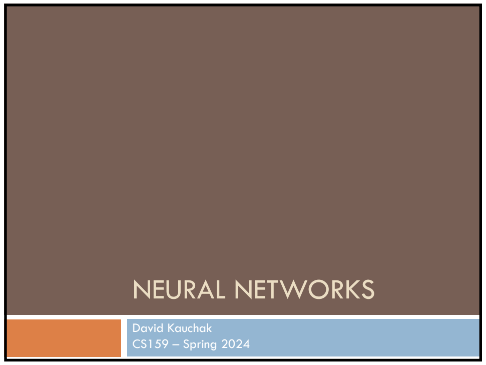
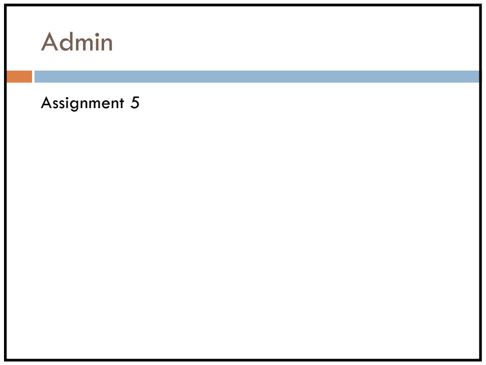


NEURAL NETWORKS

David Kauchak
CS159 – Spring 2024

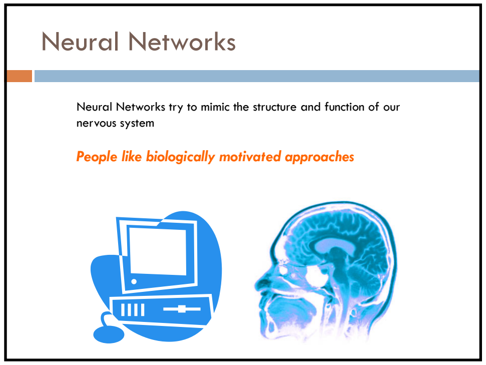
1



Admin

Assignment 5

2

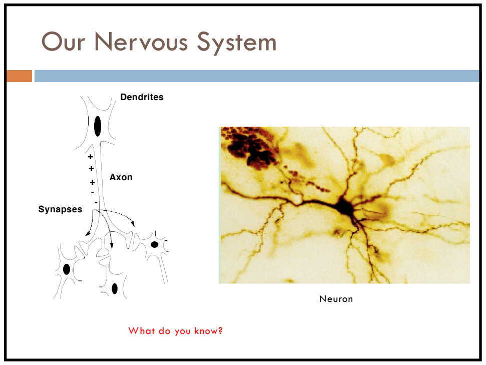


Neural Networks

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

People like biologically motivated approaches

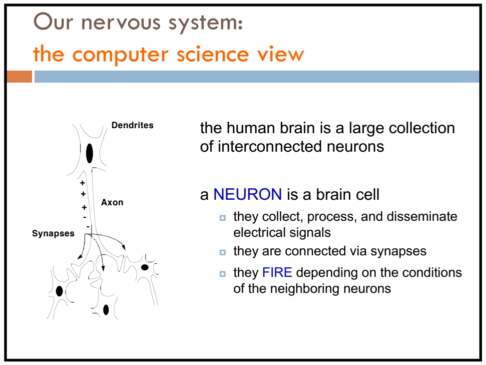
3



Our Nervous System

What do you know?

4



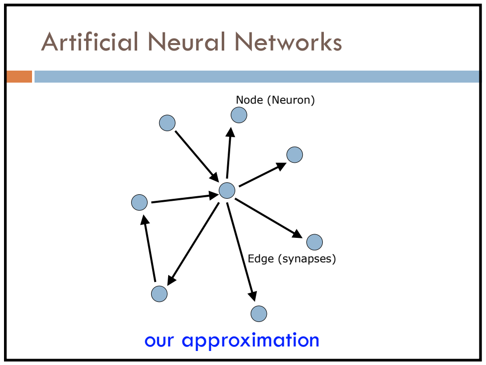
Our nervous system:
the computer science view

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

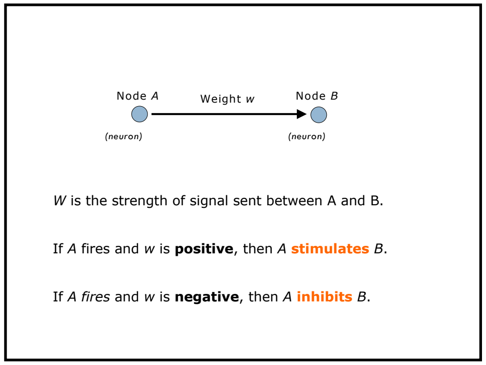
5



Artificial Neural Networks

our approximation

6

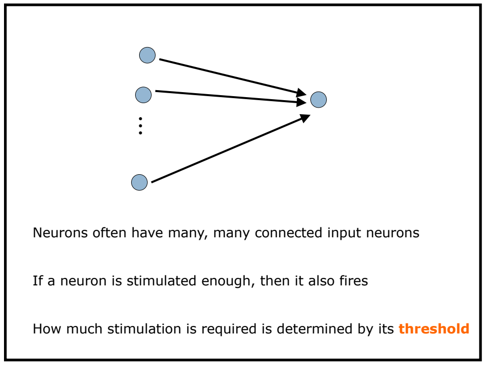


W is the strength of signal sent between A and B.

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

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

7

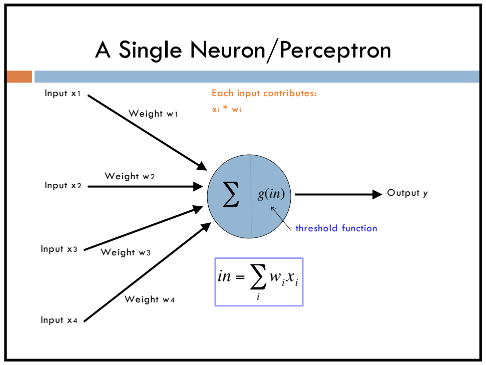


Neurons often have many, many connected input neurons

If a neuron is stimulated enough, then it also fires

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

8

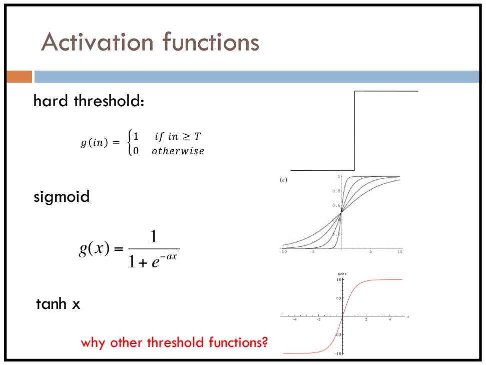


A Single Neuron/Perceptron

Each input contributes:
$x_i * w_i$

$$in = \sum_i w_i x_i$$

9



Activation functions

hard threshold:

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

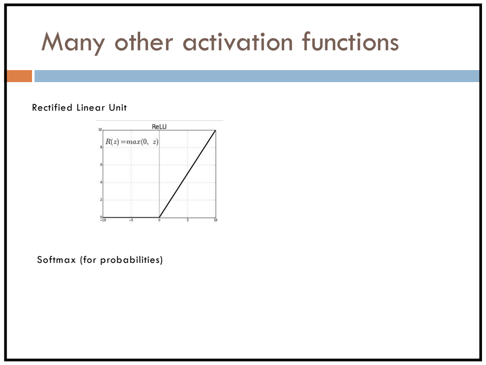
sigmoid

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

tanh x

why other threshold functions?

