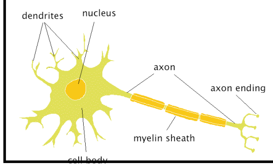
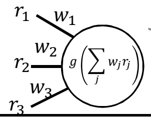
CISC 3250
Systems Neuroscience

Neural networks and information representation in computer science

March 3 edition

Professor Daniel Leeds
dleeds@fordham.edu
JMH 328A

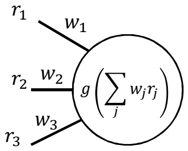


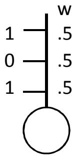


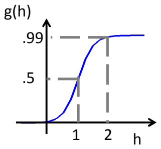
Artificial neuron – the perceptron

Perceptron – building block of artificial neural networks

- Weight inputs
- Perform summation
- Pass through non-linearity







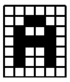
h=1x.5+0x.5+1x.5 = .5+.5 = 1

r_{out}=g(h)=g(1)=.5

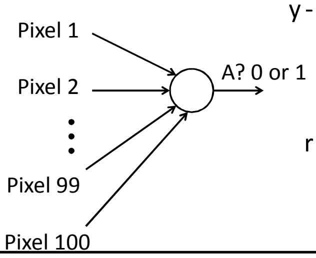
Example: Optical character recognition

Task is to identify a letter from a picture of that letter

x - input



Each pixel is 0 (white) or 1 (black)

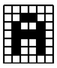


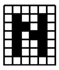
y - desired output

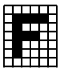
r - actual output

Feature vector input

x_k = [pixel 1, pixel 2, …, pixel 100]

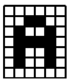
x_1 – pixels in 

x_{10} – pixels in 

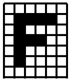
x_{15} – pixels in 

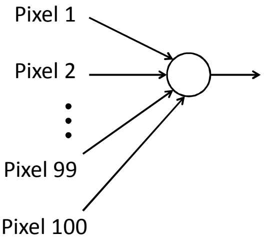
Learning to respond correctly

x_1



x_{15}





y_1=1

r_1=.3 Before learning

r_1=1 After learning

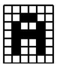
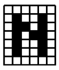
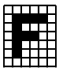
y_{15}=0

r_{15}=.7 Before learning

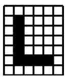
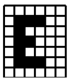
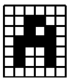
r_{15}=0 After learning

Training the perceptron

- Learn weights that will produced desired perceptron output for set of pictures of letters – **training set**

 y=1  y=0  y=0

- Evaluate these weights by testing correctness of perceptron output for separate set of pictures of letters – **testing set**

 y=0  y=0  y=1

Two learning approaches

Hebbian neurons: “cells that fire together, wire together”

Delta learning: Correcting weights to minimize error between perceptron output and expected output

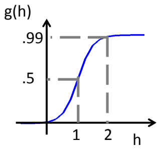
$$E = \frac{1}{2}\sum_i \left(r_i^{out} - y_i\right)^2$$

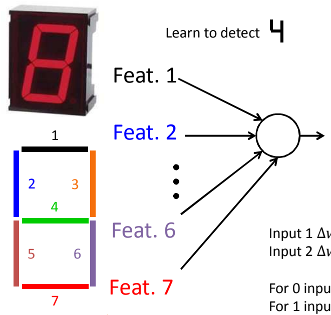
Perceptron math

Delta learning: Correcting weights to minimize error between perceptron output and expected output; *using weighted sum and sigmoid non-linearity g^{sig}*

$$\Delta w_{ij} = \overset{\text{Learning rate}}{\epsilon}\overbrace{\left(1 - r_i^{out}\right)}^{\text{Weight decay}}\underbrace{\left(\underset{\text{Desired output}}{y_i} - \underset{\text{Actual output}}{r_i^{out}}\right)}_{\text{Error correction}}\underset{\text{Hebb}}{\boxed{r_i^{out} r_j^{in}}}$$

