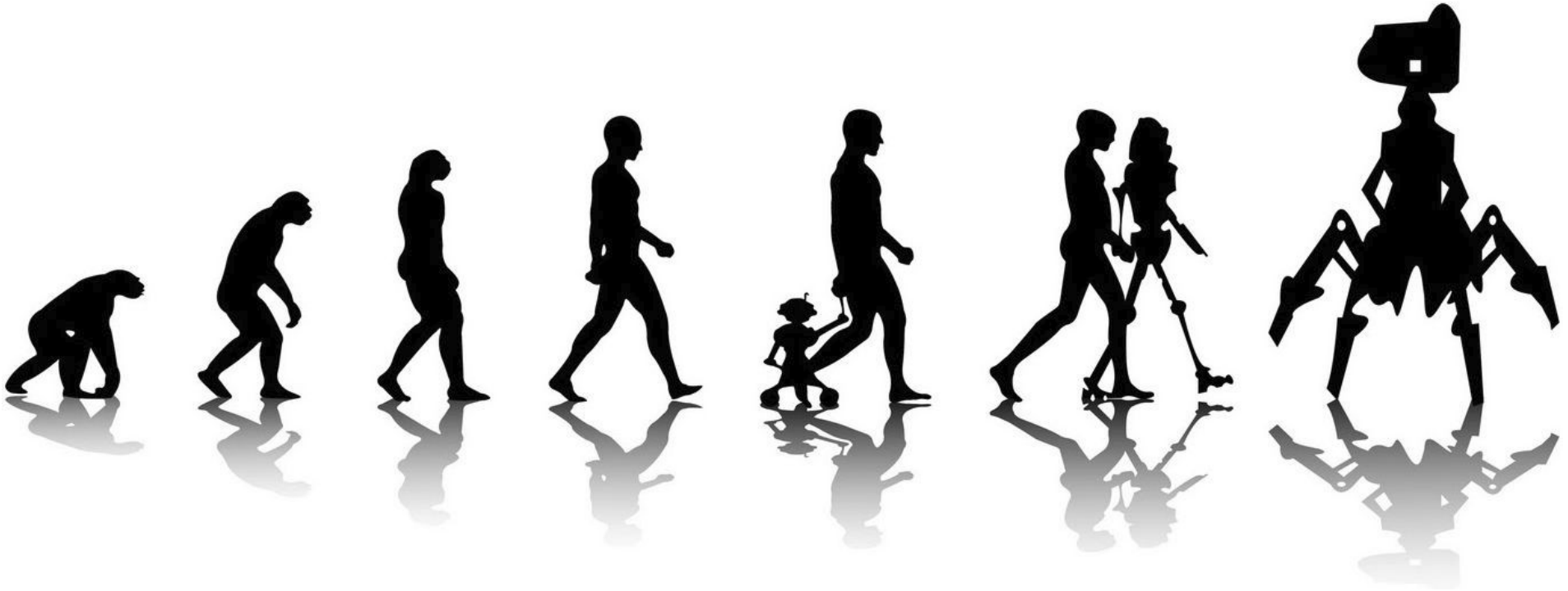


Modern Robotics: Evolutionary Robotics

COSC 4560 / COSC 5560

Professor Cheney

2/2/18

Mutation Rates

Another related parameter to the exploration-exploitation trade-off is the mutation rate

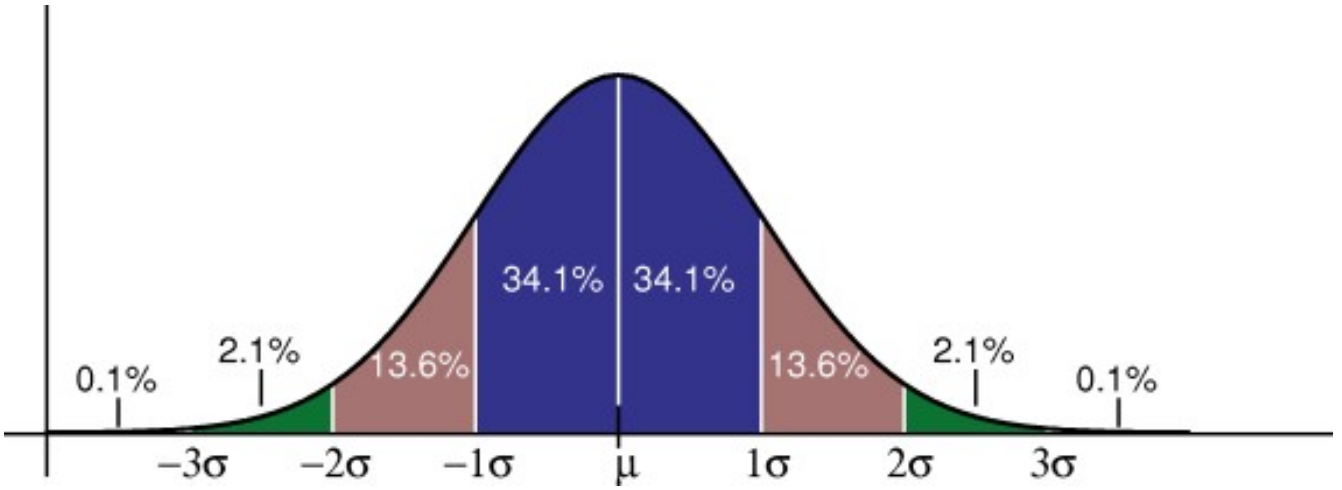
Evolutionary Programming

Typically we want to be taking small steps in the search space

With occasional large exploratory steps

And we are dealing with a vector of real valued parameters

Let's use a Gaussian mutation rate!



$$\mathcal{N}(\mu, \sigma^2)$$

We will use a zero-mean as to not bias mutation

But what should the standard deviation of our Gaussian be?

Typically we'll just choose a small deviation, such that about one entry is changed on average...

But we could optimize that parameter too!

Evolutionary Strategies

We could evolve a single mutation rate meta-parameter that is mutated and selected for just like every other allele in the genome

Now the genome is N+1 entries long (for N evolved features/traits)

Or we could evolve a separate mutation rate for every parameter

Now the genome is 2N entries long (for N evolved features/traits)

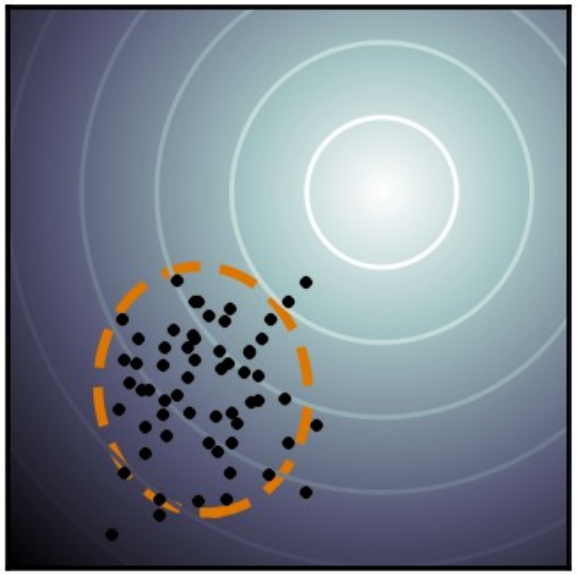
More flexibility/optimization efficiency at the cost of a larger genome

This is very effective in practice

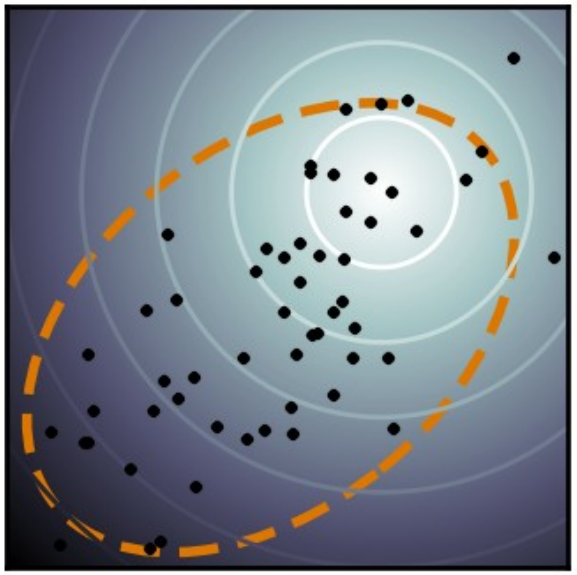
The most popular version is CMA-ES
Covariance Matrix Adaptation -
Evolutionary Strategies

This not only evolves independent
mutation rates, but also accounts for
co-variance between alleles (epigenetics!)