10

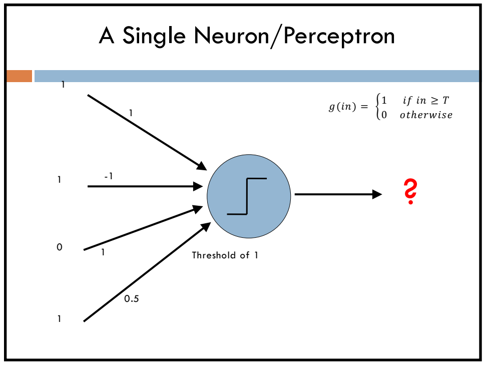


Many other activation functions

Rectified Linear Unit

Softmax (for probabilities)

11

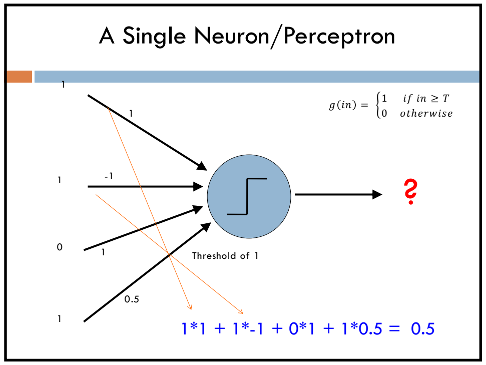


A Single Neuron/Perceptron

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

Threshold of 1

12

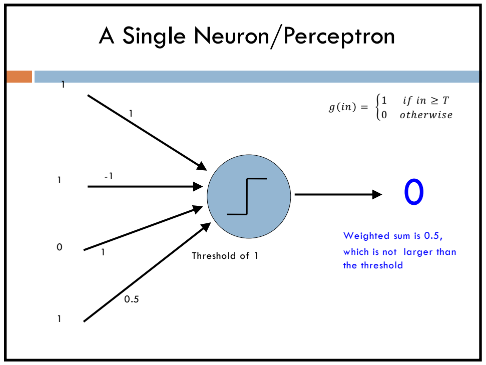


A Single Neuron/Perceptron

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

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

13

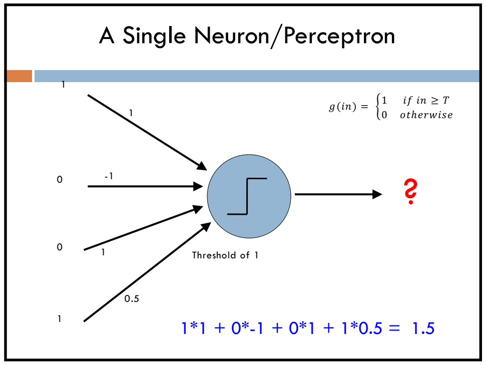


A Single Neuron/Perceptron

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

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

14

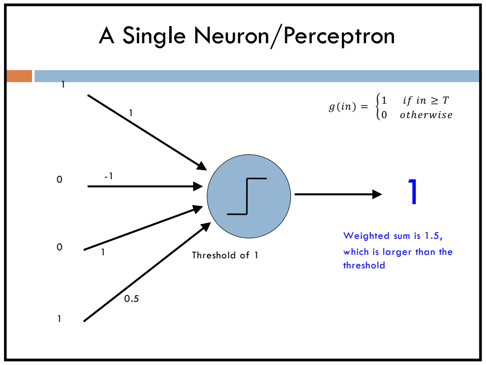


A Single Neuron/Perceptron

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

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

15

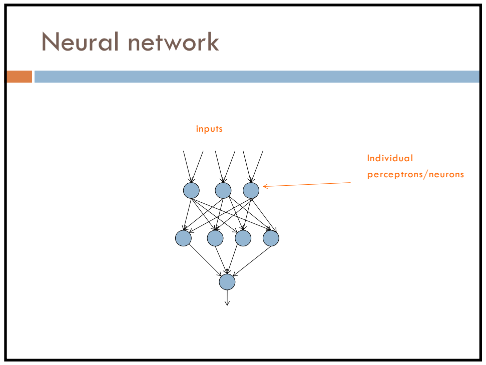


A Single Neuron/Perceptron

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

Weighted sum is 1.5, which is larger than the threshold

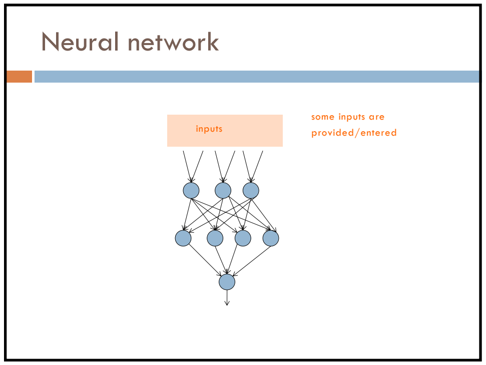
16



Neural network

Individual perceptrons/neurons

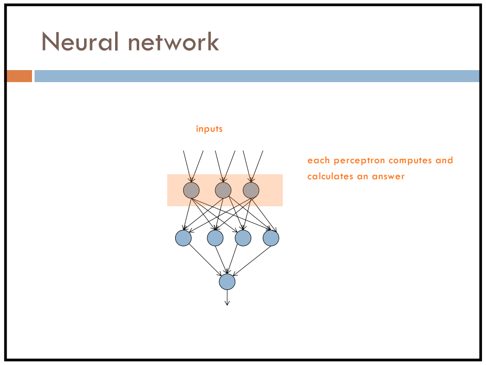
17



Neural network

some inputs are provided/entered

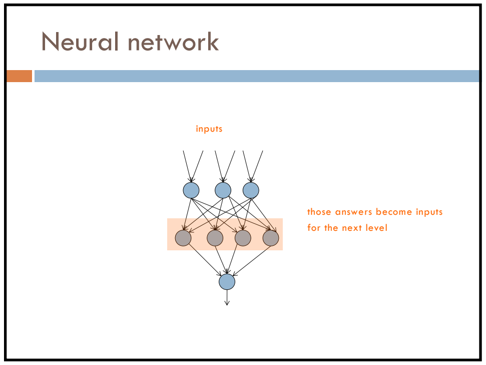
18



Neural network

each perceptron computes and calculates an answer

19

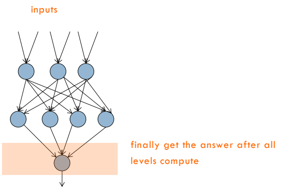


Neural network

those answers become inputs for the next level

20

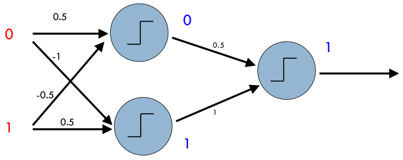
Neural network



finally get the answer after all levels compute

21

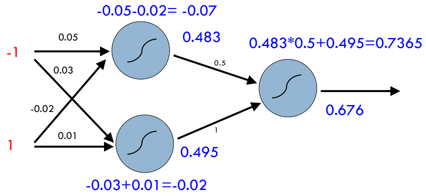
Computation (assume threshold 0)



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