Perceptron math example





w = [.2 .2 .2 .2 .2 .2 .2]
Input 4 , expect output y=1

h=0x.2+1x.2+1x.2+1x.2+0x.2+1x.2+0x.2=.8
r=g(.8)=.45

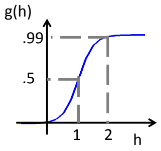
Input 1 Δw_{i1} = (1-.45) (1-.45) .45 0 = 0
Input 2 Δw_{i1} = (1-.45) (1-.45) .45 1 = .55 .55 .45 = .14

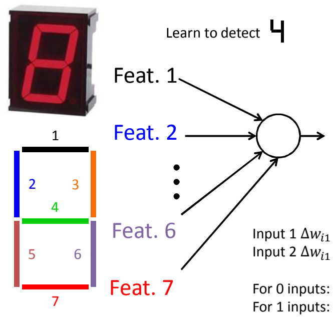
For 0 inputs: +0 -> .2+0 = .2
For 1 inputs: +.14 -> .2+.14=.34

Assume $\epsilon = 1$

$$\Delta w_{ij} = \epsilon\overbrace{\left(1 - r_i^{out}\right)}\overbrace{\left(y_i - r_i^{out}\right)}\overbrace{r_i^{out} r_j^{in}}$$

Perceptron math example





w^{new} = [.2 .34 .34 .34 .2 .34 .2]
Input 2 , expect output y=0

h=1x.2+0x.34+1x.34+1x.34+1x.2+0x.34+1x.2=1.28
r=g(.8)=.58

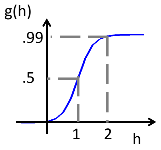
Input 1 Δw_{i1} = (1-.58) (0-.58) .58 1 = .42 -.42 .58 = -.10
Input 2 Δw_{i1} = (1-.58) (0-.58) .58 0 = 0

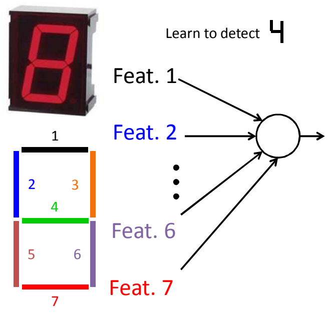
For 0 inputs: +0
For 1 inputs: -.1

Assume $\epsilon = 1$

$$\Delta w_{ij} = \epsilon\overbrace{\left(1 - r_i^{out}\right)}\overbrace{\left(y_i - r_i^{out}\right)}\overbrace{r_i^{out} r_j^{in}}$$

Perceptron math example



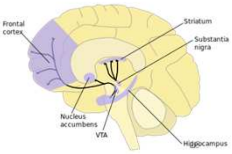


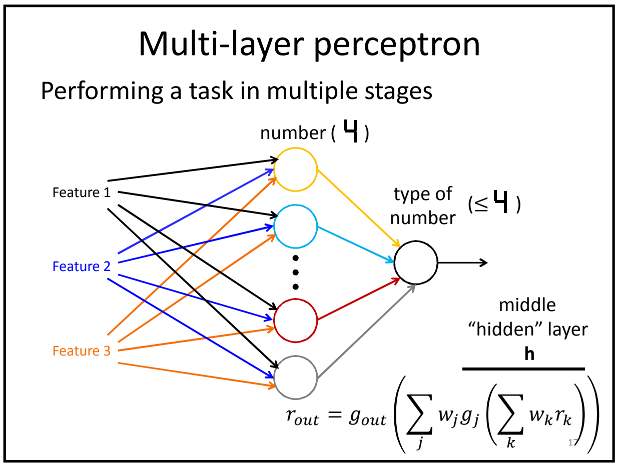
Eventually, we will get weights like:
w = [-2 .5 .5 .5 -2 .5 -2]

Requiring all edges of 4 to be present and no spurious edges

Hebb vs. delta learning

- Hebb is **unsupervised**
  - no “right answer” given
  - neuron/animal notices what inputs co-occur
- Delta rule is **supervised**
  - neuron instructed how to behave
  - mouse gets food reward for pushing lever -> mouse presses lever more often
  - in biology: dopamine reward feedback from ventral tegmental area (VTA) – in the limbic system