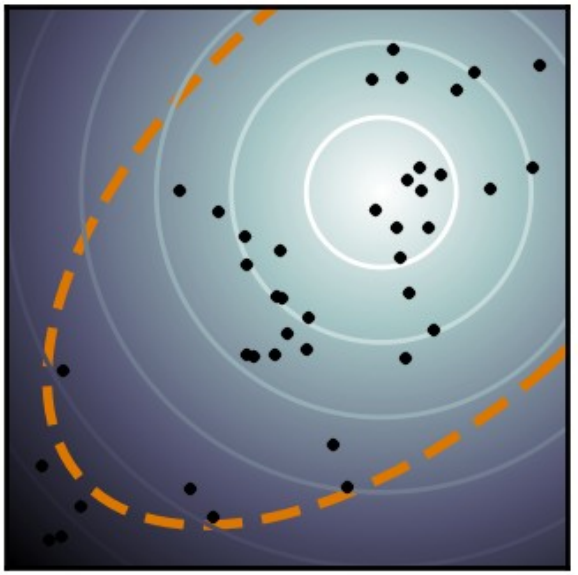
Generation 1



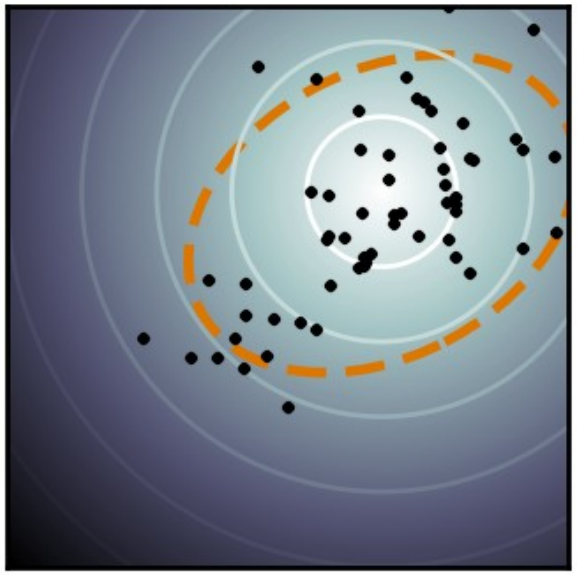
Generation 2



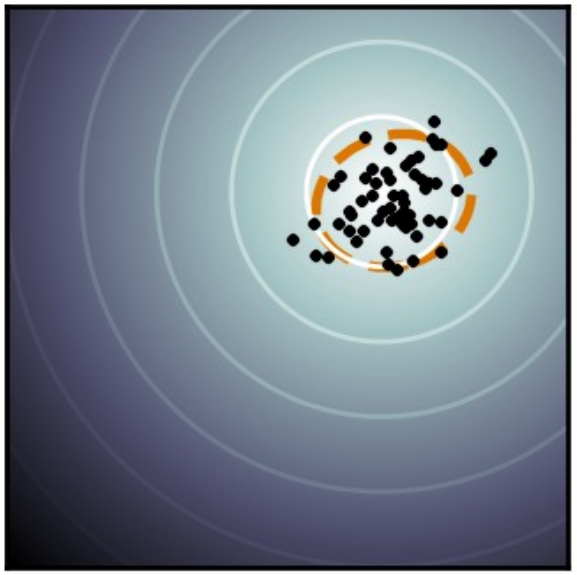
Generation 3



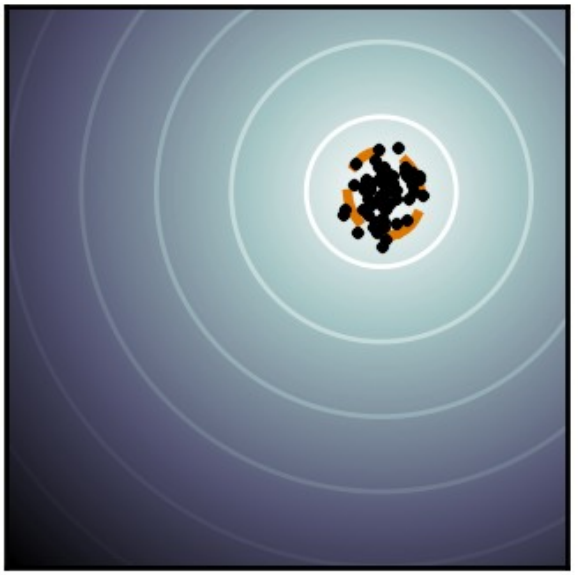
Generation 4

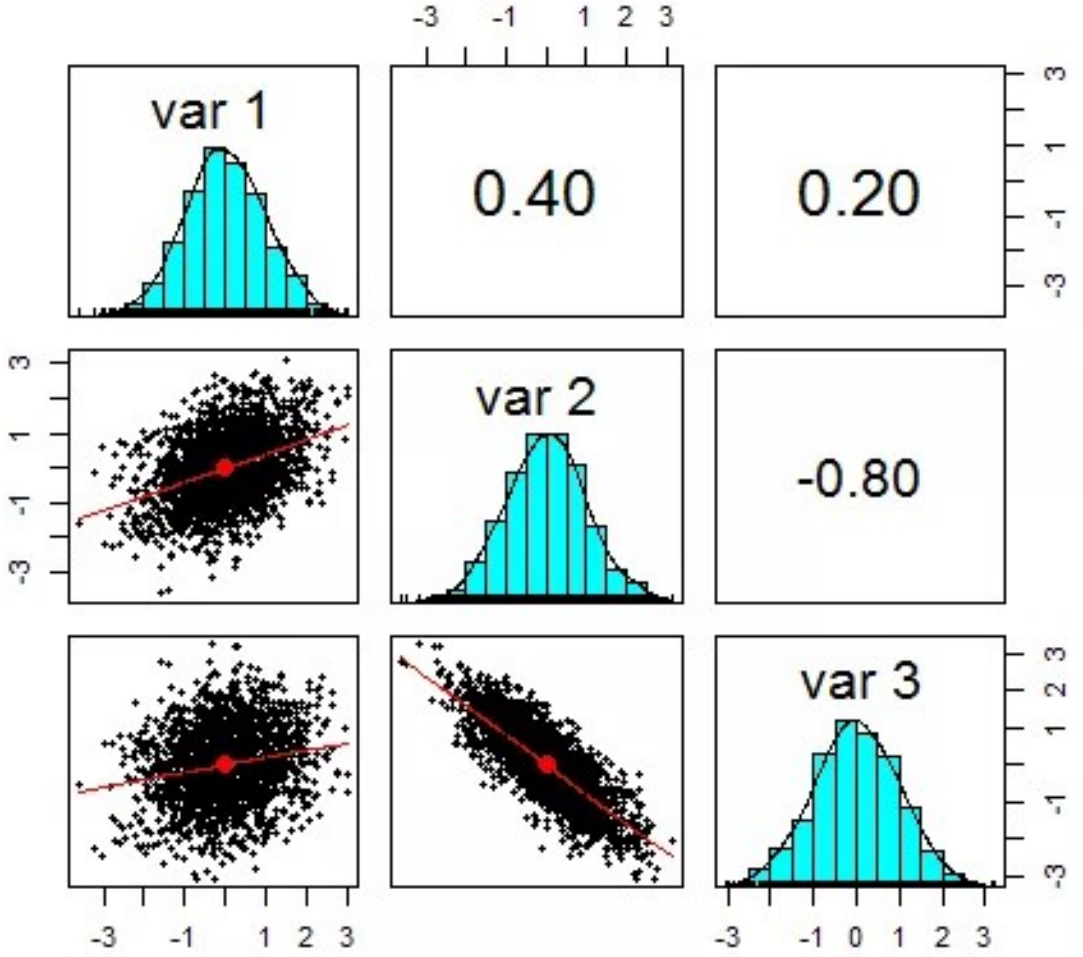


Generation 5



Generation 6



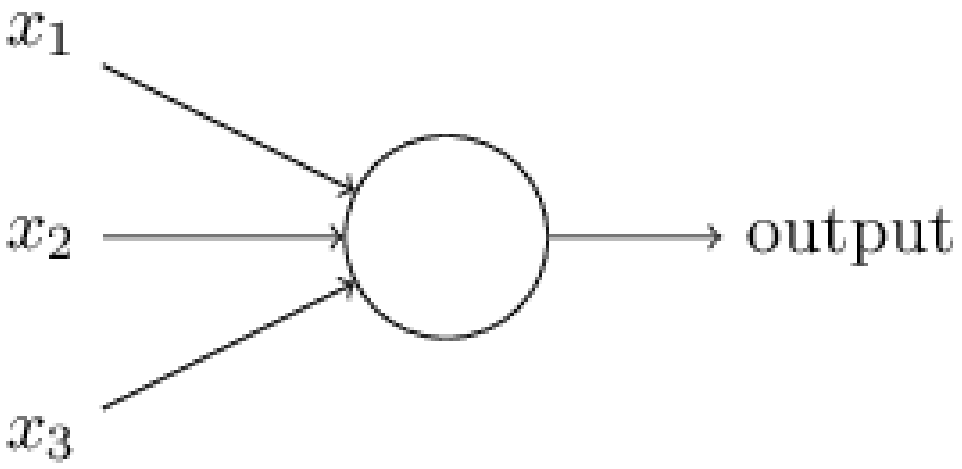

var 1
0.40
0.20
var 2
-0.80
var 3
-3
-1
1
2
3
0

If we have time later in the semester, we can explore CMA-ES, and many other advanced types of evolutionary algorithms in greater detail

