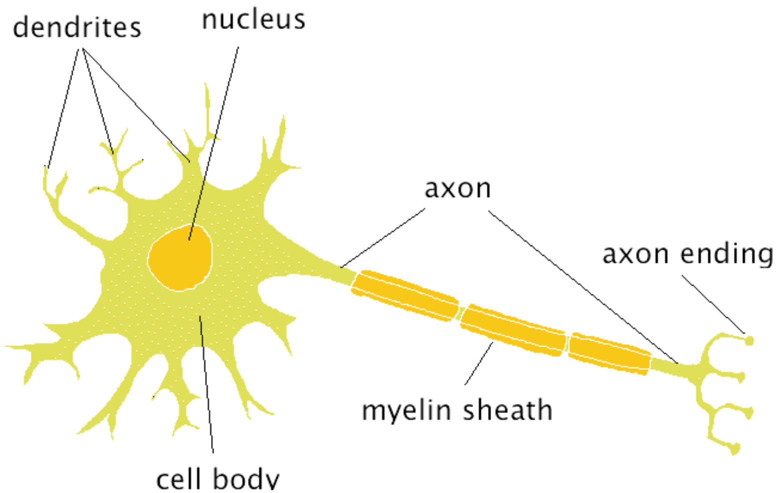
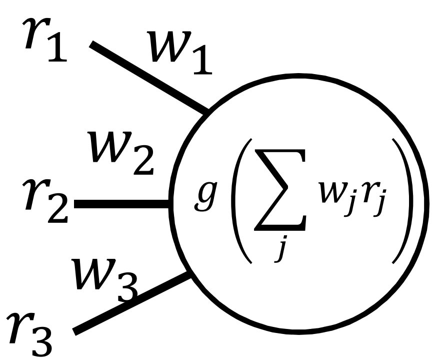
CISC 3250
Systems Neuroscience

Neural networks and information representation in computer science

Professor Daniel Leeds
dleeds@fordham.edu
JMH 328A

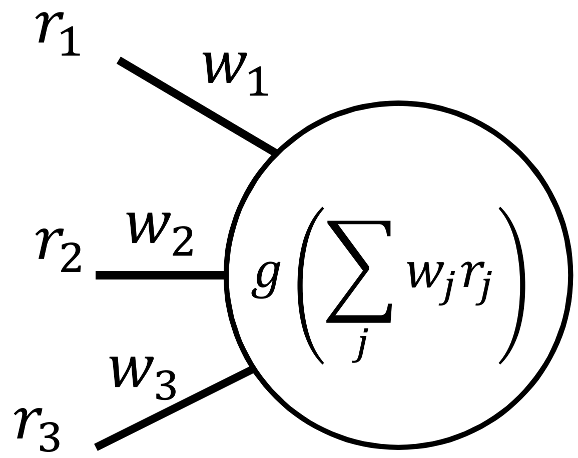




Artificial neuron – the perceptron

Perceptron – building block of artificial neural networks

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

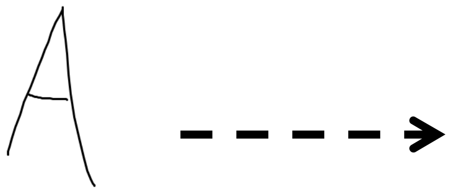


What are our inputs?

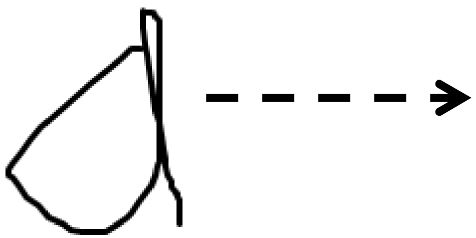
How should we construct a network?

