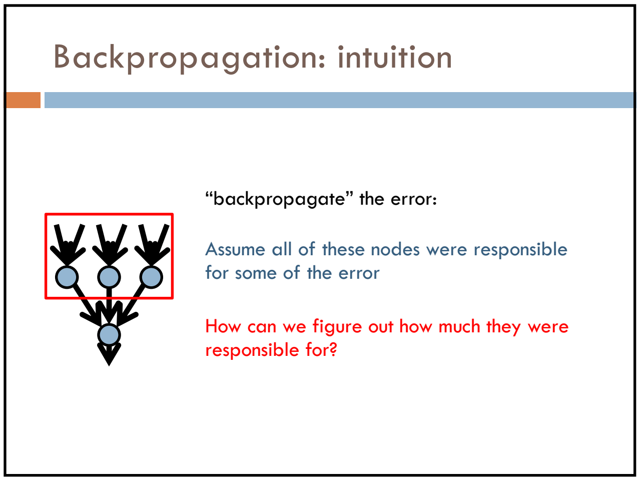
## Backpropagation: intuition

We can calculate the actual error here



## Backpropagation: intuition

Key idea: propagate the error back to this layer

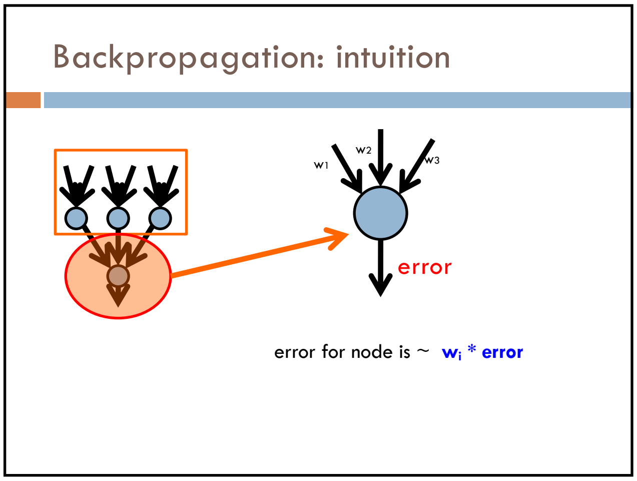


## Backpropagation: intuition

“backpropagate” the error:

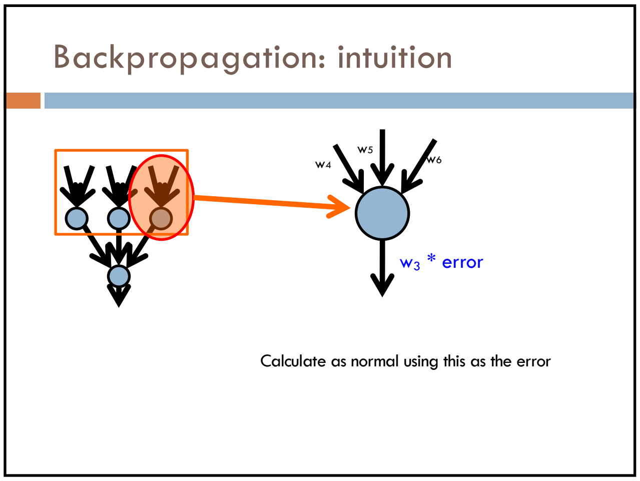
Assume all of these nodes were responsible for some of the error

How can we figure out how much they were responsible for?



## Backpropagation: intuition

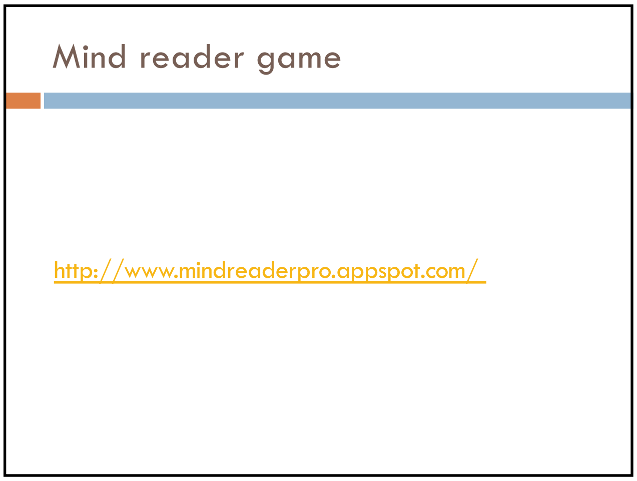
error for node is ~ $w_i$ * error



## Backpropagation: intuition

$w_3$ * error

Calculate as normal using this as the error



## Mind reader game

http://www.mindreaderpro.appspot.com/