NEURAL NETWORKS

David Kauchak
CS158 – Spring 2022

1

Admin

Assignment 7

2

Perceptron learning algorithm

repeat until convergence (or for some # of iterations):
  for each training example (f_1, f_2, …, f_n, label):

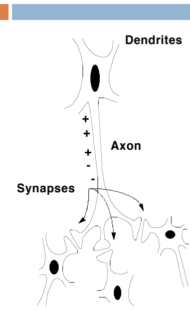
$$prediction = b + \sum_{i=1}^{n} w_i f_i$$

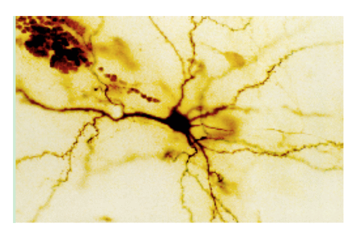
    if *prediction* * *label* ≤ 0: // they don't agree
      for each w_i:
        $w_i = w_i + f_i$*label
      $b = b$ + label

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

3

Our Nervous System



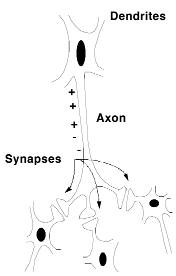


Neuron

What do you know?

4

Our nervous system: the computer science view



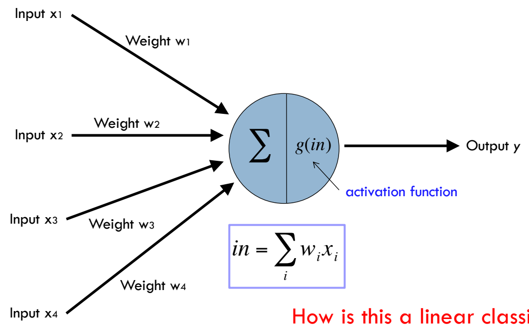
the human brain is a large collection of interconnected neurons

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

A neuron/perceptron



$$in = \sum_i w_i x_i$$

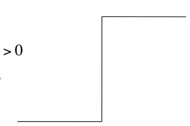
How is this a linear classifier (i.e. perceptron)?

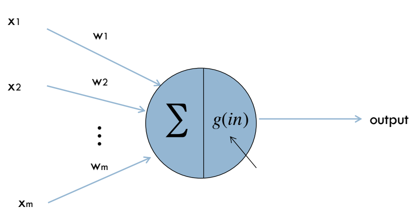
6

Hard threshold = linear classifier

hard threshold:

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





7

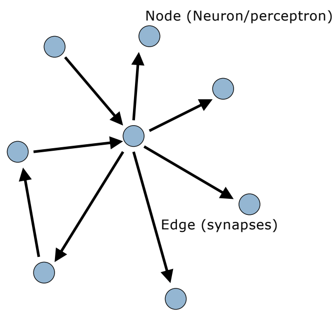
Neural Networks

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

People like biologically motivated approaches

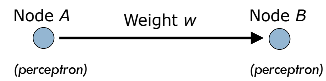
8

Artificial Neural Networks



our approximation

9



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

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

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

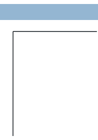
If *A fires* and *w* is **negative**, then *A* **inhibits** *B*.

10

Other activation functions

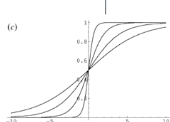
hard threshold:

$$g(in) = \begin{cases} 1 & \text{if } in > -b \\ 0 & otherwise \end{cases}$$

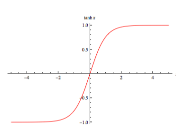


sigmoid

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



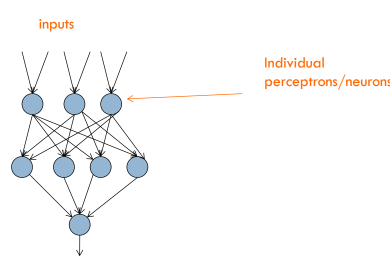
tanh x



why other threshold functions?