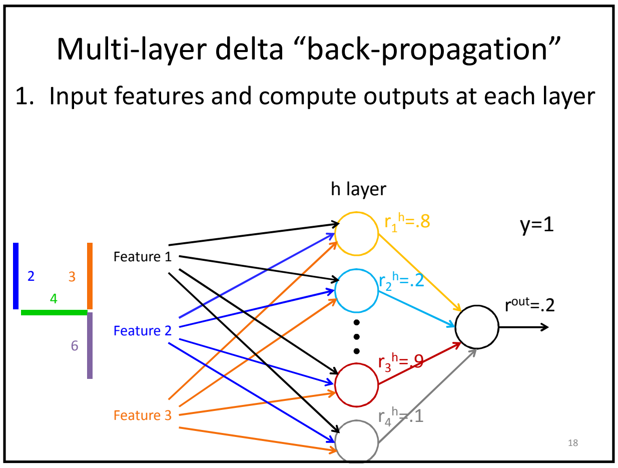




Multi-layer perceptron

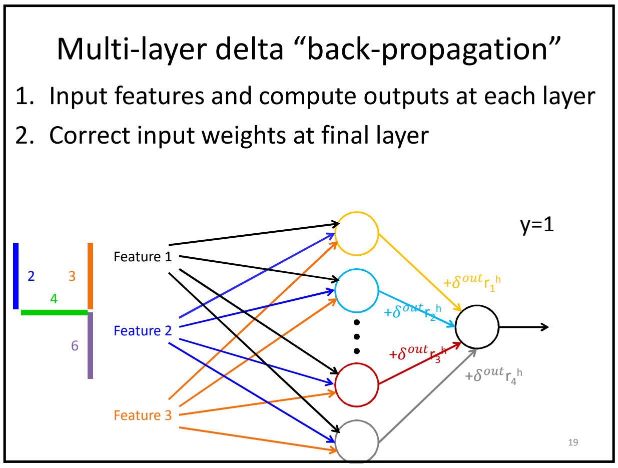
Performing a task in multiple stages

$$r_{out} = g_{out}\left(\sum_j w_j g_j\left(\sum_k w_k r_k\right)\right)$$



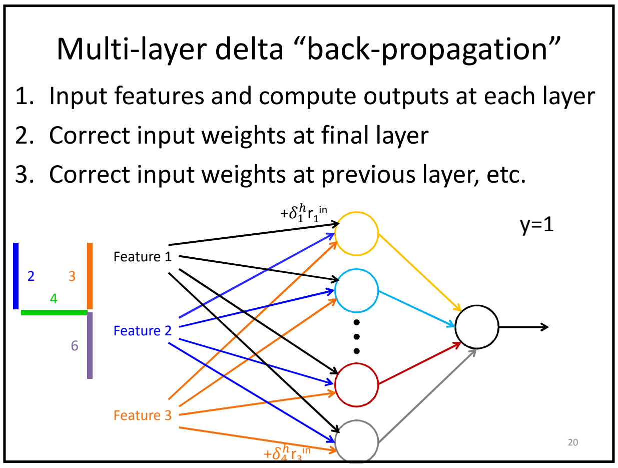
Multi-layer delta “back-propagation”

1. Input features and compute outputs at each layer



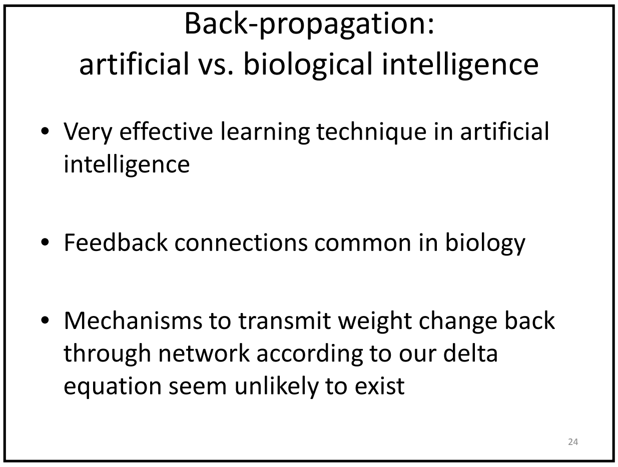
Multi-layer delta “back-propagation”

