

NEURAL NETWORKS

David Kauchak
CS158 – Fall 2016

Admin

Assignment 7

Class 11/22

Schedule for the rest of the semester

Perceptron learning algorithm

repeat until convergence (or for some # of iterations):

for each training example $(f_1, f_2, \dots, f_n, \text{label})$:

$$\text{prediction} = b + \sum_{i=1}^n w_i f_i$$

if $\text{prediction} * \text{label} \leq 0$: // they don't agree

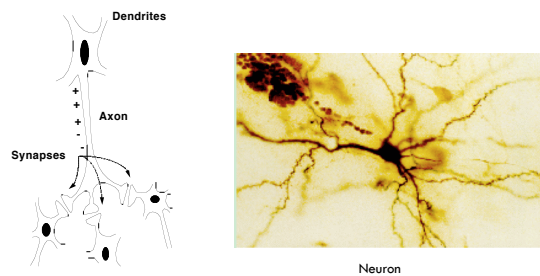
for each w_i :

$$w_i = w_i + f_i * \text{label}$$

$$b = b + \text{label}$$

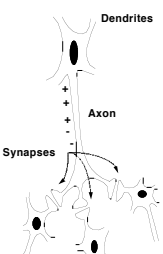
Why is it called the "perceptron" learning algorithm if what it learns is a line? Why not "line learning" algorithm?

Our Nervous System



What do you know?

Our nervous system: the computer science view

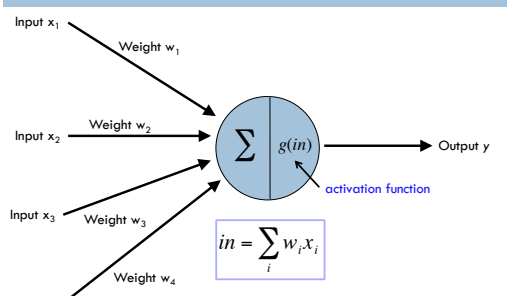


the human brain is a large collection of interconnected neurons

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

A neuron/perceptron



Input x_1 Weight w_1

Input x_2 Weight w_2

Input x_3 Weight w_3

Input x_4 Weight w_4

$in = \sum_i w_i x_i$

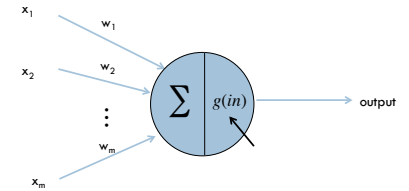
activation function $g(in)$

Output y

How is this a linear classifier (i.e. perceptron)?

Hard threshold = linear classifier


hard threshold:

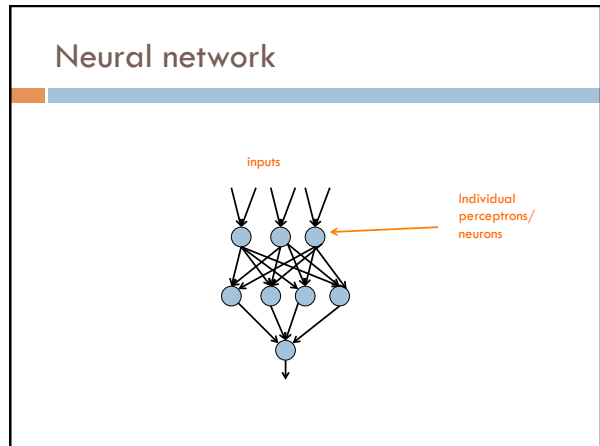
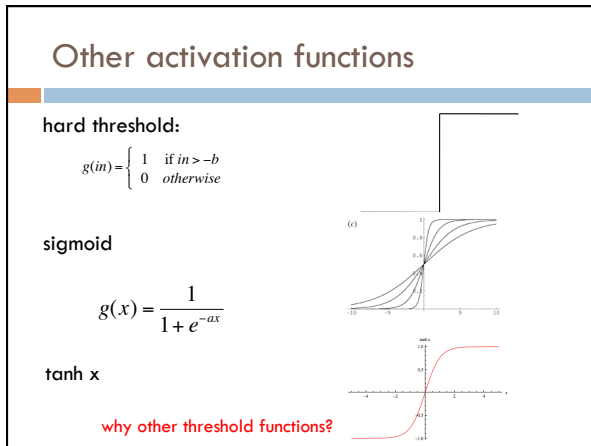
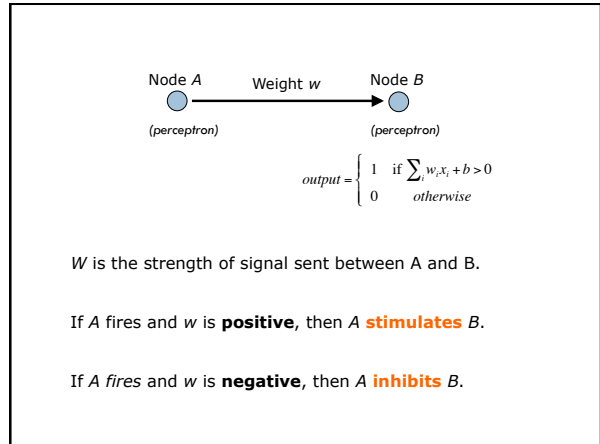
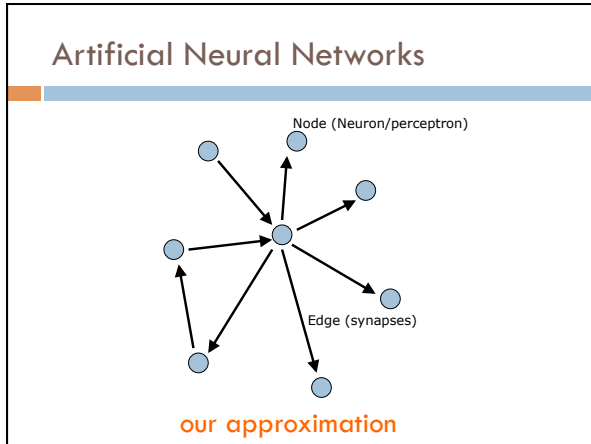
$$g(in) = \begin{cases} 1 & \text{if } in > -b \\ 0 & \text{otherwise} \end{cases} \quad output = \begin{cases} 1 & \text{if } \sum w_i x_i + b > 0 \\ 0 & \text{otherwise} \end{cases}$$


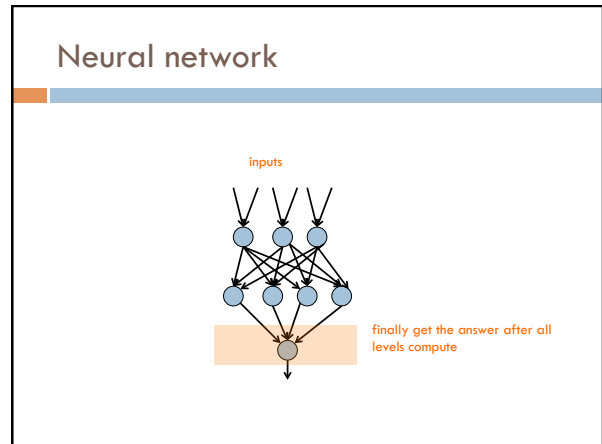
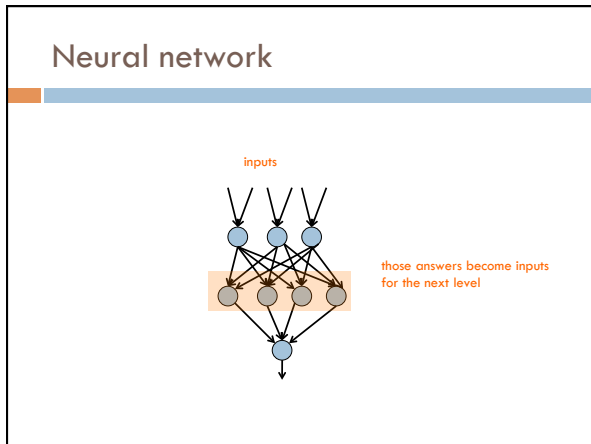
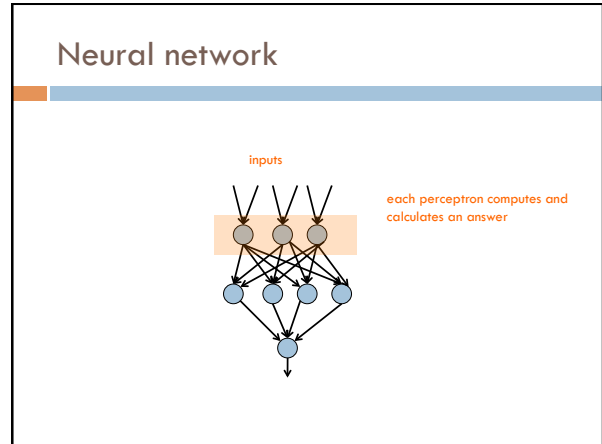
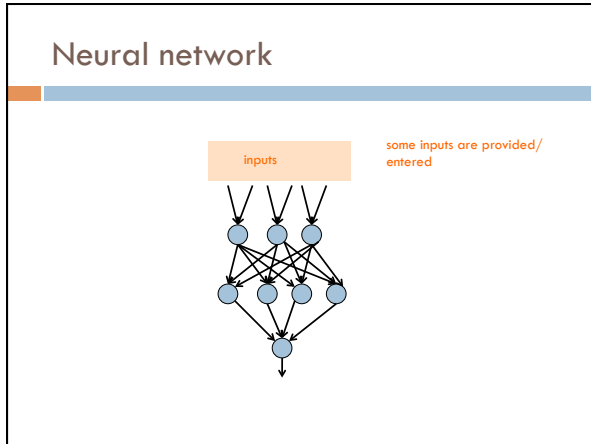
Neural Networks