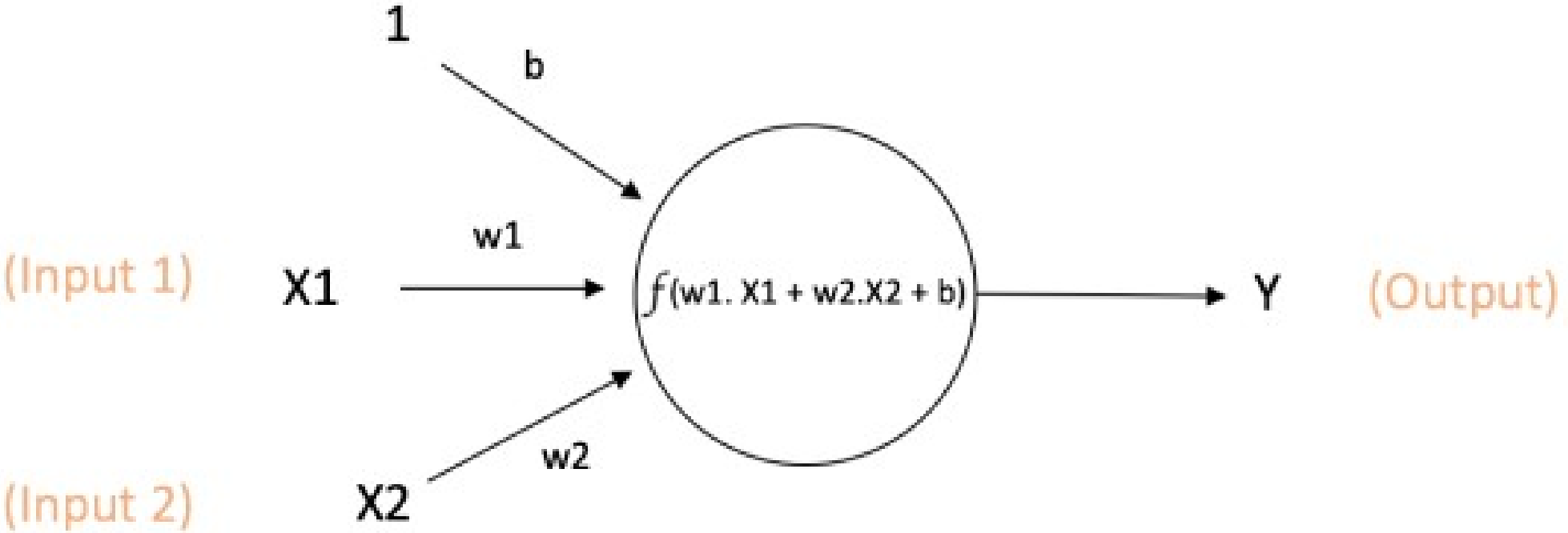
But for now let's skip over them so that we can talk about all the basic tools necessary to start evolving robots!

Neural Networks

artificial neuron



artificial neuron



Output of neuron = Y= f(w1. X1 + w2.X2 + b)

if “activation function” f(x) is:

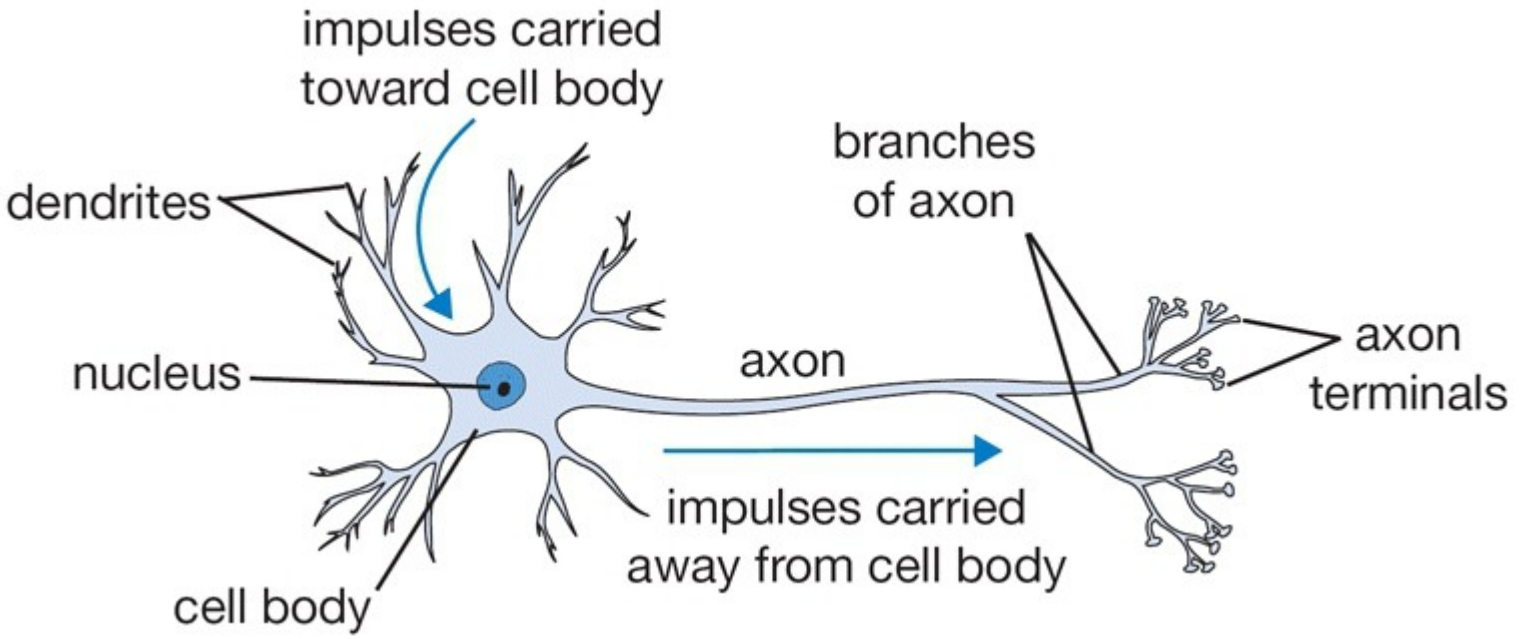
$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