Example: Optical character recognition

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



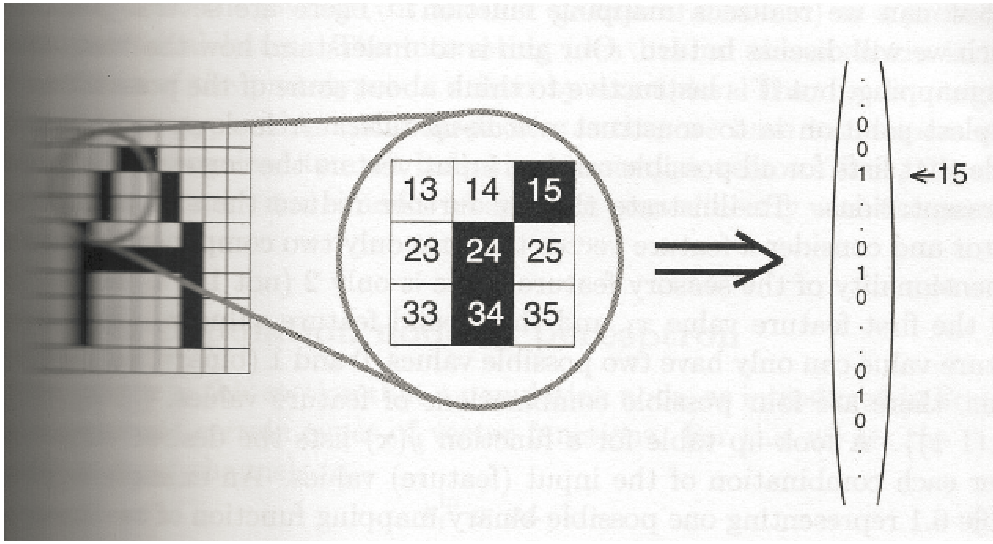
"the letter A"



"the letter A"

Computational representations

- Input: black-and-white pixels – binary vector



A **vector** is a list of numbers, displayed in a column (or a row)

rowVector=[1 0 2 0 .5]

$$\begin{bmatrix} 1 & 0 & 2 & 0 & .5 \end{bmatrix}$$

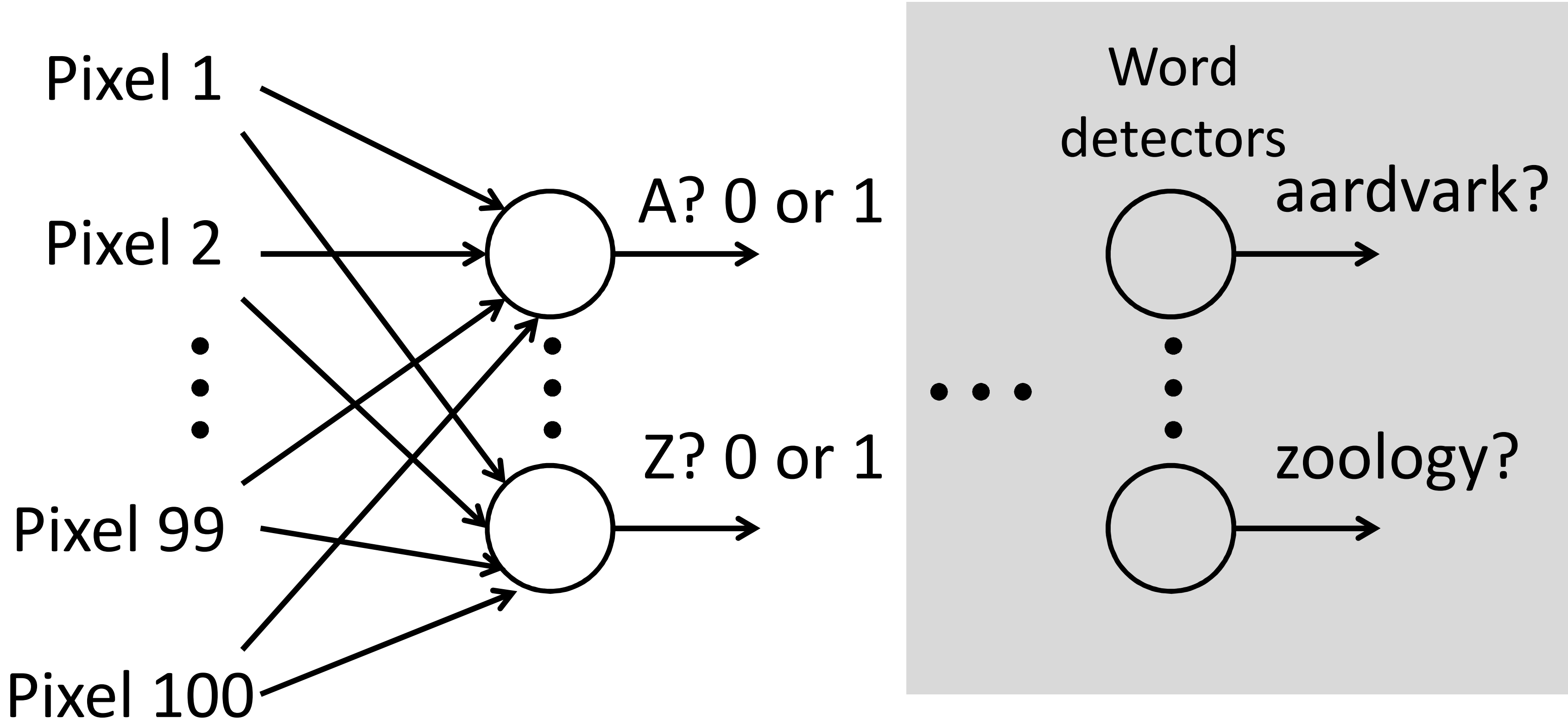
colVector=[1; 0; 2; 0; .5]