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

People like biologically motivated approaches



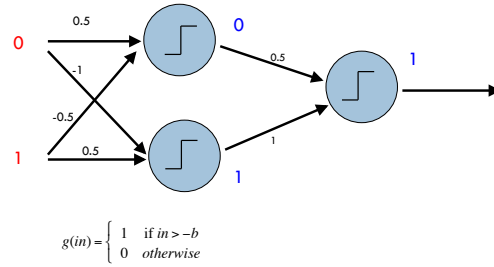




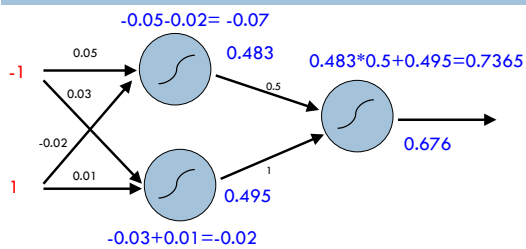
Activation spread

<http://www.youtube.com/watch?v=Yq7d4ROvZ6I>

Computation (assume 0 bias)

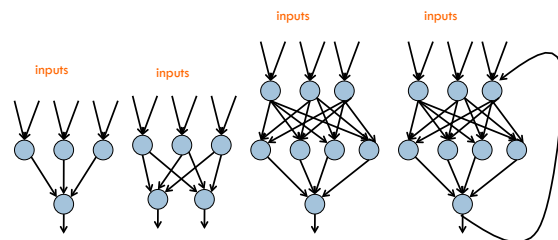


Computation

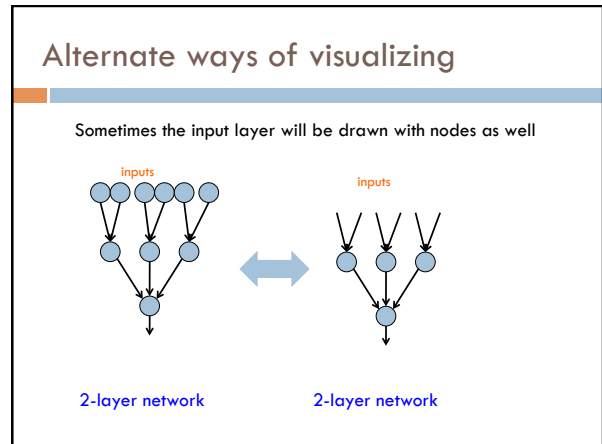
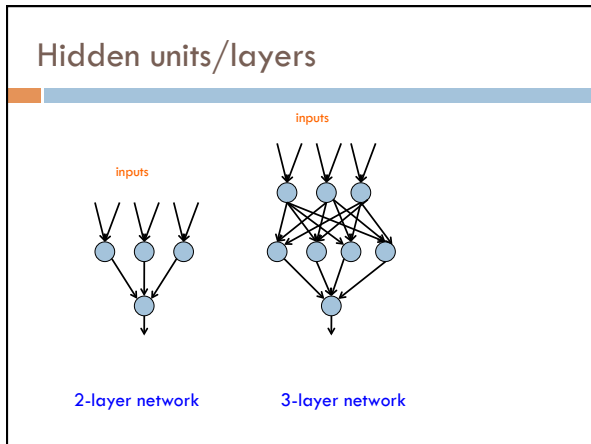
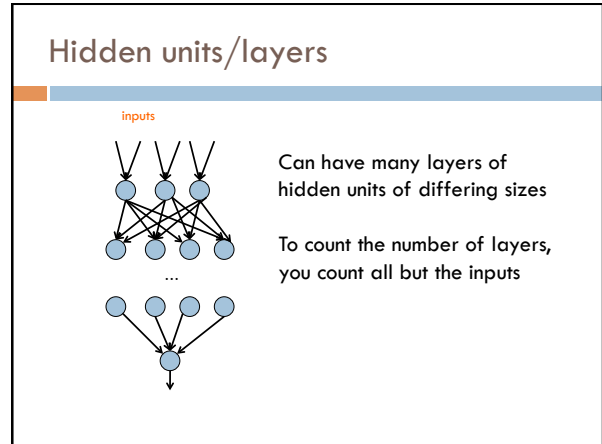
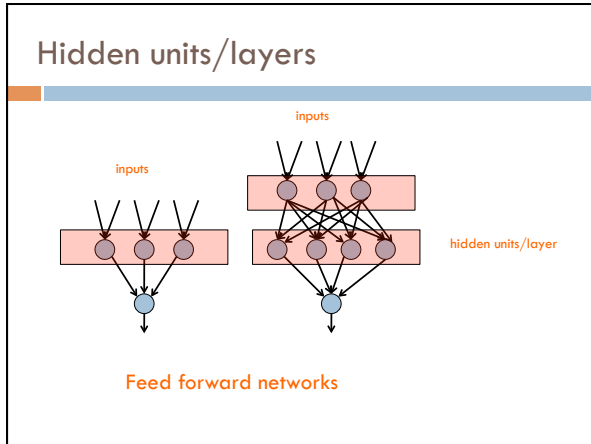