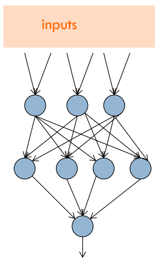
11

Neural network



12

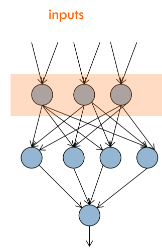
Neural network



some inputs are provided/entered

13

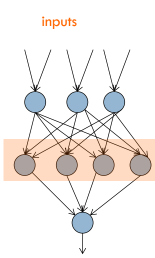
Neural network



each perceptron computes and calculates an answer

14

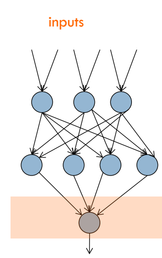
Neural network



those answers become inputs for the next level

15

Neural network



finally get the answer after all levels compute

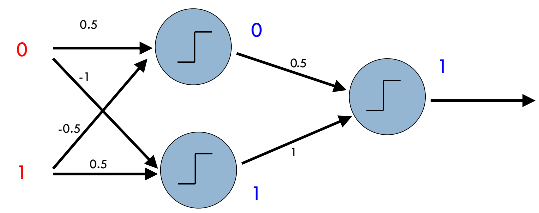
16

Activation spread

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

17

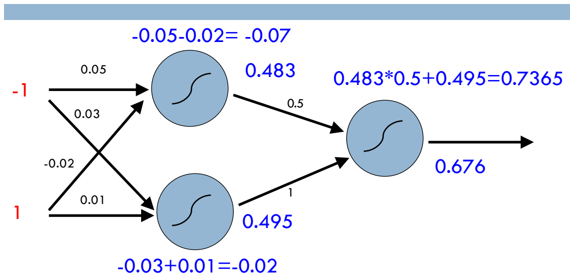
Computation (assume 0 bias)



$$g(in) = \begin{cases} 1 & \text{if } in > -b \\ 0 & otherwise \end{cases}$$

18

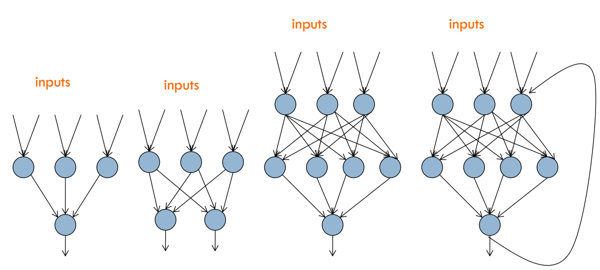
Computation



19

Neural networks

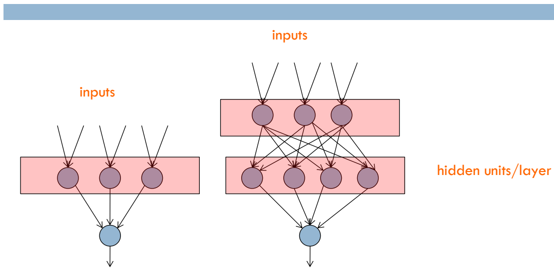
Different kinds/characteristics of networks



How are these different?

20

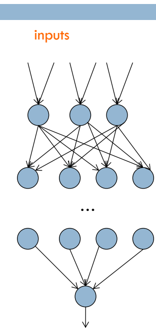
Hidden units/layers



Feed forward networks

21

Hidden units/layers

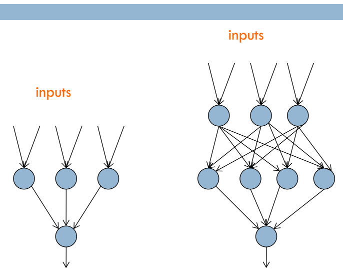


Can have many layers of hidden units of differing sizes

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

22

Hidden units/layers

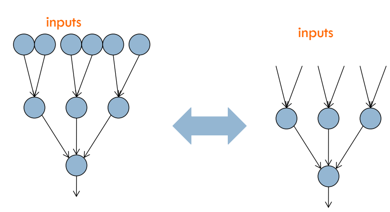


2-layer network 3-layer network

23

Alternate ways of visualizing

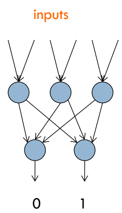
Sometimes the input layer will be drawn with nodes as well



2-layer network 2-layer network

24

Multiple outputs



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

25

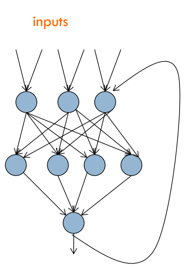
Multiple outputs

input

output
(edge detection)

26

Neural networks



Recurrent network

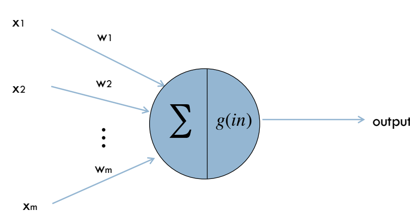
Output is fed back to input

Can support memory!