22

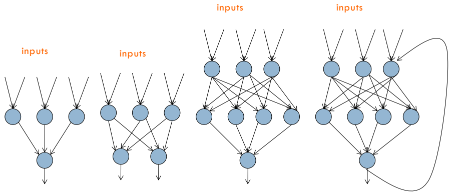
Computation



23

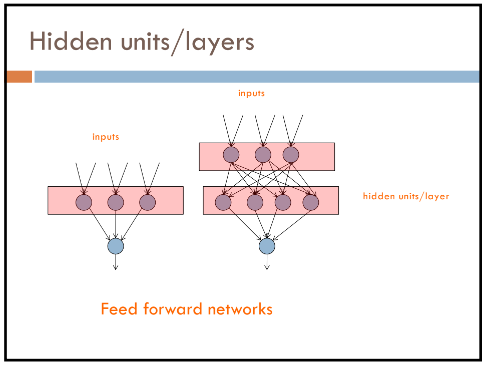
Neural networks

Different kinds/characteristics of networks



How are these different?

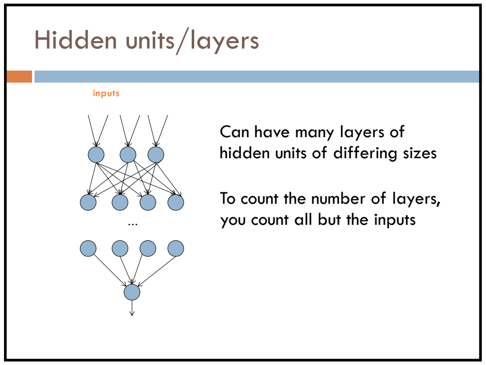
24



Hidden units/layers

Feed forward networks

25

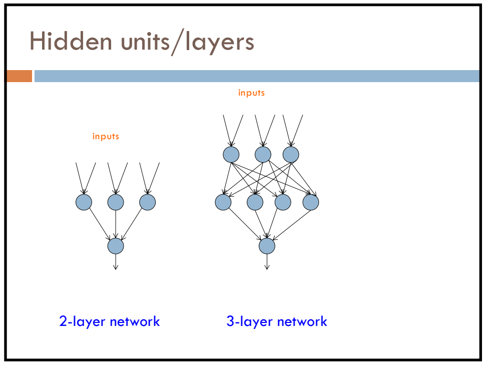


Hidden units/layers

Can have many layers of hidden units of differing sizes

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

26

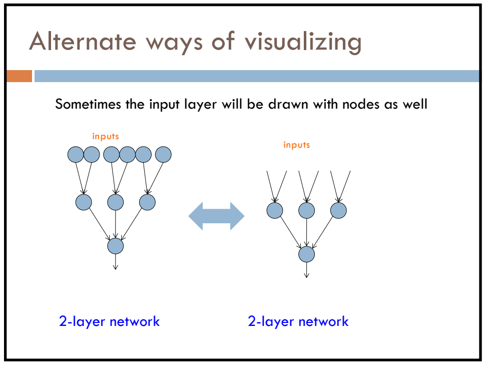


Hidden units/layers

2-layer network

3-layer network

27



Alternate ways of visualizing

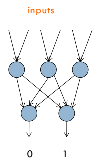
Sometimes the input layer will be drawn with nodes as well

2-layer network

2-layer network

28

Multiple outputs

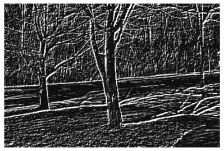


Can be used to model multiclass datasets or more interesting predictors, e.g. images

29

Multiple outputs

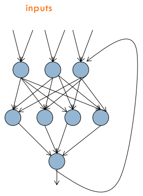
input



output
(edge detection)

30

Neural networks



Recurrent network

Output is fed back to input

Can support memory!

Good for temporal data

31

History of Neural Networks

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

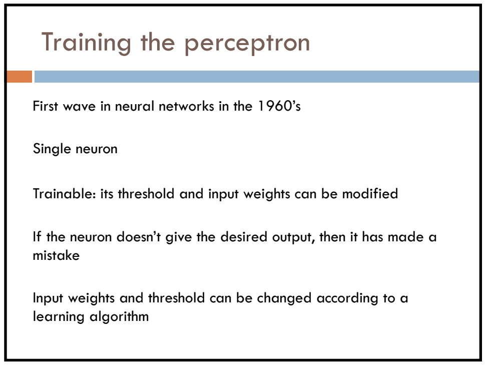
Hebb (1949) – Simple updating rule for learning

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

32



Training the perceptron

First wave in neural networks in the 1960's

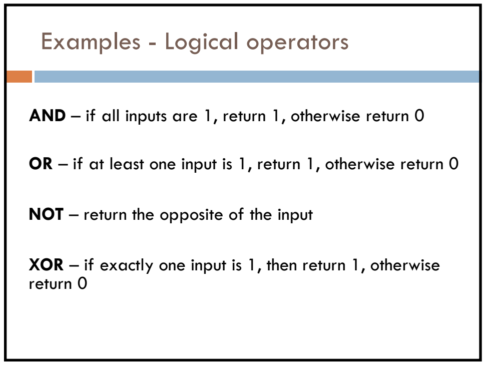
Single neuron

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

33



Examples - Logical operators

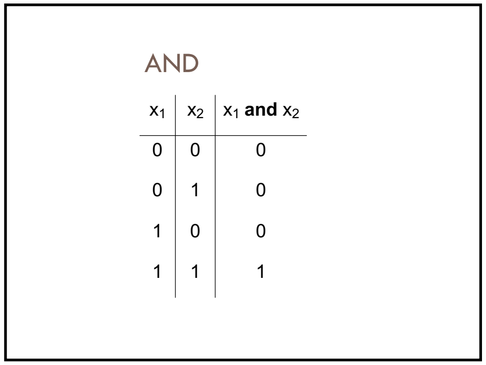
AND – if all inputs are 1, return 1, otherwise return 0