Neural networks

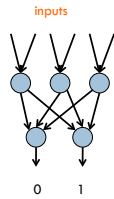
Different kinds/characteristics of networks



How are these different?



Multiple outputs



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

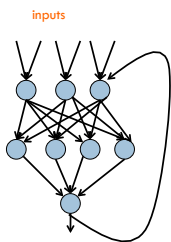
Multiple outputs



input

output
(edge detection)

Neural networks



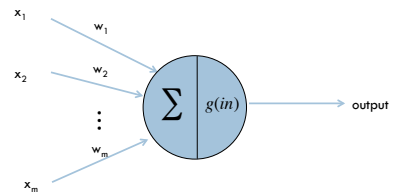
Recurrent network

Output is fed back to input

Can support memory!

Good for temporal data

NN decision boundary



What does the decision boundary of a perceptron look like?

Line (linear set of weights)

NN decision boundary

What does the decision boundary of a 2-layer network look like?
Is it linear?
What types of things can and can't it model?

XOR

Output = $x_1 \text{ XOR } x_2$

x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

$$\text{output} = \begin{cases} 1 & \text{if } \sum_i w_i x_i + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

XOR

Output = $x_1 \text{ XOR } x_2$

x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

$$\text{output} = \begin{cases} 1 & \text{if } \sum_i w_i x_i + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

What does the decision boundary look like?

Output = $x_1 \text{ XOR } x_2$

x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

What does the decision boundary look like?

Input x_1 (1, -1) and Input x_2 (-1, 1) feed into two hidden nodes with bias $b=-0.5$. The output node has bias $b=-0.5$ and produces Output = $x_1 \text{ XOR } x_2$.

x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

What does this perceptron's decision boundary look like?

NN decision boundary

Input x_1 (-1) and Input x_2 (1) feed into a single perceptron with bias $b=-0.5$.

Let $x_2 = 0$, then:
 $-x_1 - 0.5 = 0$
 $x_1 = -0.5$

(without the bias)