Good for temporal/sequential data

27

NN decision boundary

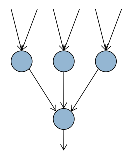


What does the decision boundary of a perceptron look like?

Line (linear set of weights)

28

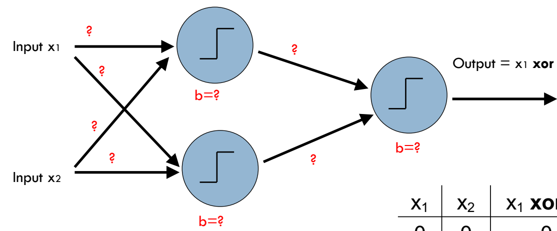
NN decision boundary



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?

29

XOR

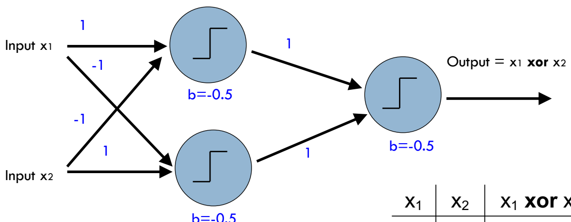


x_1	x_2	x_1 **xor** x_2
0	0	0
0	1	1
1	0	1
1	1	0

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

30

XOR

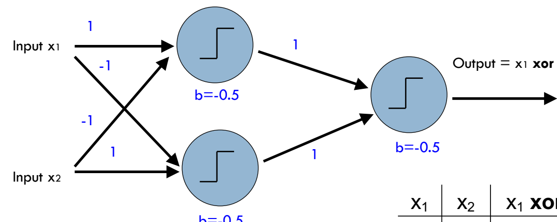


x_1	x_2	x_1 **xor** x_2
0	0	0
0	1	1
1	0	1
1	1	0

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

31

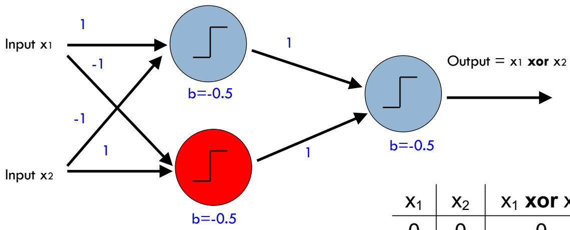
What does the decision boundary look like?



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

32

What does the decision boundary look like?

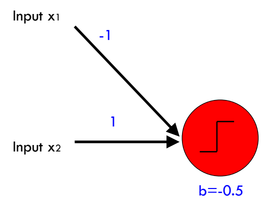


x_1	x_2	x_1 **xor** x_2
0	0	0
0	1	1
1	0	1
1	1	0

What does this perceptron's decision boundary look like?

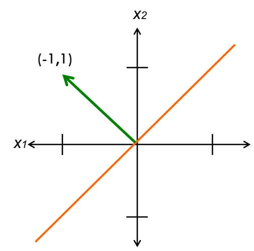
33

NN decision boundary



Let $x_2 = 0$, then:

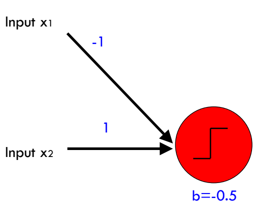
$$-x_1 - 0.5 = 0$$
$$x_1 = -0.5$$

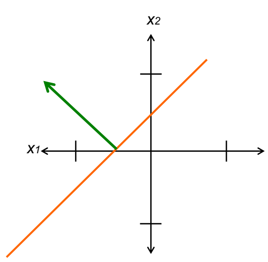


(without the bias)

34

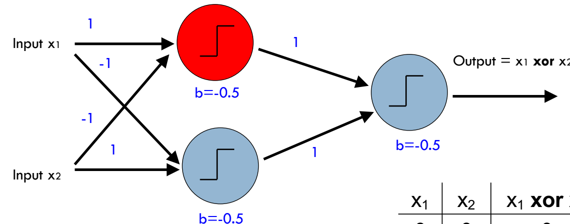
NN decision boundary





35

What does the decision boundary look like?