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

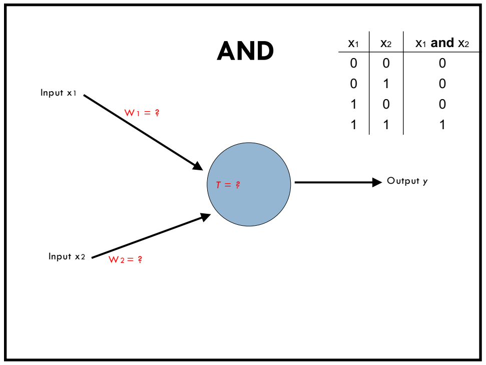
34



AND

x_1	x_2	x_1 **and** x_2
0	0	0
0	1	0
1	0	0
1	1	1

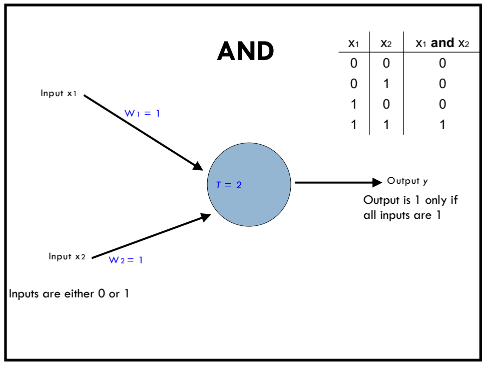
35



AND

x_1	x_2	x_1 **and** x_2
0	0	0
0	1	0
1	0	0
1	1	1

36



AND

x_1	x_2	x_1 **and** x_2
0	0	0
0	1	0
1	0	0
1	1	1

Input x_1

$w_1 = 1$

$T = 2$

Output y

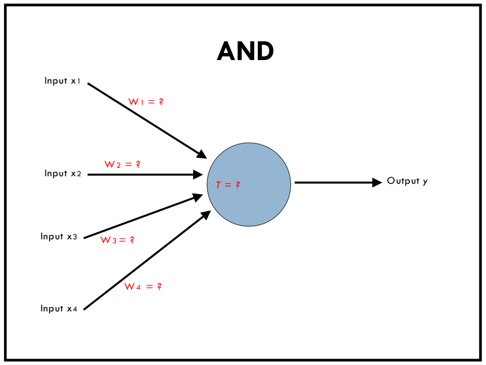
Output is 1 only if all inputs are 1

Input x_2

$w_2 = 1$

Inputs are either 0 or 1

37



AND

Input x_1

$w_1 = ?$

Input x_2

$w_2 = ?$

$T = ?$

Output y

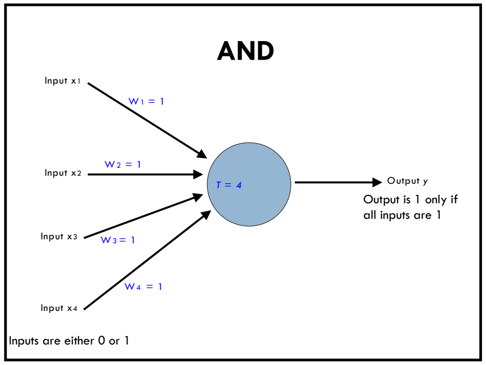
Input x_3

$w_3 = ?$

$w_4 = ?$

Input x_4

38



AND

Input x_1

$w_1 = 1$

$w_2 = 1$

Input x_2

$T = 4$

Output y

Output is 1 only if all inputs are 1

Input x_3

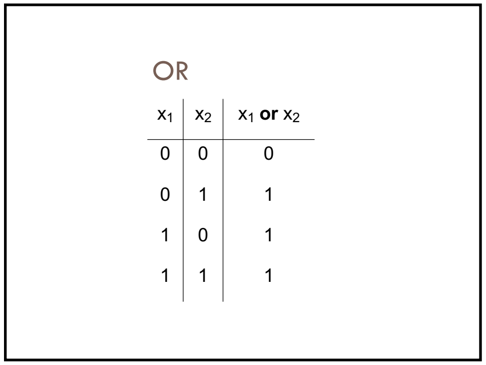
$w_3 = 1$

$w_4 = 1$

Input x_4

Inputs are either 0 or 1

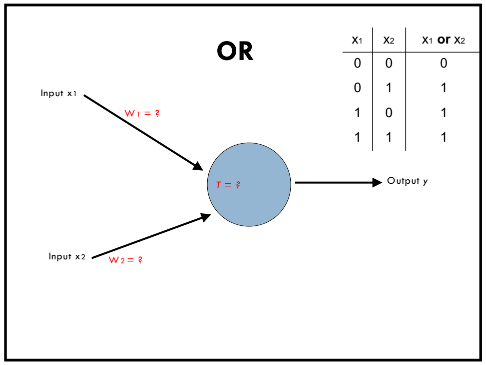
39



OR

x_1	x_2	x_1 **or** x_2
0	0	0
0	1	1
1	0	1
1	1	1

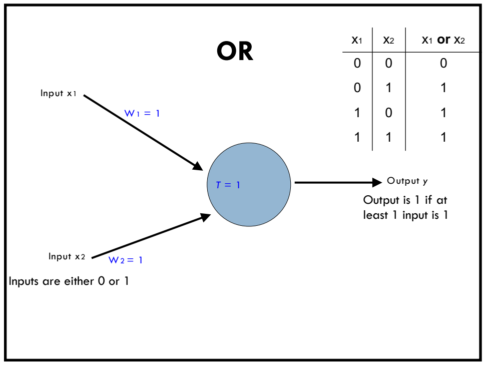
40



OR

x_1	x_2	x_1 **or** x_2
0	0	0
0	1	1
1	0	1
1	1	1

41



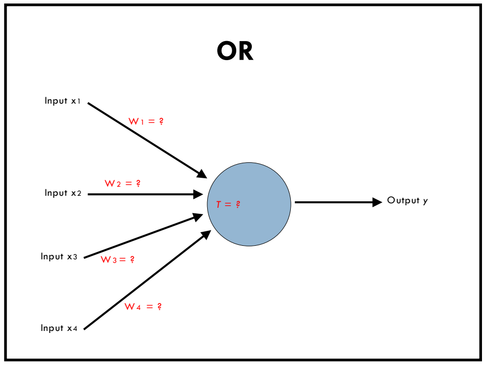
OR

x_1	x_2	x_1 **or** x_2
0	0	0
0	1	1
1	0	1
1	1	1