1. Input features and compute outputs at each layer
2. Correct input weights at final layer



Multi-layer delta “back-propagation”

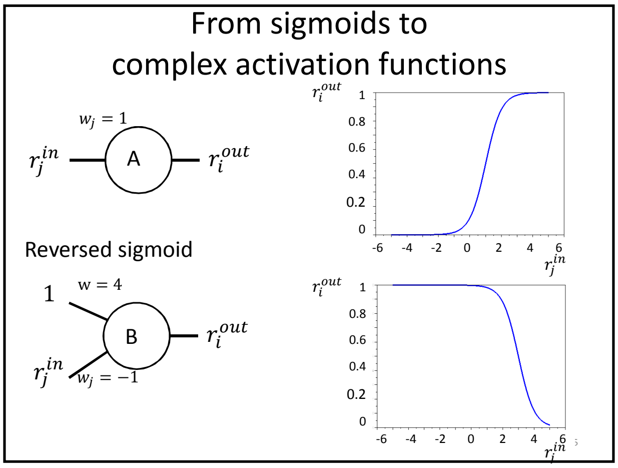
1. Input features and compute outputs at each layer
2. Correct input weights at final layer
3. Correct input weights at previous layer, etc.



Back-propagation: artificial vs. biological intelligence

- Very effective learning technique in artificial intelligence
- Feedback connections common in biology
- Mechanisms to transmit weight change back through network according to our delta equation seem unlikely to exist

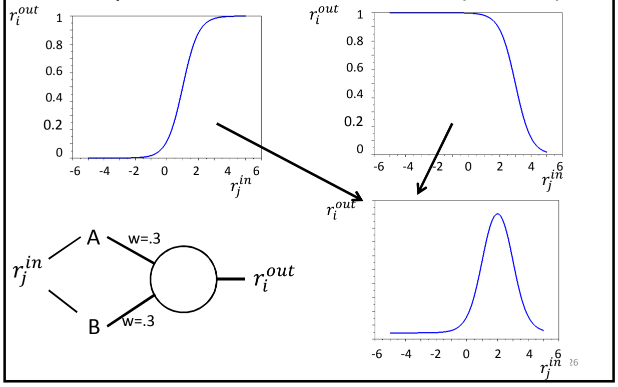
24



From sigmoids to complex activation functions

Reversed sigmoid

From sigmoids to complex activation functions (cont'd)



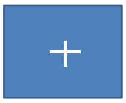
26

Finding structure in data with perceptron learning

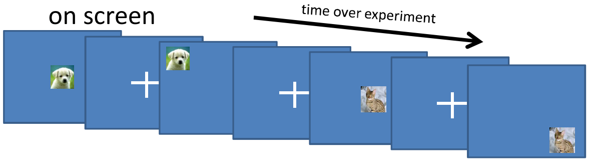
- If we can learn perceptron weights from a training set to predict correct outputs on testing set, there is a simple connection between the input features and the output
- Assign the input features to be variables in an experiment, assign 0-or-1 perceptron output to indicate condition under study

27

Example



- Have subject stare at center of screen
- Flash image of dog or cat very quickly (50 ms) on screen



- Ask subject to press button if they see a dog

Expect subjects will see dog only if dog appears at center of screen where subject is looking.

28

Example

- x_{train} : [1 0 0 0 1 1 …], 1 if dog appears at center for a given display, 0 if dog at side
- y_{train} : [1 0 0 0 1 1 …], 1 if subject sees dog, 0 if subject does not see dog
- Learned perceptron weights will predict subject's future perception of dog based on picture location, because there is a connection between picture location and ability to rapidly perceived an image

29