NN decision boundary

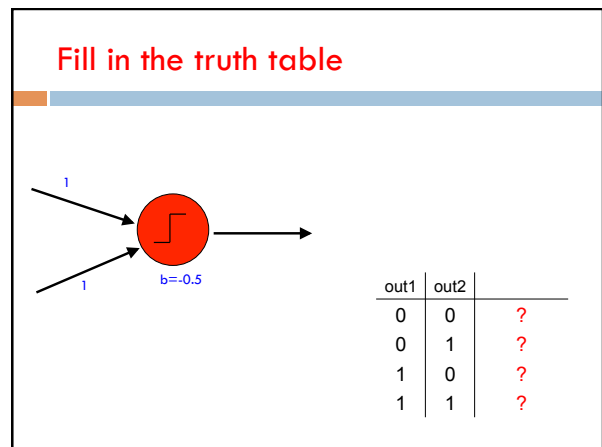
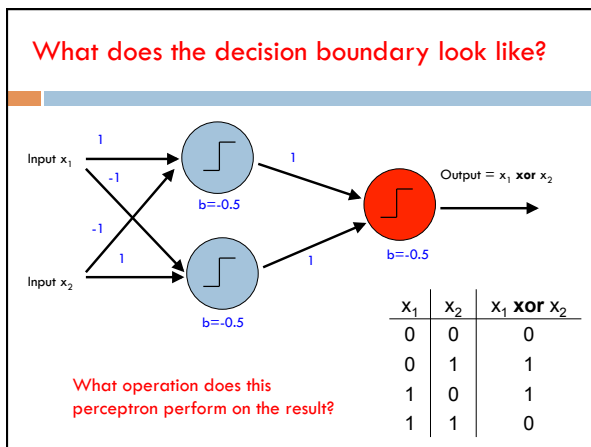
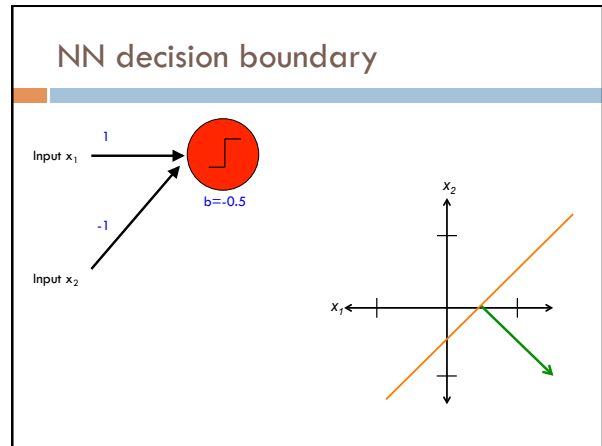
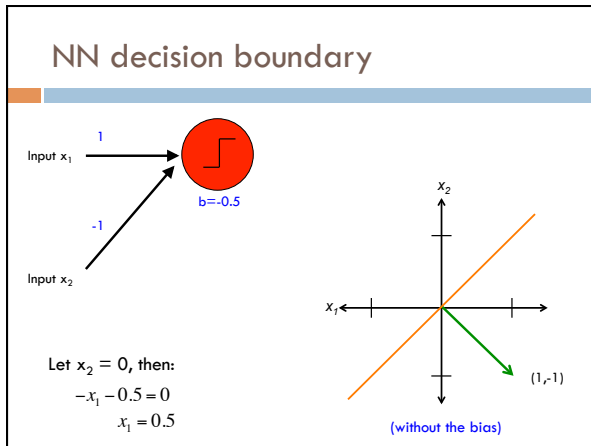
Input x_1 (-1) and Input x_2 (1) feed into a single perceptron with bias $b=-0.5$.