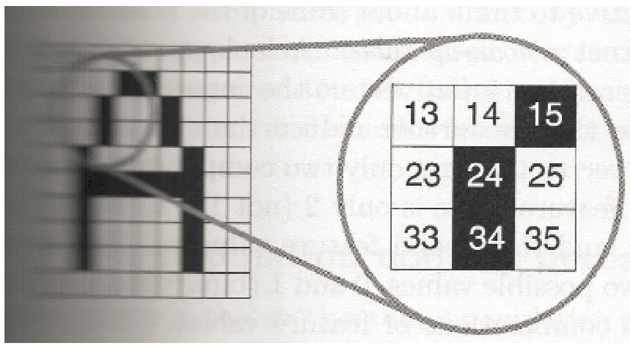
$$\begin{bmatrix} 1 \\ 0 \\ 2 \\ 0 \\ .5 \end{bmatrix}$$

- Output: ASCII (American Standard Code for Information Interchange) – single integer

A	**C**	**E**	**G**	**I**	**K**	**M**	**O**
65	67	69	71	73	75	77	79
B	D	F	H	J	L	N	P
66	68	70	72	74	76	78	80

Pixel input to letter detector output



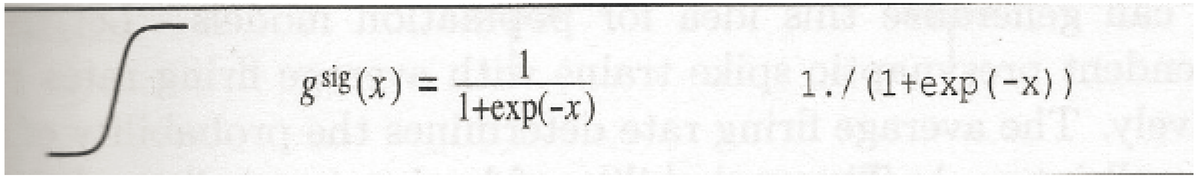


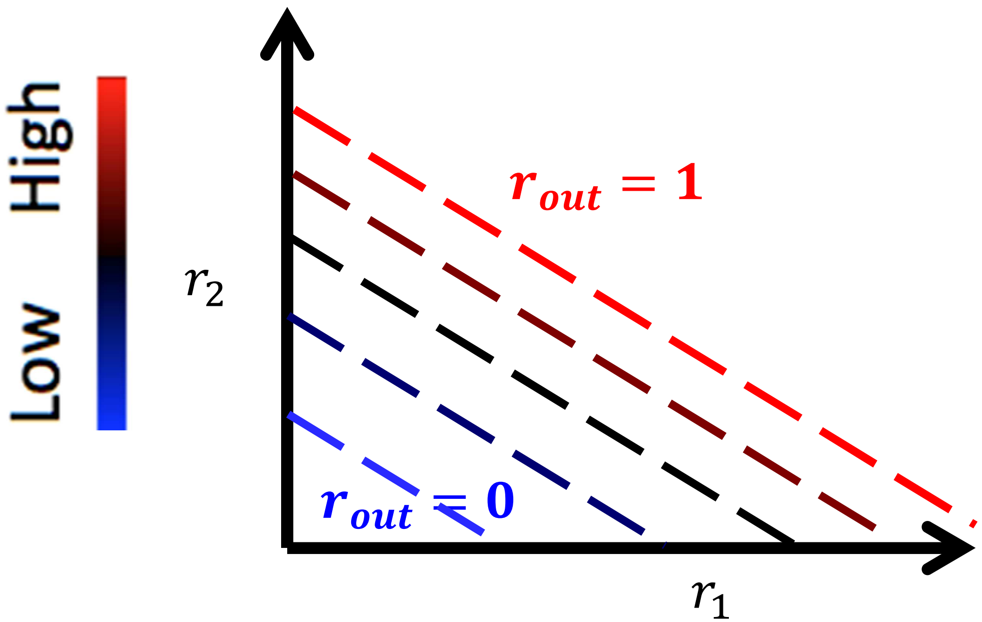
Each set of inputs are **features** describing the world