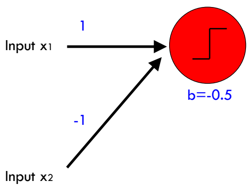


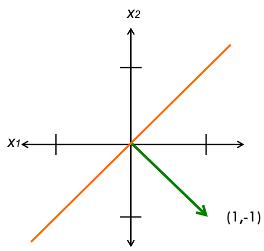
x_1	x_2	x_1 **xor** x_2
0	0	0
0	1	1
1	0	1
1	1	0

What does this perceptron's decision boundary look like?

36

NN decision boundary





Let $x_2 = 0$, then:

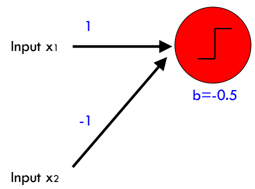
$$x_1 - 0.5 = 0$$

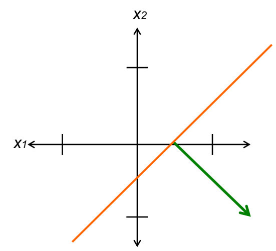
$$x_1 = 0.5$$

(without the bias)

37

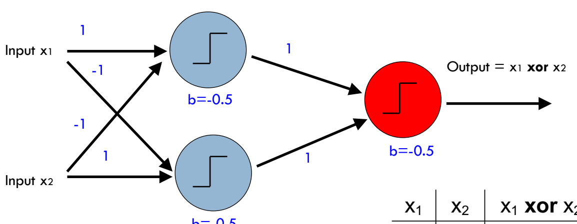
NN decision boundary





38

What does the decision boundary look like?

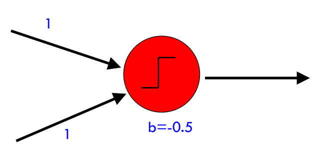


x_1	x_2	x_1 **xor** x_2
0	0	0
0	1	1
1	0	1
1	1	0

What operation does this perceptron perform on the result?

39

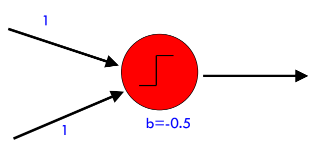
Fill in the truth table



out1	out2	
0	0	?
0	1	?
1	0	?
1	1	?

40

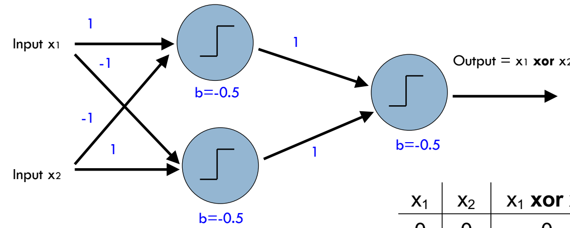
OR



out1	out2	
0	0	0
0	1	1
1	0	1
1	1	1

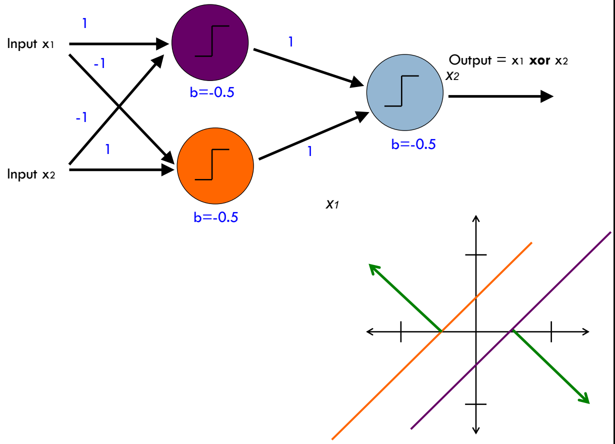
41

What does the decision boundary look like?