Output is 1 if at least 1 input is 1

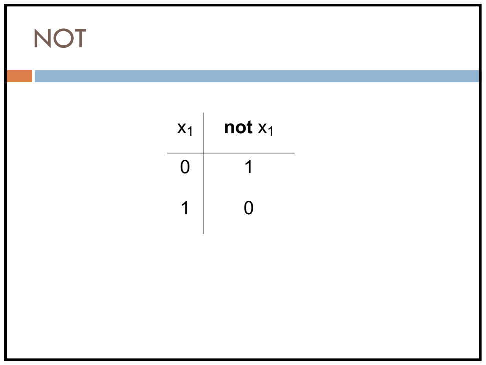
Inputs are either 0 or 1

42



OR

43

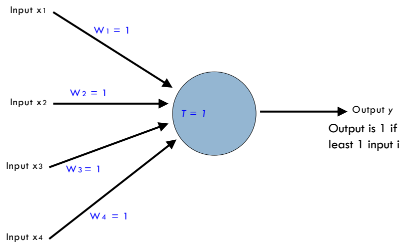


NOT

x_1	**not** x_1
0	1
1	0

44

OR



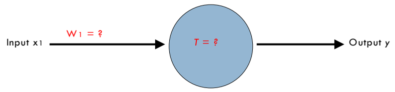
Output is 1 if at least 1 input is 1

Inputs are either 0 or 1

45

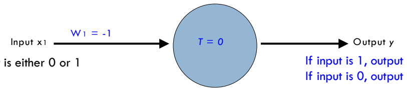
NOT

x_1	not x_1
0	1
1	0



46

NOT



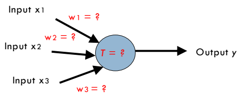
Input is either 0 or 1

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

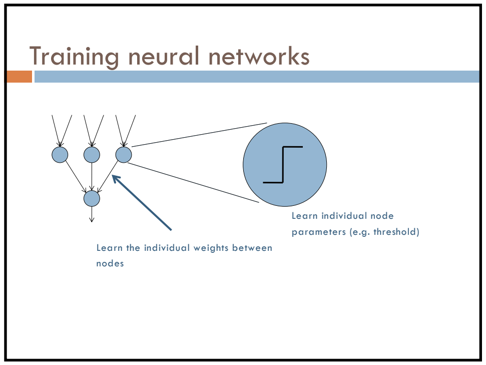
47

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



48

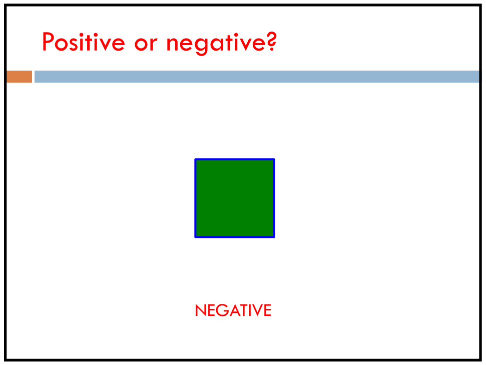


Training neural networks

Learn individual node parameters (e.g. threshold)

Learn the individual weights between nodes

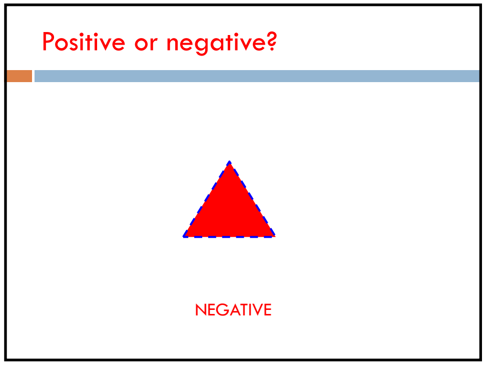
49



Positive or negative?

NEGATIVE

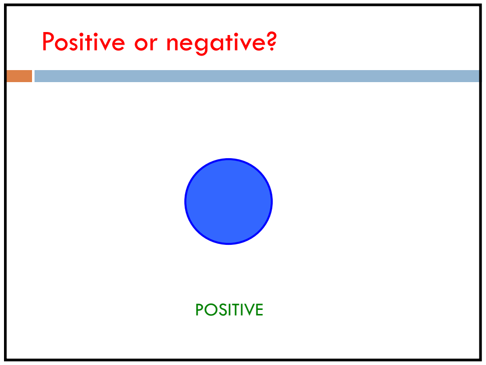
50



Positive or negative?

NEGATIVE

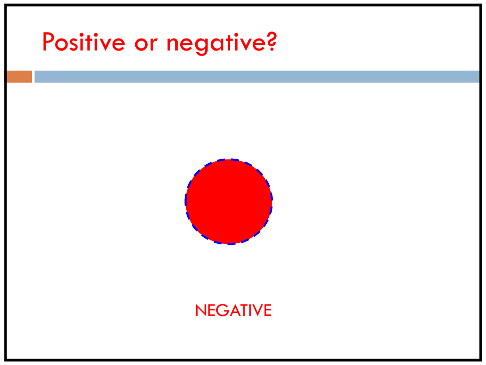
51



Positive or negative?

POSITIVE

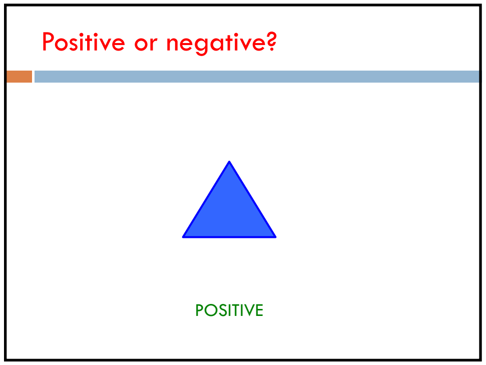
52



Positive or negative?

NEGATIVE

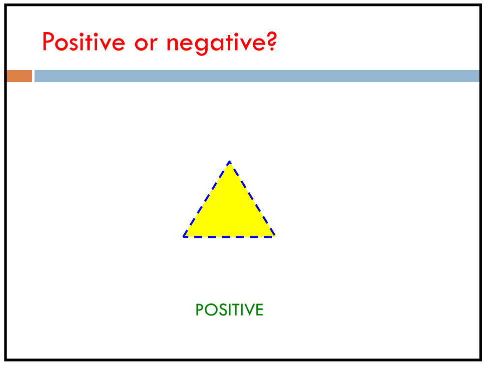
53



Positive or negative?