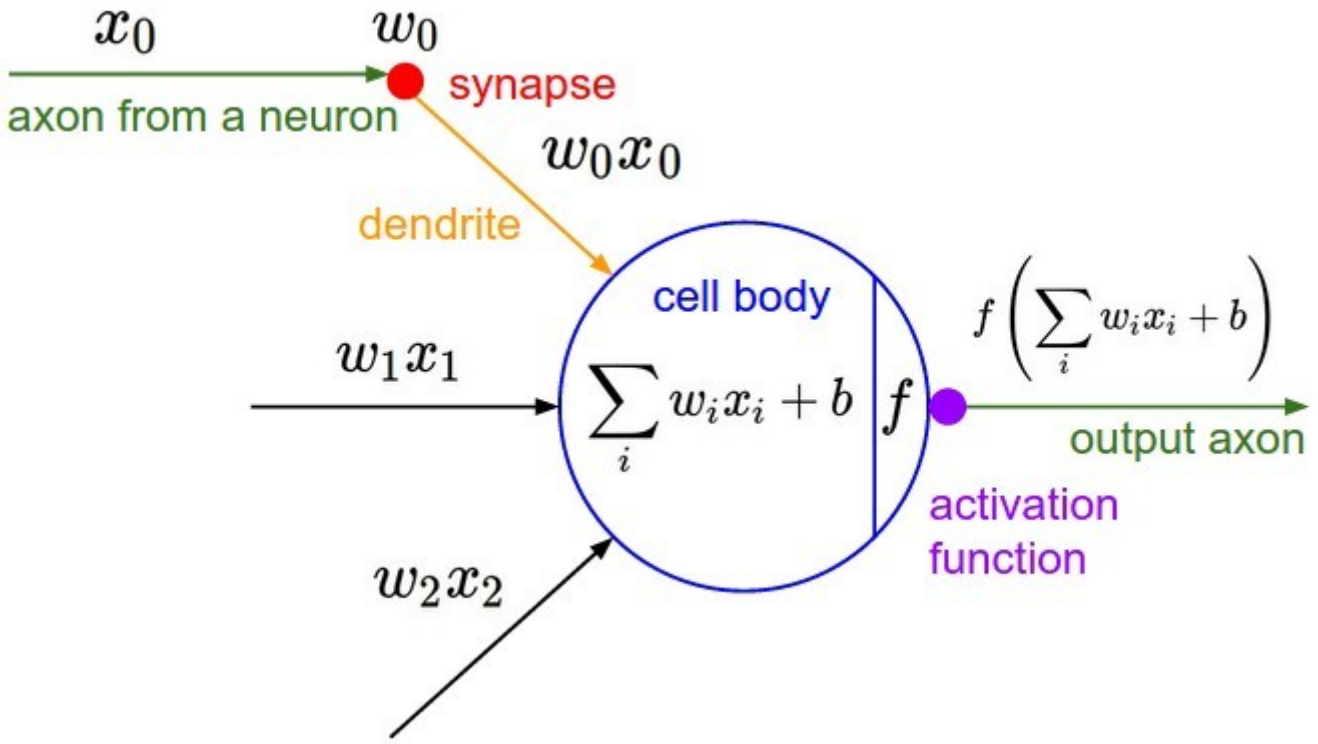
called a “perceptron”

this basic framework is not new…

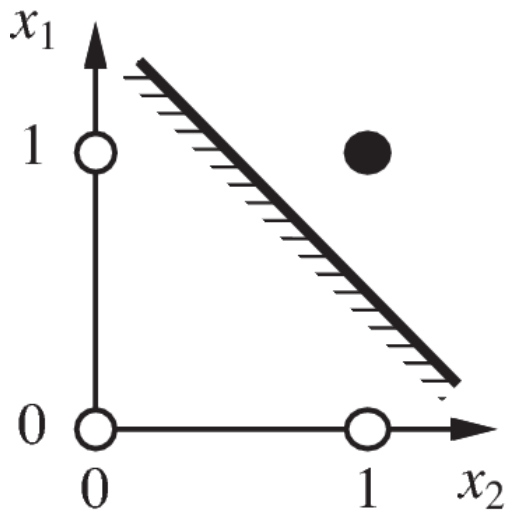
first artificial neuron:
McCulloch & Pitts (1943)

perceptron:
Rosenblatt (1958)

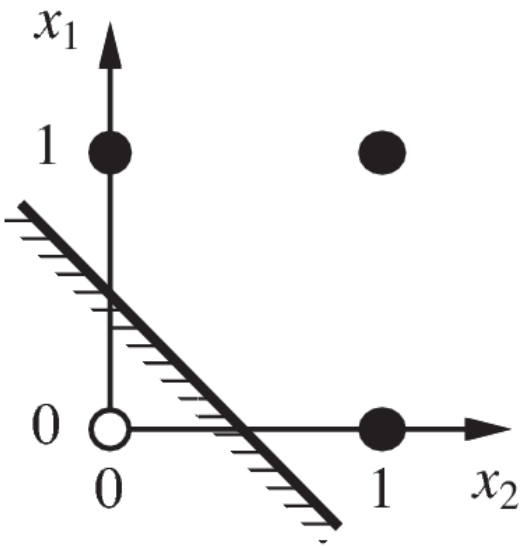

impulses carried
toward cell body
dendrites
branches
of axon
nucleus
axon
axon
terminals
impulses carried
away from cell body
cell body


x_0
w_0
axon from a neuron
synapse
$w_0 x_0$
dendrite
cell body
$w_1 x_1$
$\sum_i w_i x_i + b$
f
$f\left(\sum_i w_i x_i + b\right)$
output axon
activation
function
$w_2 x_2$