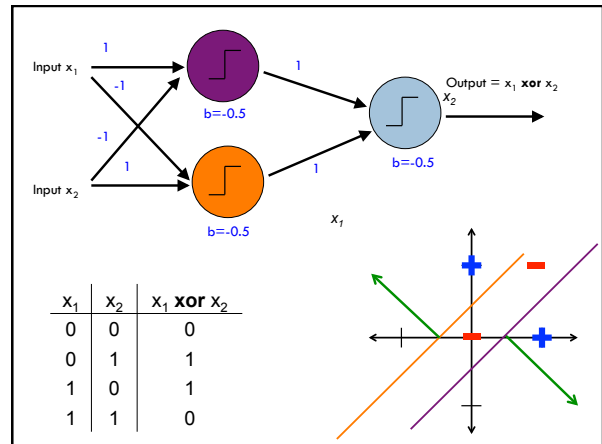
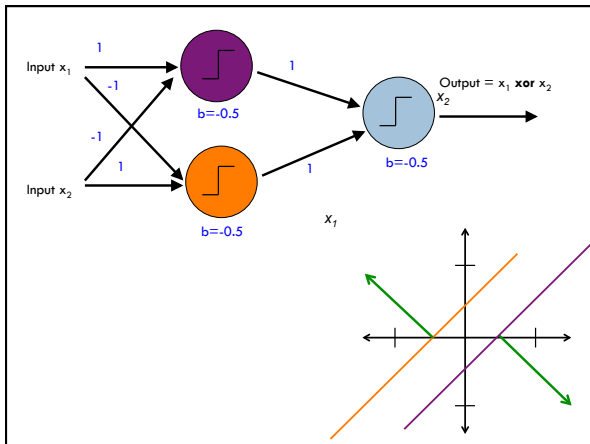
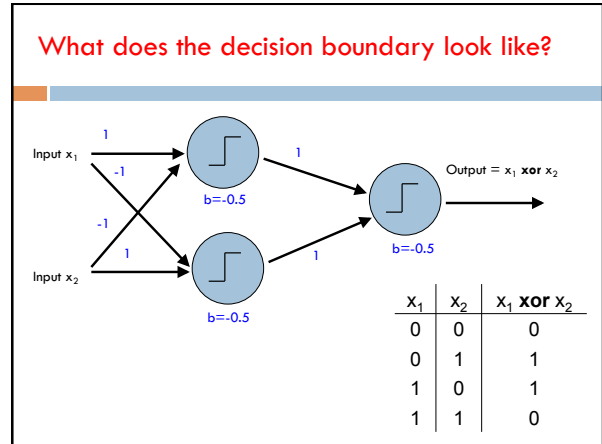
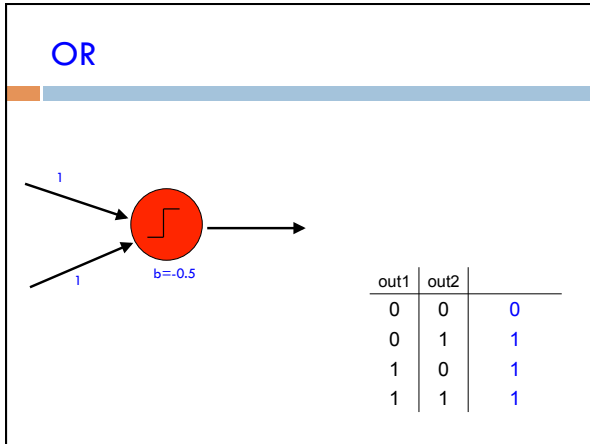
What does the decision boundary look like?

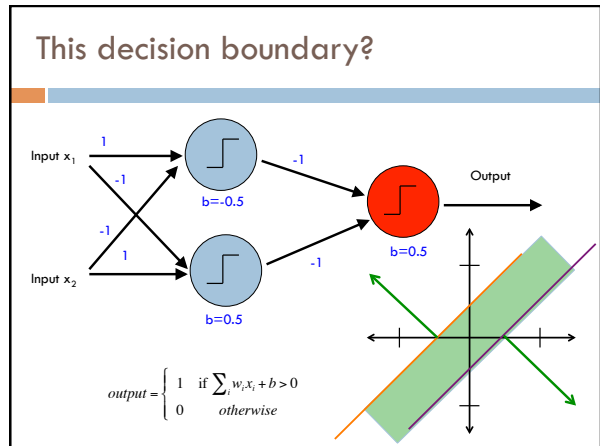
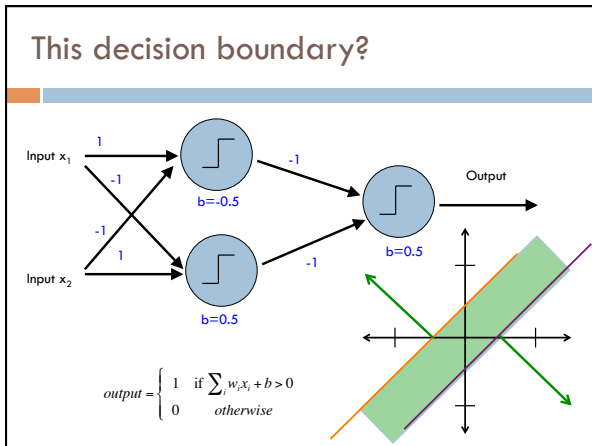
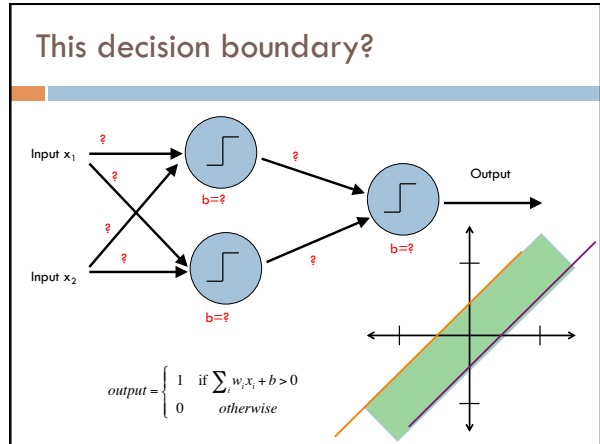
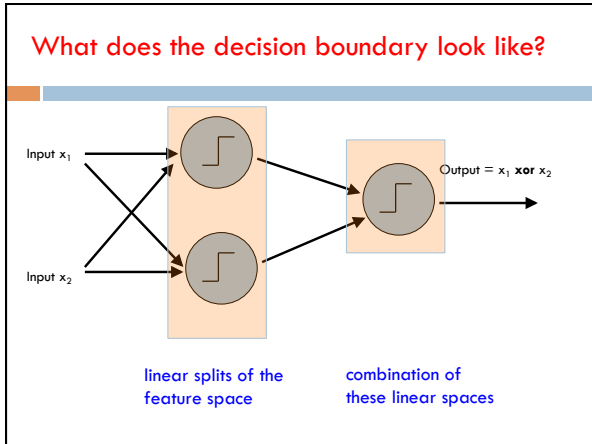
Input x_1 (1, -1) and Input x_2 (-1, 1) feed into two hidden nodes with bias $b=-0.5$. The output node has bias $b=-0.5$ and produces Output = $x_1 \text{ XOR } x_2$.