POSITIVE

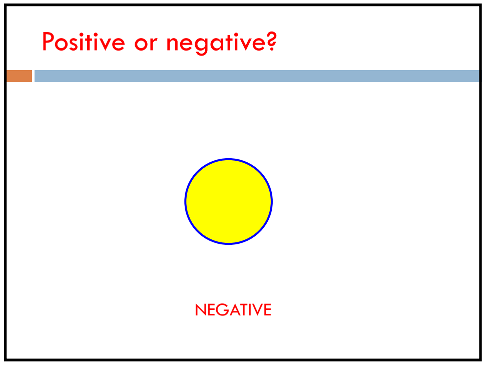
54



Positive or negative?

POSITIVE

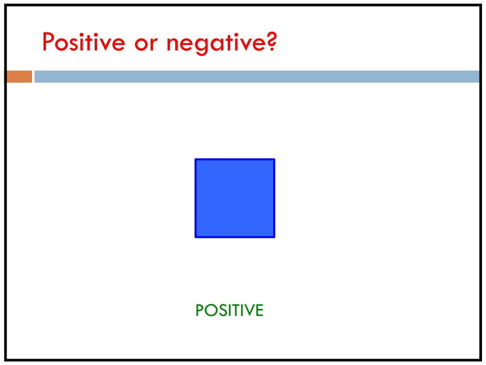
55



Positive or negative?

NEGATIVE

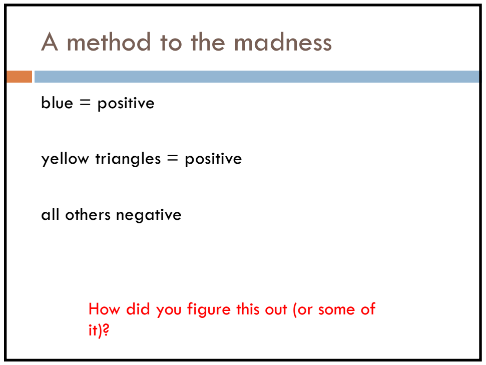
56



Positive or negative?

POSITIVE

57



A method to the madness

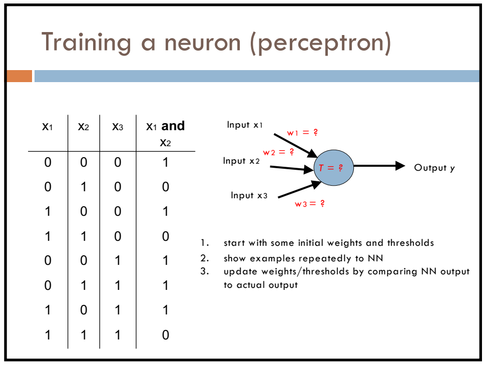
blue = positive

yellow triangles = positive

all others negative

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

58



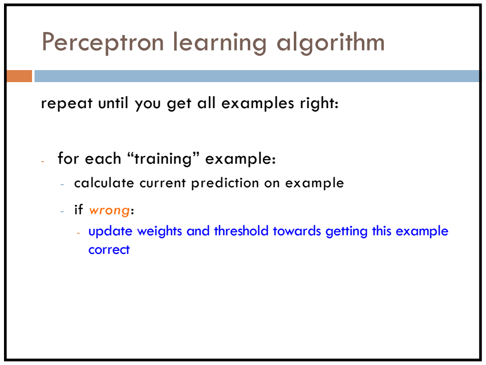
Training a neuron (perceptron)

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

Input x1
w1 = ?
w2 = ?
Input x2
T = ?
Output y
Input x3
w3 = ?

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

59

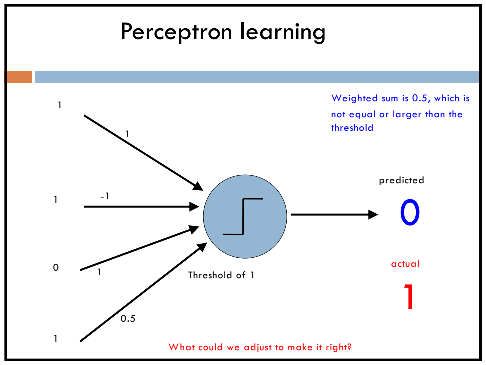


Perceptron learning algorithm

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

60

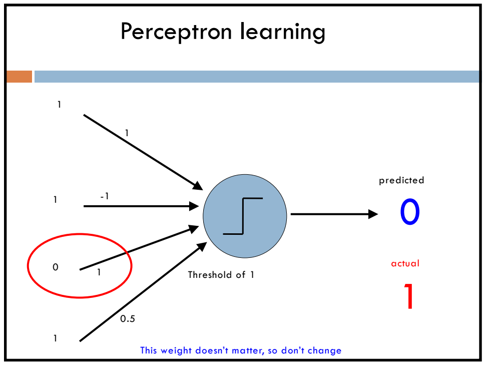


Perceptron learning

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

What could we adjust to make it right?