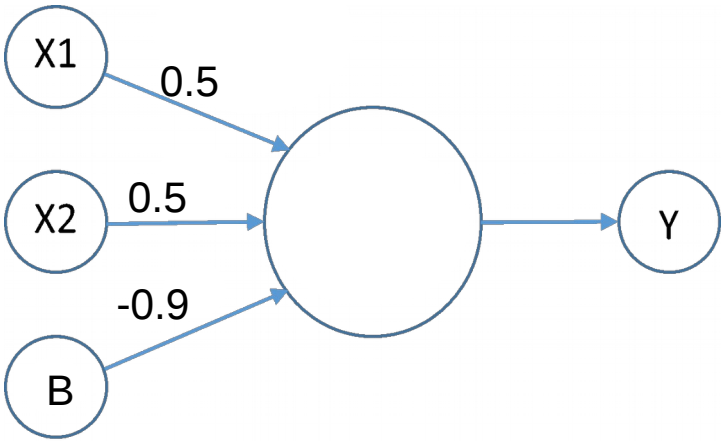
perceptron as logical opperator

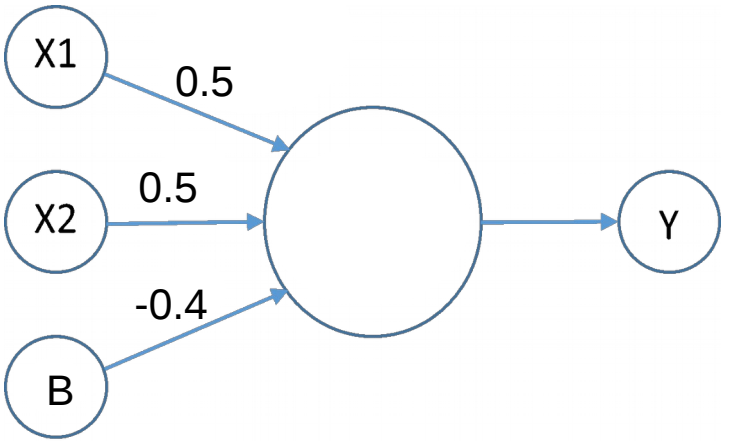


(a) x_1 **and** x_2

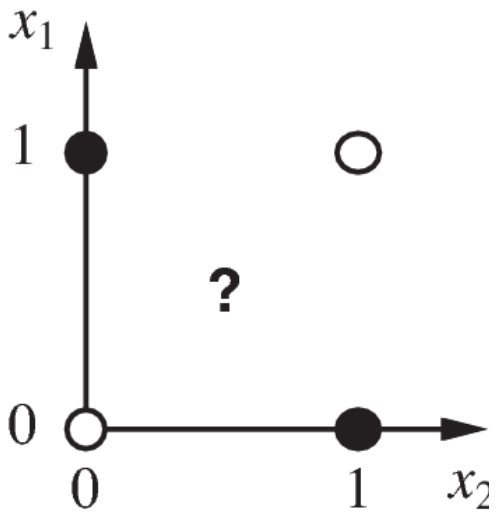


(b) x_1 **or** x_2



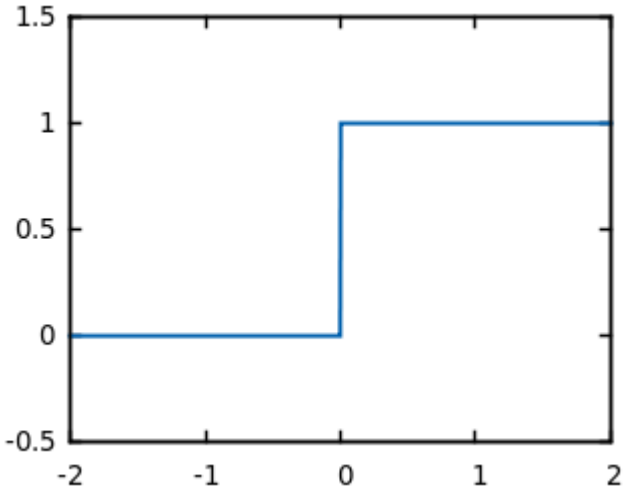


perceptron as logical opperator



(c) x_1 **xor** x_2

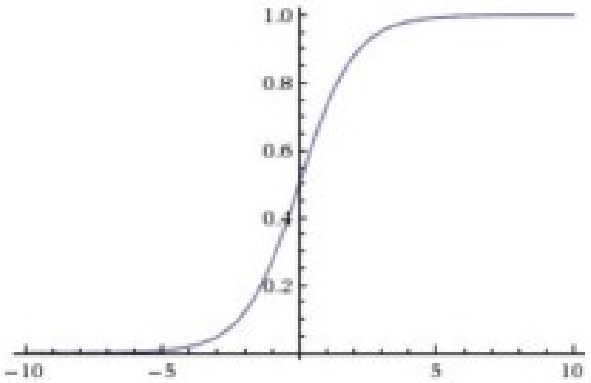
activation functions



Perceptrons use the “step function”

But any non-linear function will work

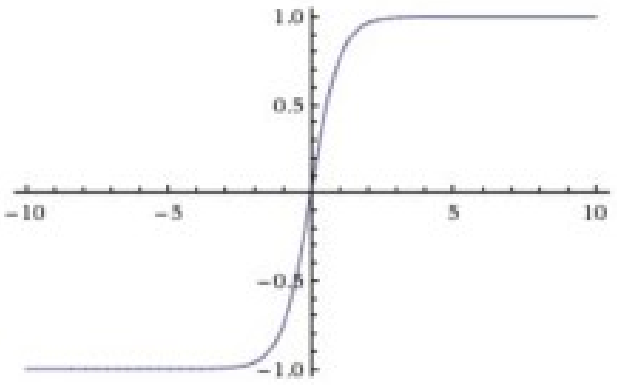
activation functions



Sigmoid

input domain:
[-∞, ∞]

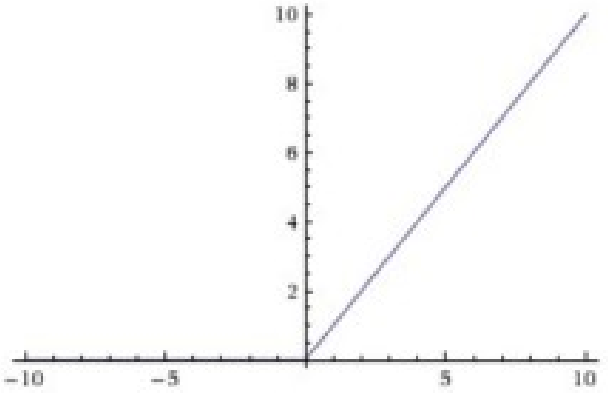
output range:
[0, 1]



tanh

input domain:
[-∞, ∞]

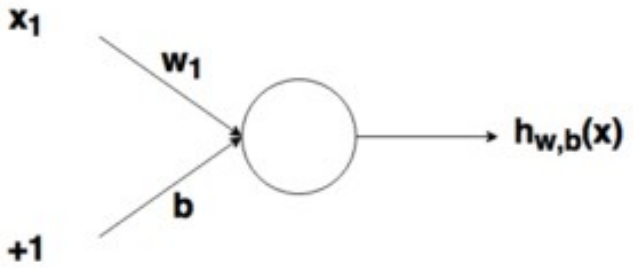
output range:
[-1, 1]