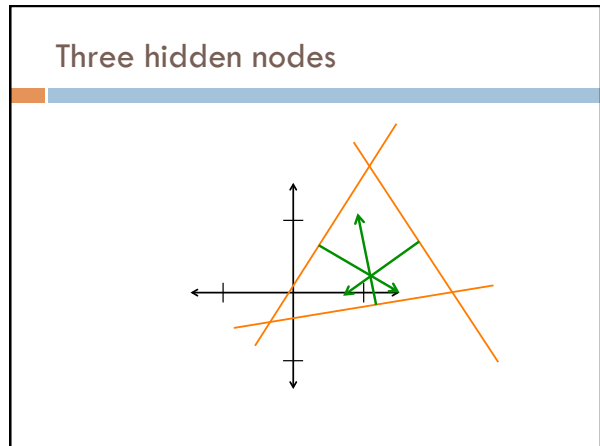
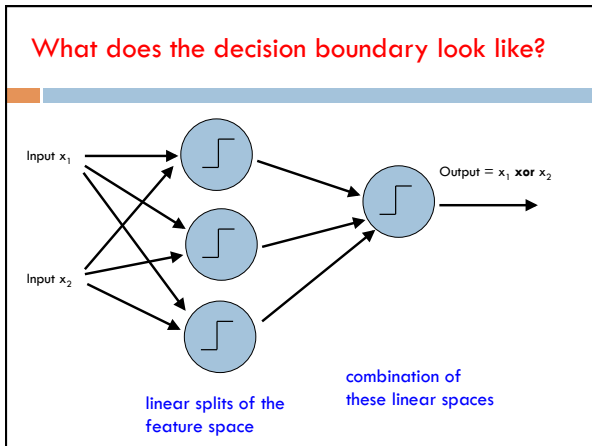
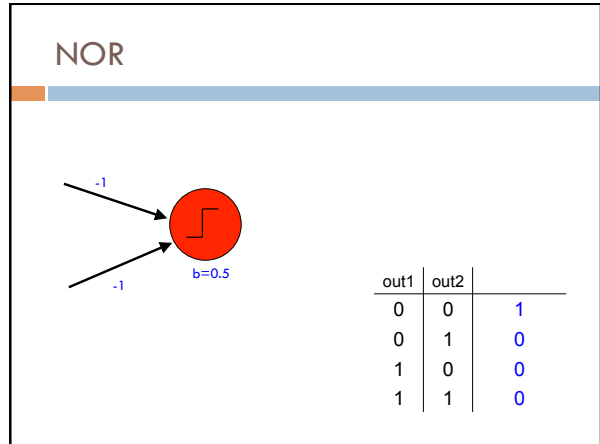
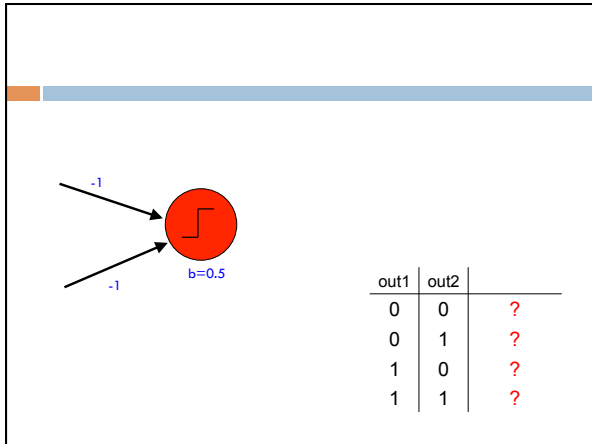
x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

What does this perceptron's decision boundary look like?