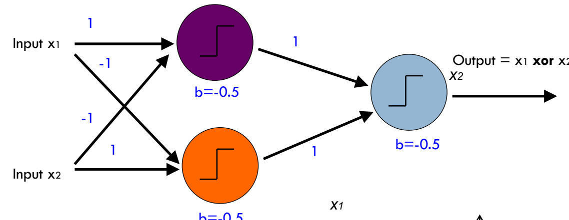


x_1	x_2	x_1 **xor** x_2
0	0	0
0	1	1
1	0	1
1	1	0

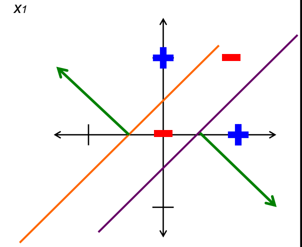
42



43

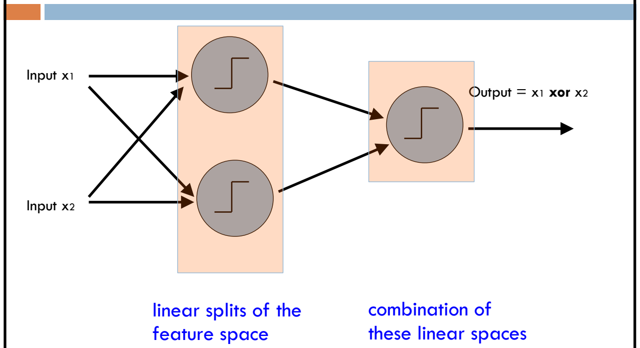


x_1	x_2	x_1 **xor** x_2
0	0	0
0	1	1
1	0	1
1	1	0



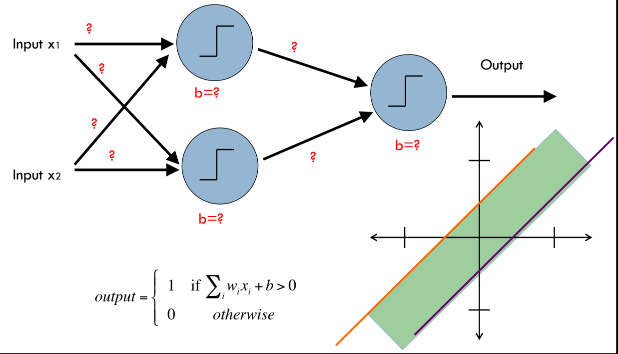
44

What does the decision boundary look like?



45

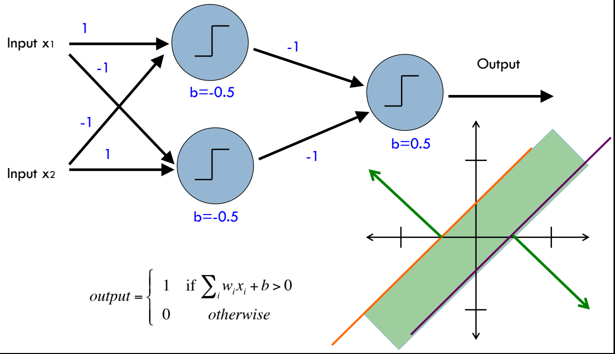
This decision boundary?



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

46

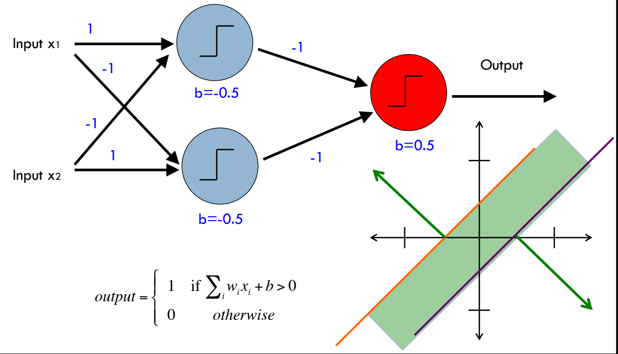
This decision boundary?



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

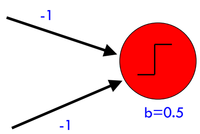
47

This decision boundary?