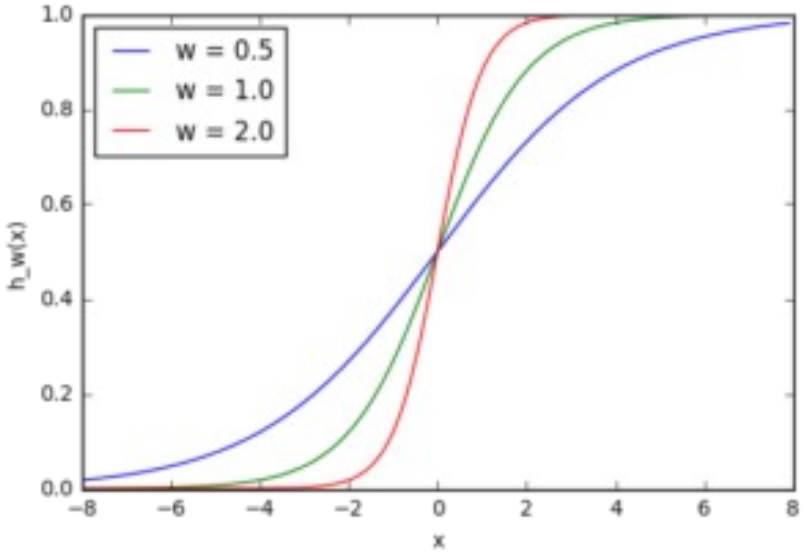


ReLU

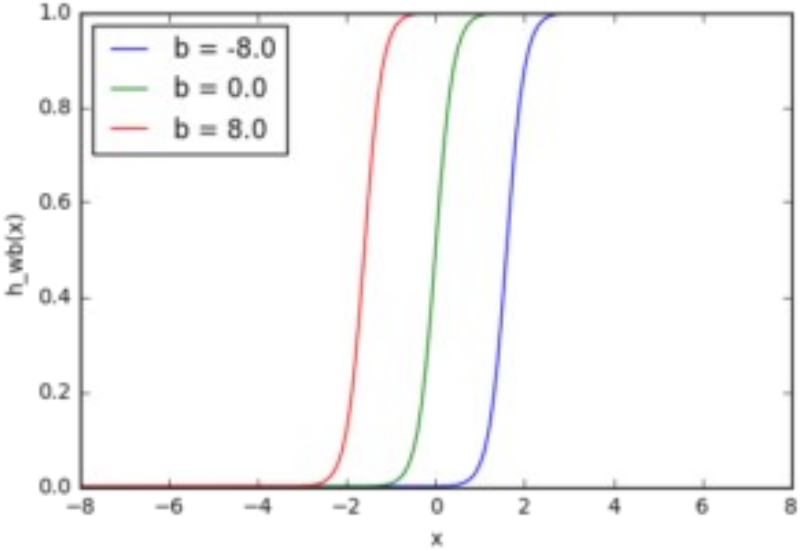
input domain:
[-∞, ∞]

output range:
[0, ∞]



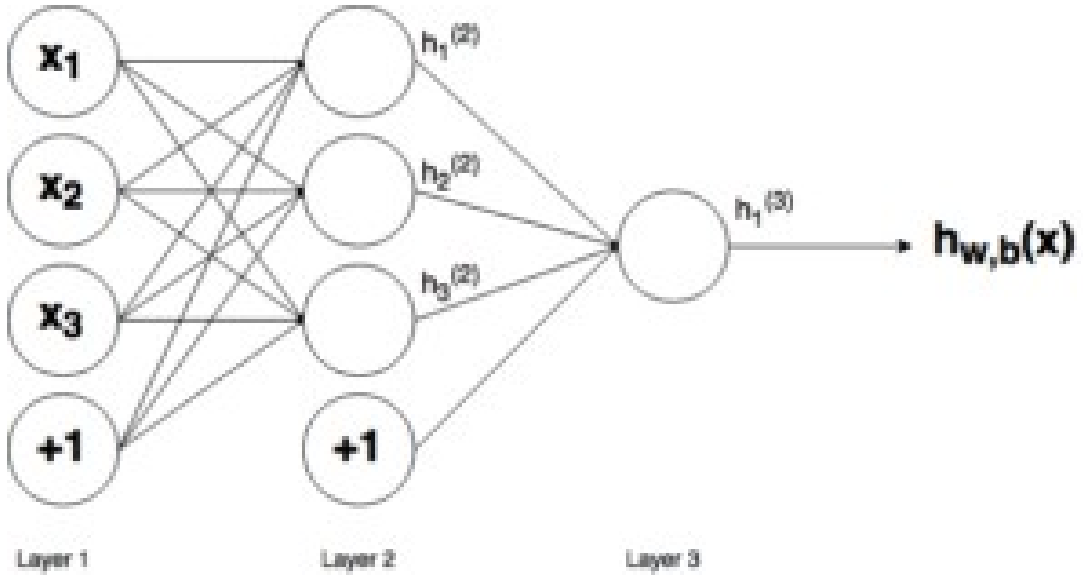


Effect of changing the weight w_1 (if bias=0)



Effect of changing the bias weight (if w_1=5)

multi-layer networks



$$h_1^{(2)} = f(w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + w_{13}^{(1)} x_3 + b_1^{(1)})$$
$$h_2^{(2)} = f(w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{23}^{(1)} x_3 + b_2^{(1)})$$
$$h_3^{(2)} = f(w_{31}^{(1)} x_1 + w_{32}^{(1)} x_2 + w_{33}^{(1)} x_3 + b_3^{(1)})$$

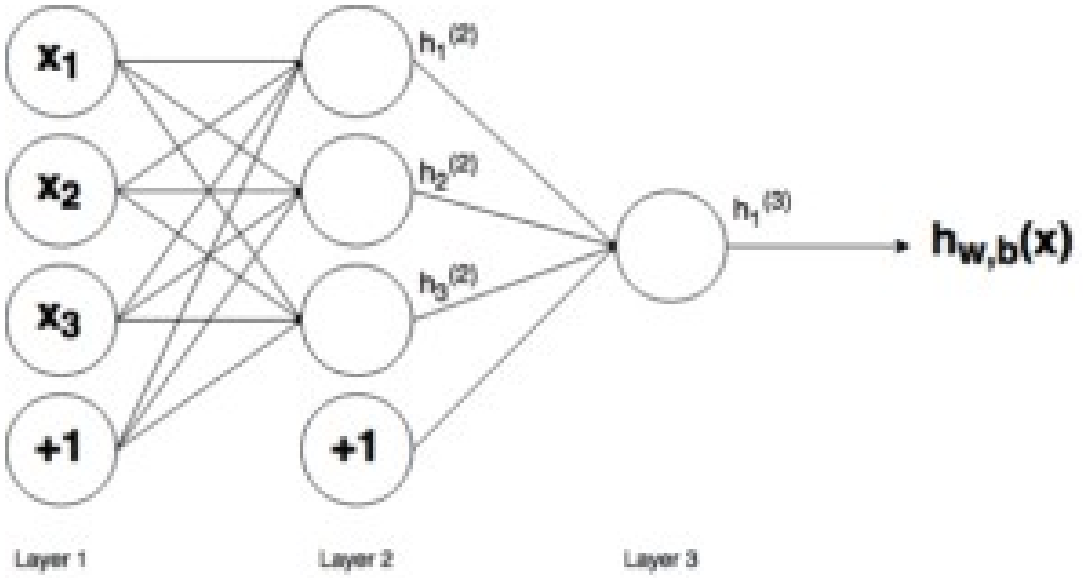
$$
\begin{aligned}
& w_{11}\, x_1 + w_{12}\, x_2 + w_{13}\, x_3 + b_1 \\
& w_{21}\, x_1 + w_{22}\, x_2 + w_{23}\, x_3 + b_2 \\
& w_{31}\, x_1 + w_{32}\, x_2 + w_{33}\, x_3 + b_3
\end{aligned}
$$

$$
= \begin{pmatrix} w_{11}\, x_1 + w_{12}\, x_2 + w_{13}\, x_3 \\ w_{21}\, x_1 + w_{22}\, x_2 + w_{23}\, x_3 \\ w_{31}\, x_1 + w_{32}\, x_2 + w_{33}\, x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}
$$

$$
= \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}
$$

$$
z^{(l+1)} = W^{(l)} h^{(l)} + b^{(l)} = \sum_{j=1}^{n} w_{ij}^{(1)} x_i + b_i^{(1)}
$$

multi-layer networks



All we need to optimize is a weight array for each layer:

$$W^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} \end{pmatrix} \qquad W^{(2)} = \begin{pmatrix} w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)} \end{pmatrix}$$

What about the bias weights?