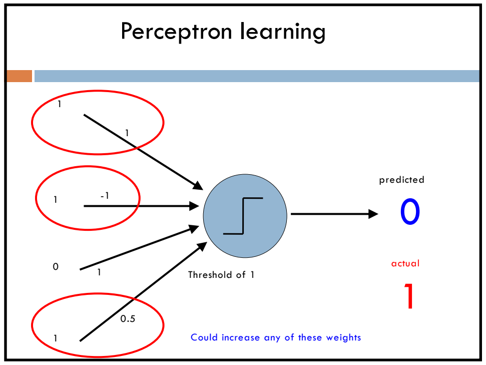
61



Perceptron learning

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

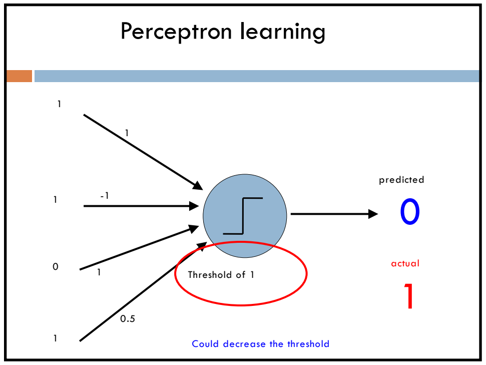
62



Perceptron learning

Could increase any of these weights

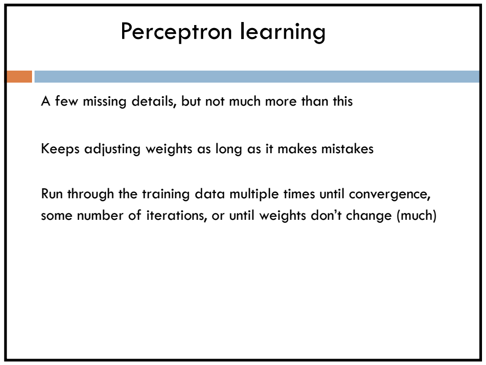
63



Perceptron learning

Could decrease the threshold

64



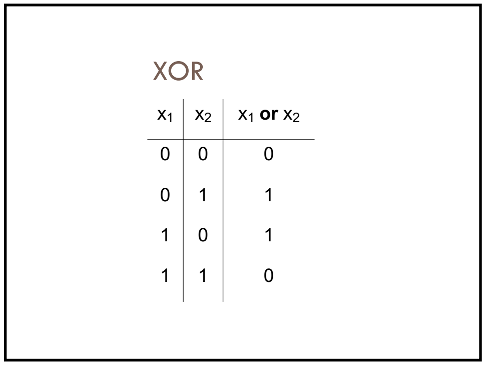
Perceptron learning

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)

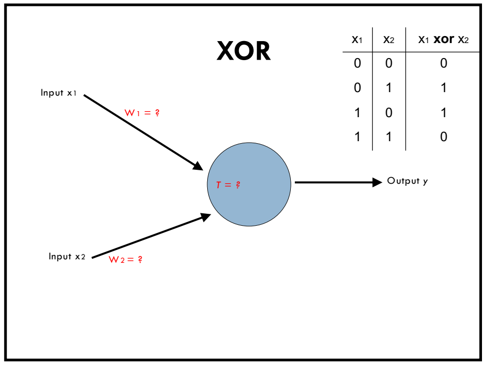
65



XOR

x_1	x_2	x_1 **or** x_2
0	0	0
0	1	1
1	0	1
1	1	0

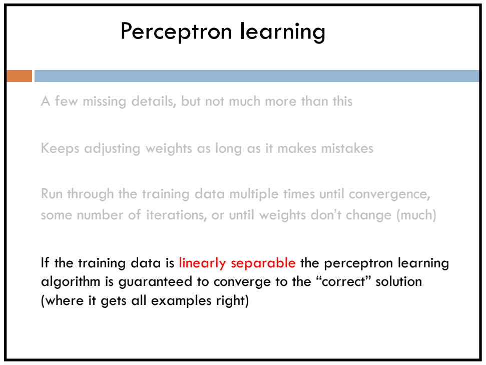
66



XOR

x_1	x_2	x_1 **xor** x_2
0	0	0
0	1	1
1	0	1
1	1	0

67



Perceptron learning

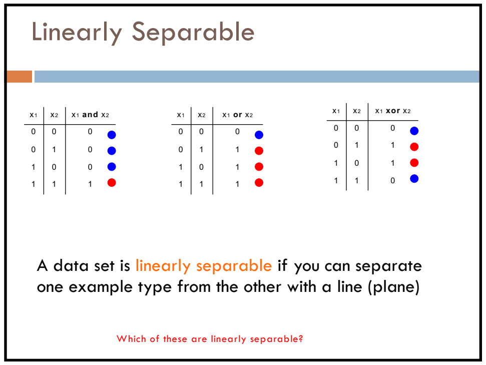
A few missing details, but not much more than this

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)

68



Linearly Separable

x_1	x_2	x_1 **and** x_2	
0	0	0	🔵
0	1	0	🔵
1	0	0	🔵
1	1	1	🔴

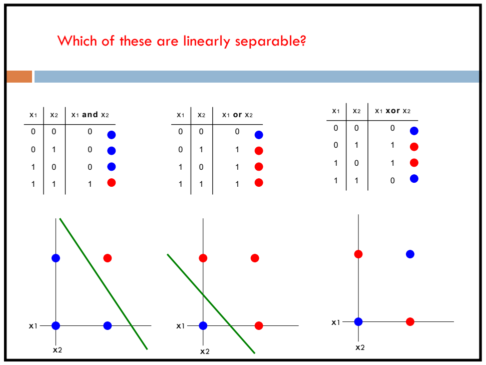
x_1	x_2	x_1 **or** x_2	
0	0	0	🔵
0	1	1	🔴
1	0	1	🔴
1	1	1	🔴

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 with a line (plane)

Which of these are linearly separable?

69



Which of these are linearly separable?

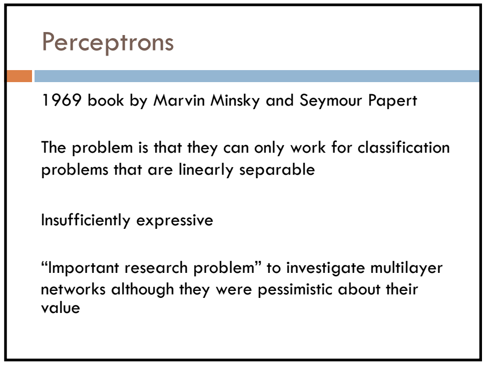
x_1	x_2	x_1 **and** x_2	
0	0	0	🔵
0	1	0	🔵
1	0	0	🔵
1	1	1	🔴

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	🔵

x1
x2
x1
x2
x1
x2

70



Perceptrons

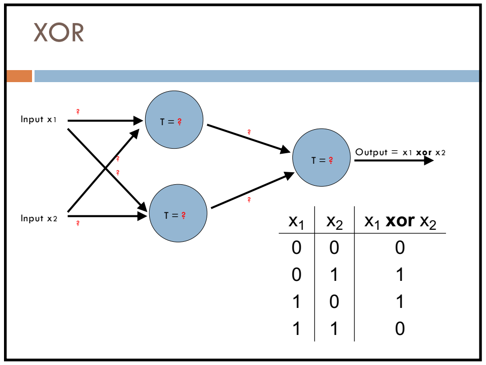
1969 book by Marvin Minsky and Seymour Papert