Decision through threshold

Typical non-linearity

Sigmoid


$$g^{sig}(x) = \frac{1}{1+\exp(-x)}$$
```
1./(1+exp(-x))
```



Learning

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 learning

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

Learning rate

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

Actual output

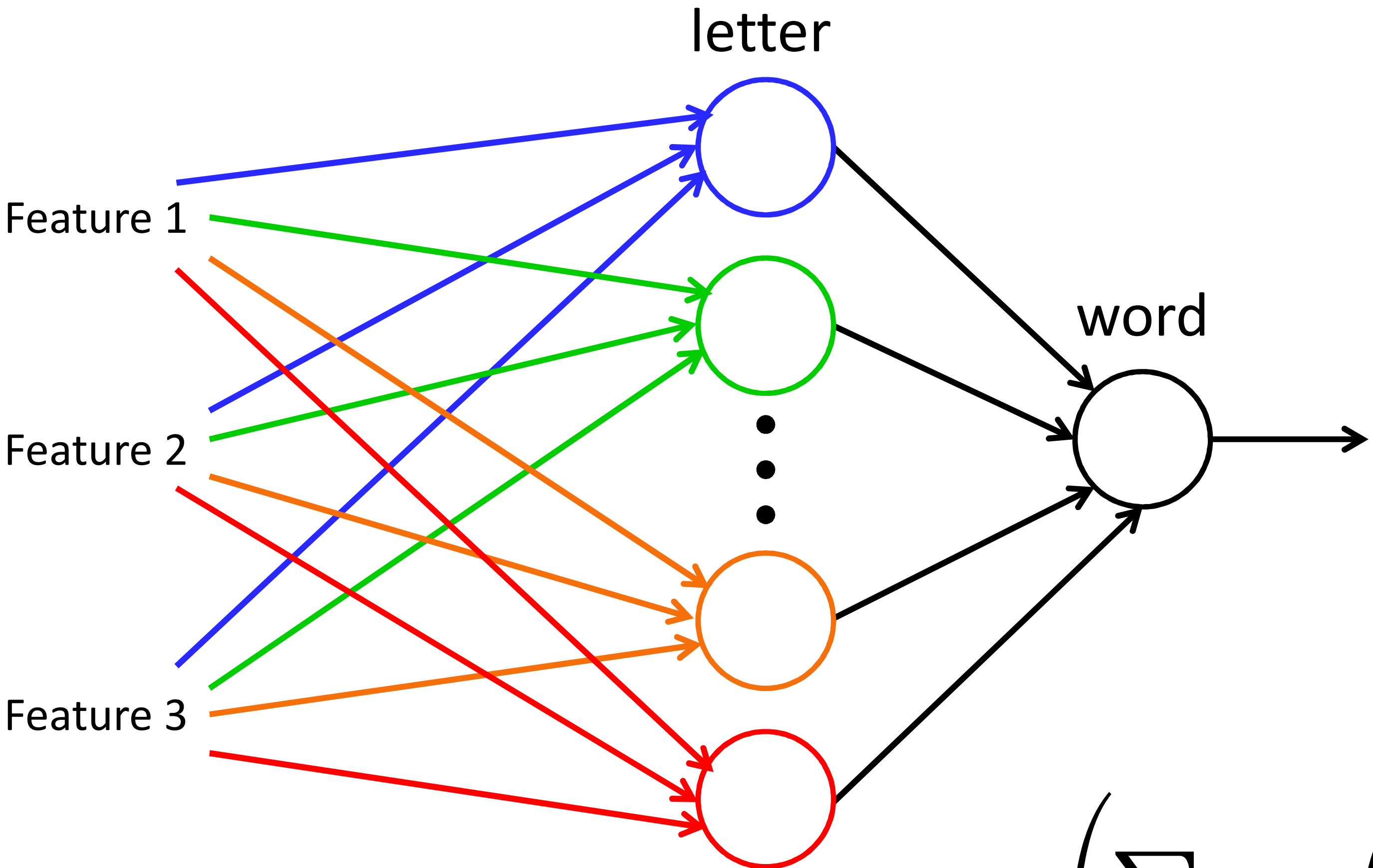
Desired output

Input

What are possible mechanisms for this correction?