$$\begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

$$= \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{41} & w_{41} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

if $x_4 = 1$

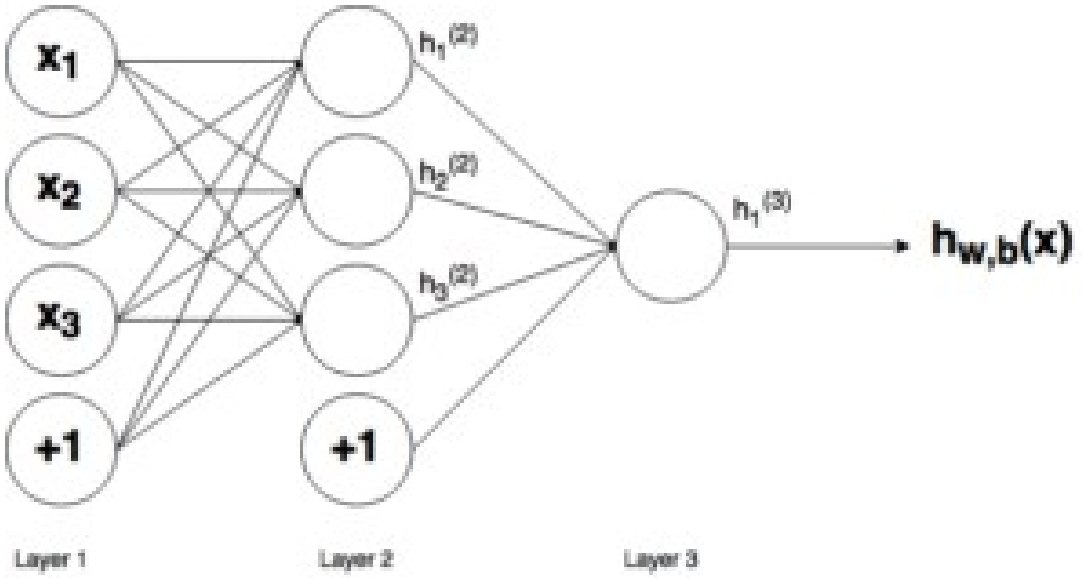
(p.s. don't forget about the non-linearity)

$$
\begin{aligned}
z^{(2)} &= W^{(1)}x + b^{(1)} = \sum_{j=1}^{n} w_{ij}^{(1)} x_i + b_i^{(1)} \\
h^{(2)} &= f(z^{(2)}) \\
z^{(3)} &= W^{(2)}h^{(2)} + b^{(2)}
\end{aligned}
$$

without the non-linearity...

$$
W^{(1+2)} = \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} \end{pmatrix} \begin{pmatrix} w_{11}^{(2)} \\ w_{12}^{(2)} \\ w_{13}^{(2)} \end{pmatrix}
$$

multi-layer networks



All we need to optimize is a weight array for each layer:

$$W^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} \end{pmatrix} \qquad W^{(2)} = \begin{pmatrix} w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)} \end{pmatrix}$$

And we already know how to optimize vectors using evolutionary computation (e.g. hillclimbers)!

This will be the basis of your homeworks for the next couple of weeks

(neural networks are just function approximates)

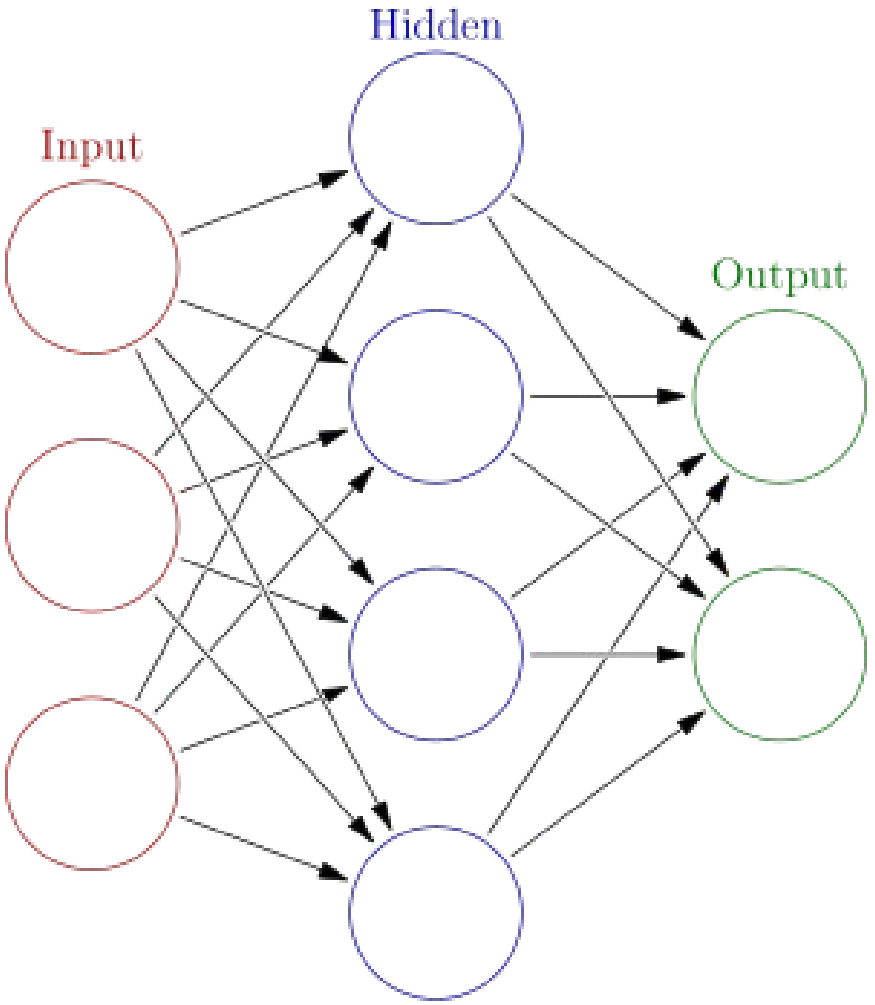
There are also other methods to optimize neural networks, we'll come back to them over the course of the semester

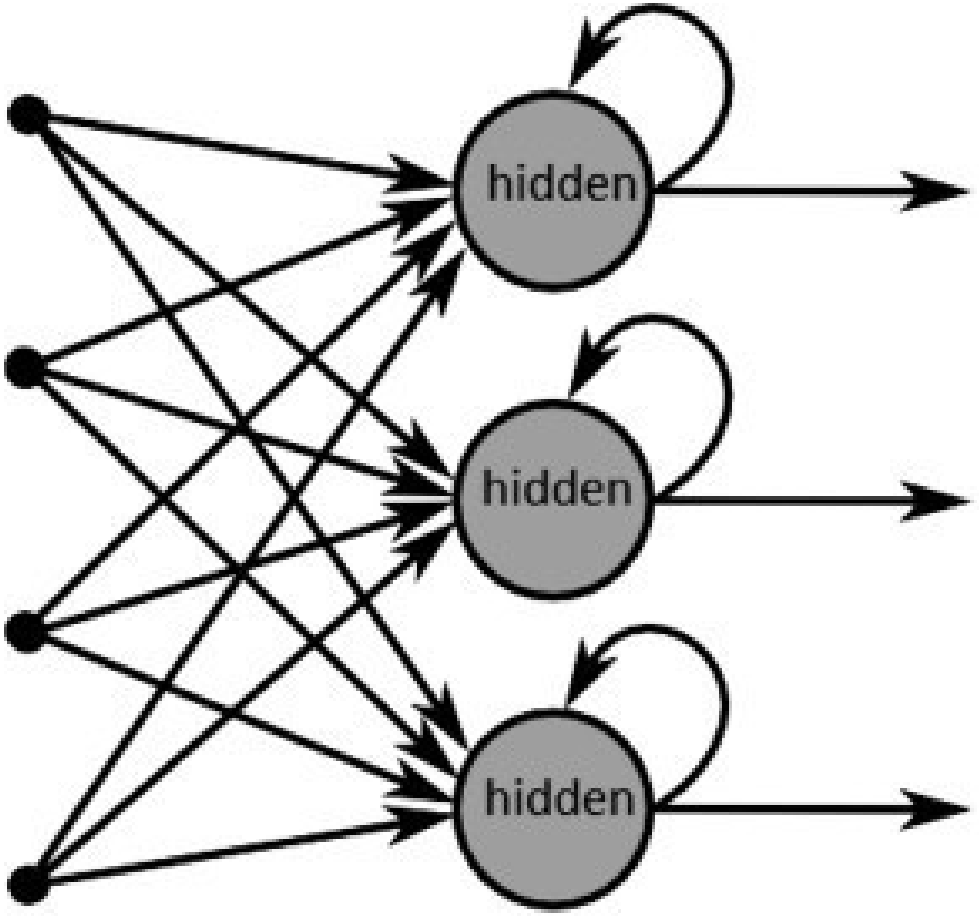
But this is the simplest and most versatile
(if maybe not the most efficient?)