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

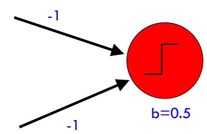
48



out1	out2	
0	0	?
0	1	?
1	0	?
1	1	?

49

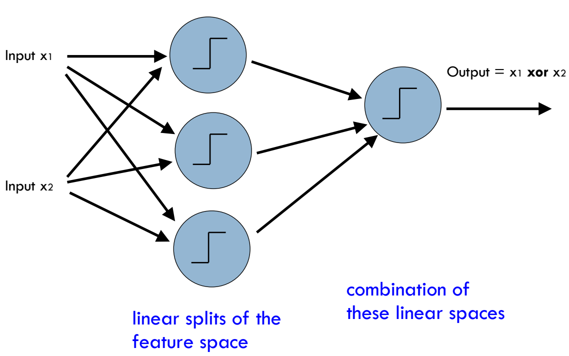
NOR



out1	out2	
0	0	1
0	1	0
1	0	0
1	1	0

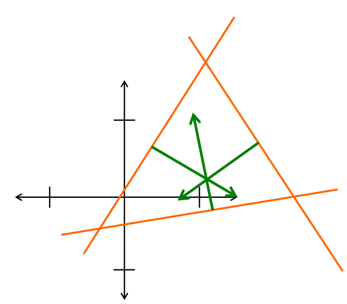
50

What does the decision boundary look like?



51

Three hidden nodes



52

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 $\boldsymbol{x}$ in the domain of F, $|F(\boldsymbol{x}) - \hat{F}(\boldsymbol{x})| < \epsilon$.*

Put simply: two-layer networks can approximate any function

53

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

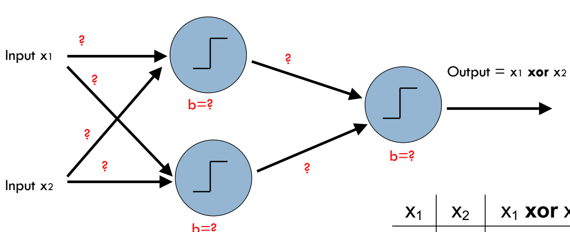
Adding more layers adds even more complexity (and much more quickly)

Good rule of thumb:

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

54

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?

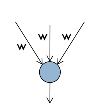
55

Training multilayer networks

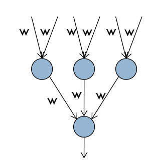
perceptron learning: if the perceptron's output is different than the expected output, update the weights

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

Any other problem with these for general NNs?



perceptron/
linear model



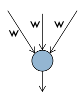
neural network

56

Learning in multilayer networks

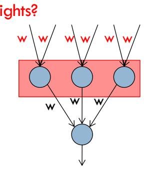
Challenge: for multilayer networks, we don't know what the expected output/error is for the internal nodes!

how do we learn these weights?



perceptron/
linear model

expected output?



neural network

57

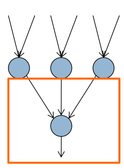
Backpropagation: intuition

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

58

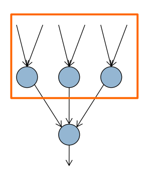
Backpropagation: intuition



We can calculate the actual error here

59

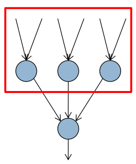
Backpropagation: intuition



Key idea: propagate the error back to this layer

60

Backpropagation: intuition



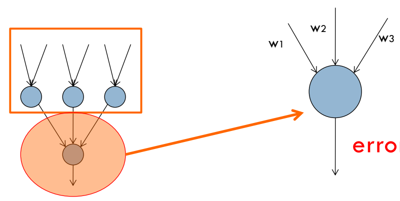
"backpropagate" the error:

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

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

61

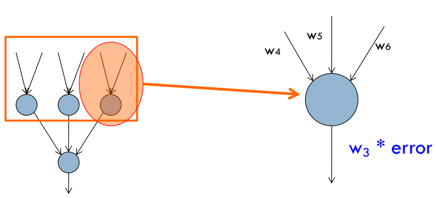
Backpropagation: intuition



error for node is ~ **w_i * error**

62

Backpropagation: intuition



Calculate as normal using this as the error

63

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?

64

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

$$loss = \sum_{x} \frac{1}{2}(y - \hat{y})^2$$ squared error

65