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 learning

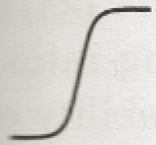
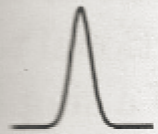
Method:

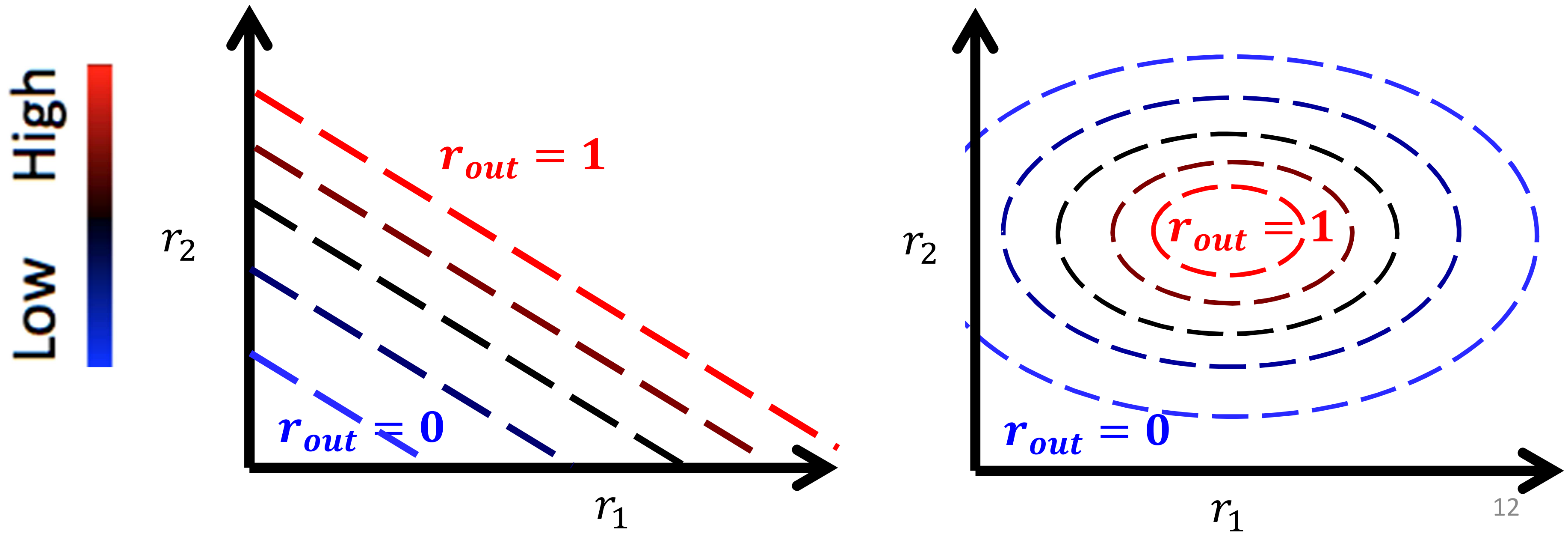
- Input features and compute outputs at each layer
- Correct input weights at final layer
  - $\delta_i^{out} = \epsilon r_i^{out}\left(1 - r_i^{out}\right)\left(r_i^{out} - y_i\right)$
- Correct input weights at previous layer
  - $\delta_i^{l-1} = \epsilon r_i^{l-1}\left(1 - r_i^{l-1}\right)\sum_j w_{ji}^l \delta_j^l$
- …
- Update weights at each layer: $\Delta w_{ij}^l = \epsilon \delta_{ij}^l r_j^{l-1}$

Outdated slides

Decision through threshold

Typical non-linearities

Sigmoid		$g^{\mathrm{sig}}(x) = \frac{1}{1+\exp(-x)}$	`1./(1+exp(-x))`
Radial-basis		$g^{\mathrm{gauss}}(x) = \exp(-x^2)$	`exp(-x.^2)`



Perceptron learning

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

Learning rate

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

Desired output

Actual output

Input

What are possible mechanisms for this correction?

Multi-layer perceptron

- Assembling information across multiple layers
- Equation with back-propagation
- Is it biologically plausible?