NN decision boundaries

Theorem 9 (Two-Layer Networks are Universal Function Approximators). *Let F be a continuous function on a bounded subset of D -dimensional space. Then there exists a two-layer neural network \hat{F} with a finite number of hidden units that approximate F arbitrarily well. Namely, for all x in the domain of F , $|F(x) - \hat{F}(x)| < \epsilon$.*

'Or, in colloquial terms "two-layer networks can approximate any function."

NN decision boundaries

For DT, as the tree gets larger, the model gets more complex

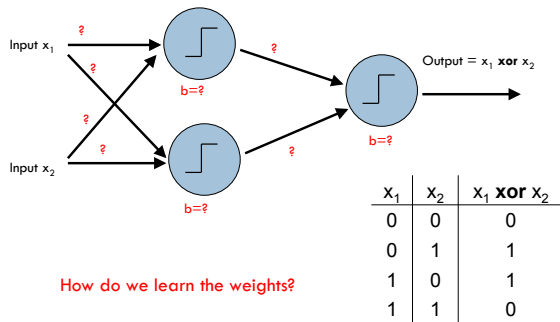
The same is true for neural networks: more hidden nodes = more complexity

Adding more layers adds even more complexity (and much more quickly)

Good rule of thumb:

$$\text{number of 2-layer hidden nodes} \leq \frac{\text{number of examples}}{\text{number of dimensions}}$$

Training



Training multilayer networks

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

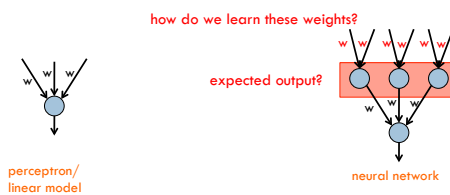
gradient descent: compare output to label and adjust based on loss function

Any other problem with these for general NNs?



Learning in multilayer networks

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

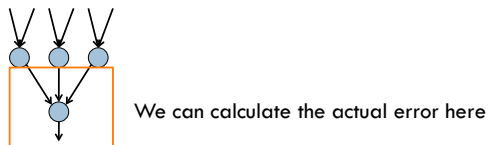


Backpropagation: intuition

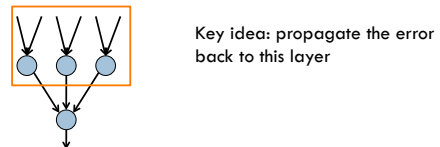
Gradient descent method for learning weights by optimizing a loss function

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

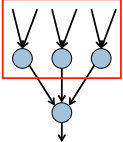
Backpropagation: intuition



Backpropagation: intuition



Backpropagation: intuition

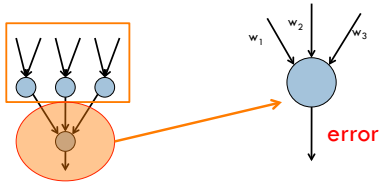


"backpropagate" the error:

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

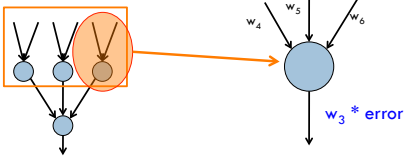
How can we figure out how much they were responsible for?

Backpropagation: intuition



error for node is: $w_i * \text{error}$

Backpropagation: intuition



Calculate as normal using this as the error

Backpropagation: the details

Gradient descent method for learning weights by optimizing a **loss function**

1. calculate output of all nodes
2. calculate the updates directly for the output layer
3. "backpropagate" errors through hidden layers

What loss function?

Backpropagation: the details

Gradient descent method for learning weights by optimizing a **loss function**

1. calculate output of all nodes
2. calculate the updates directly for the output layer
3. “backpropagate” errors through hidden layers

$$loss = \sum_x \frac{1}{2} (y - \hat{y})^2 \quad \text{squared error}$$