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

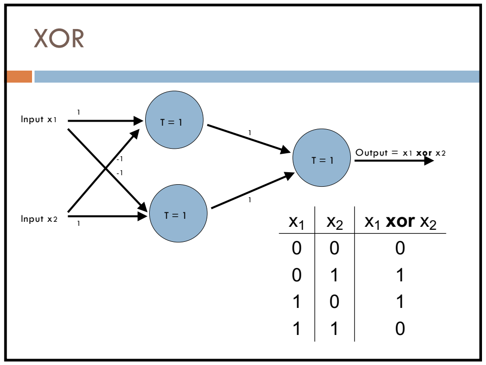
71



XOR

x_1	x_2	x_1 **xor** x_2
0	0	0
0	1	1
1	0	1
1	1	0

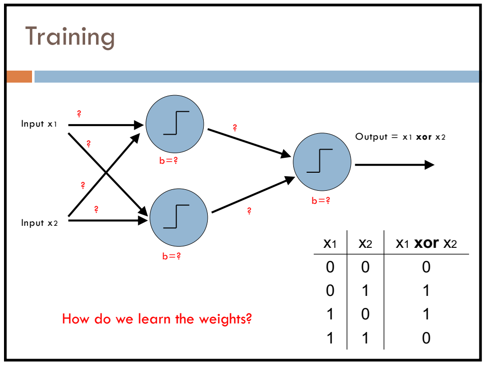
72



XOR

x_1	x_2	x_1 **xor** x_2
0	0	0
0	1	1
1	0	1
1	1	0

73

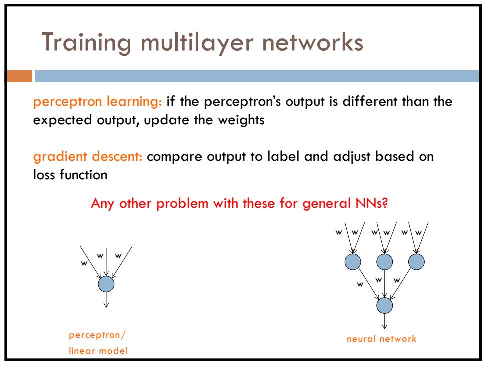


Training

How do we learn the weights?

x1	x2	x1 **xor** x2
0	0	0
0	1	1
1	0	1
1	1	0

74



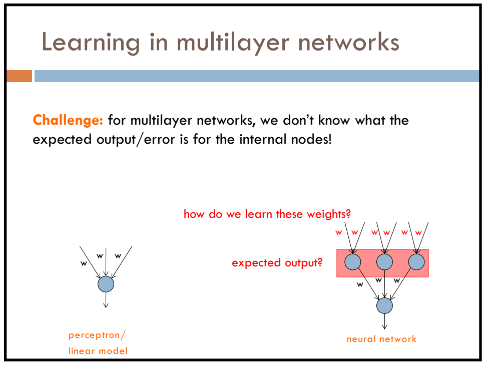
Training multilayer networks

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?

75



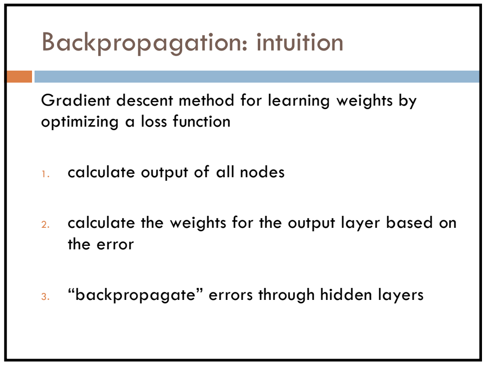
Learning in multilayer networks

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

how do we learn these weights?

expected output?

76

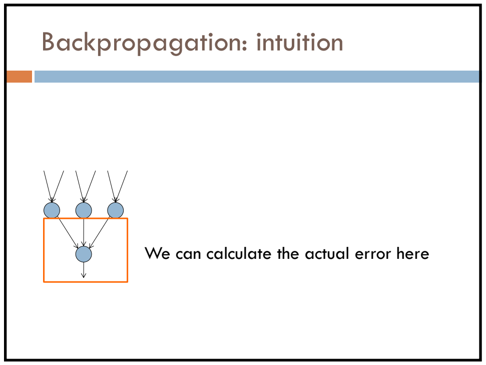


Backpropagation: intuition

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

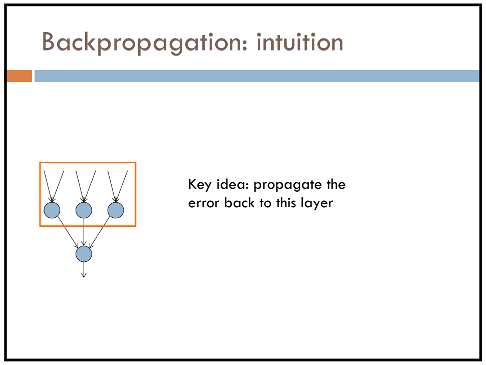
77



Backpropagation: intuition

We can calculate the actual error here

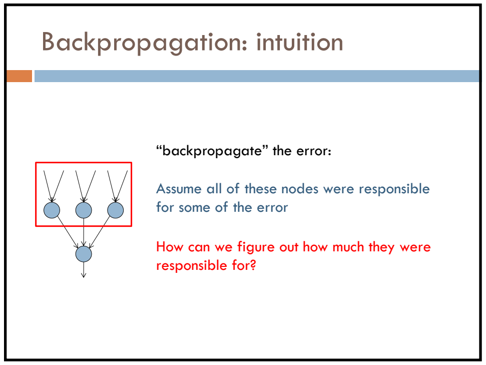
78



Backpropagation: intuition

Key idea: propagate the error back to this layer

79



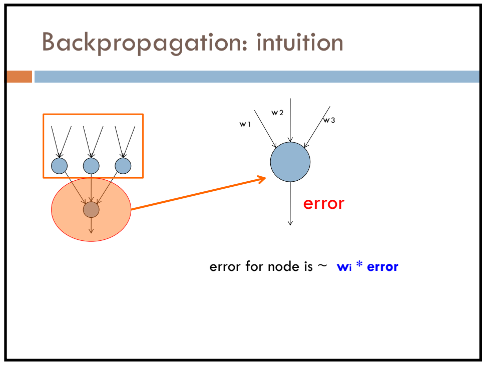
Backpropagation: intuition

“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?

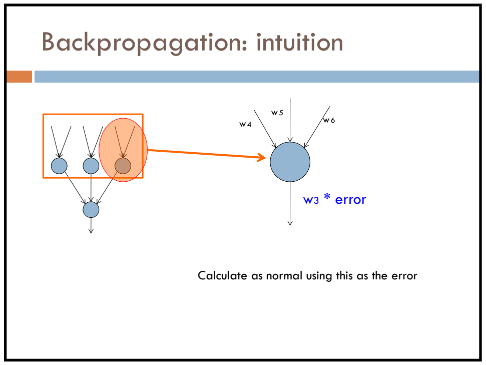
80



Backpropagation: intuition

error for node is ~ w_i * error

81



Backpropagation: intuition

w_3 * error

Calculate as normal using this as the error

82