For the next couple months we'll focus on understanding what we can use this simple optimization method for

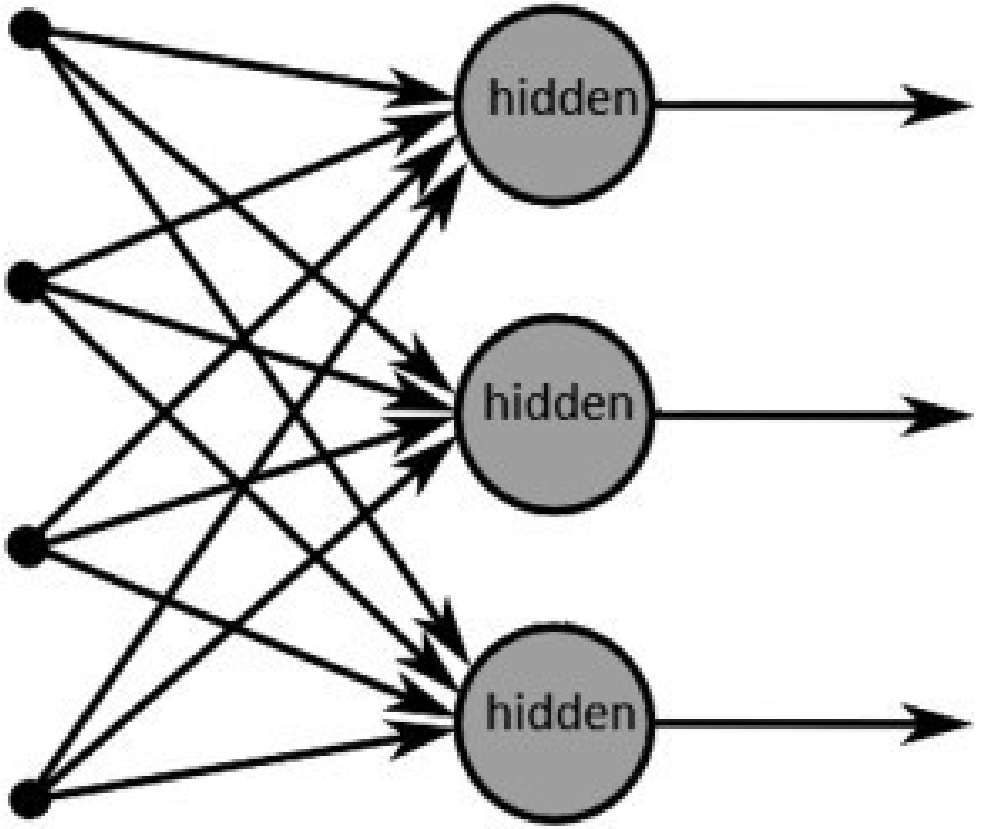
And try just understand just how it works as well as it does (???)

“feedforward” artificial neural network

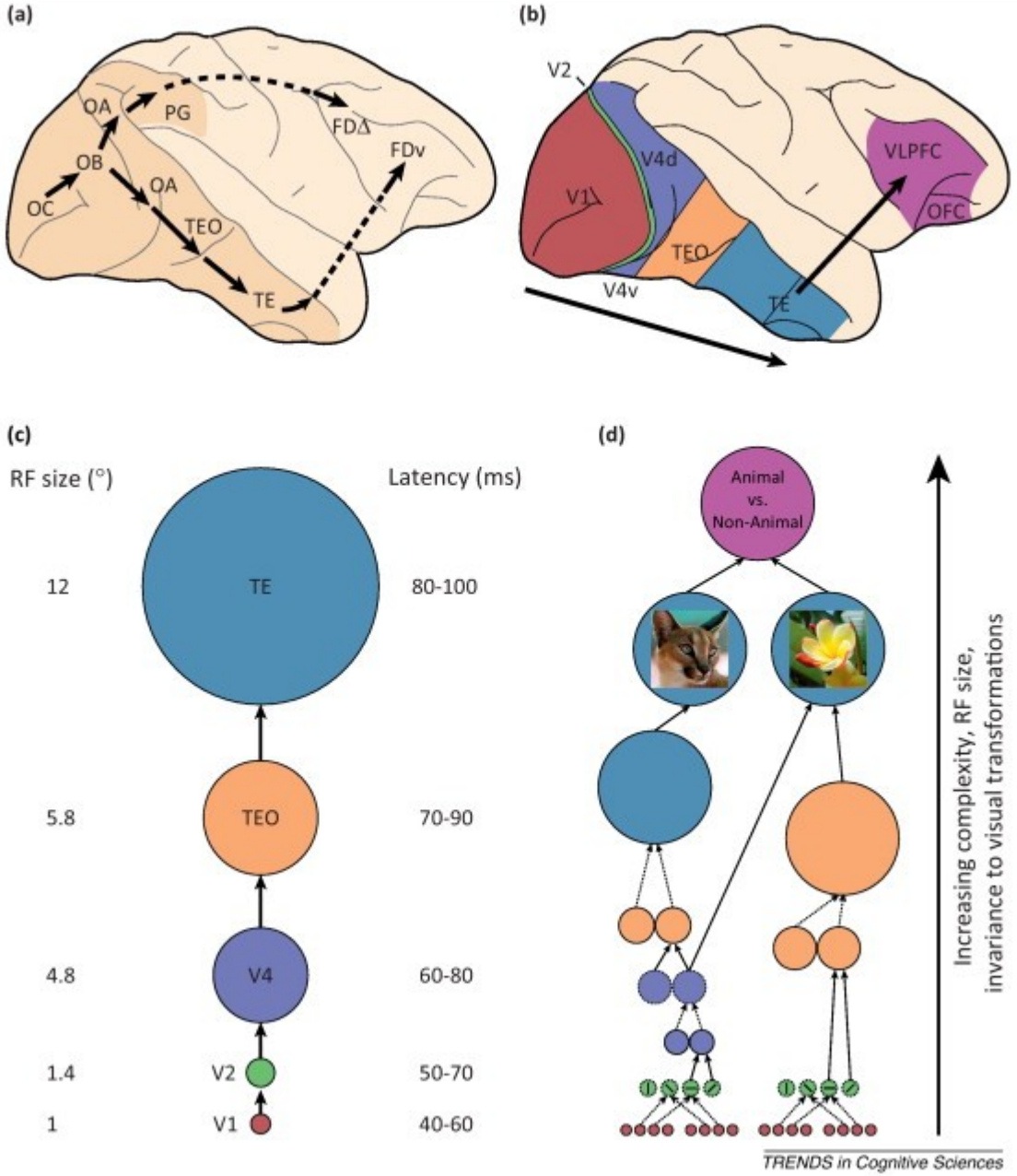


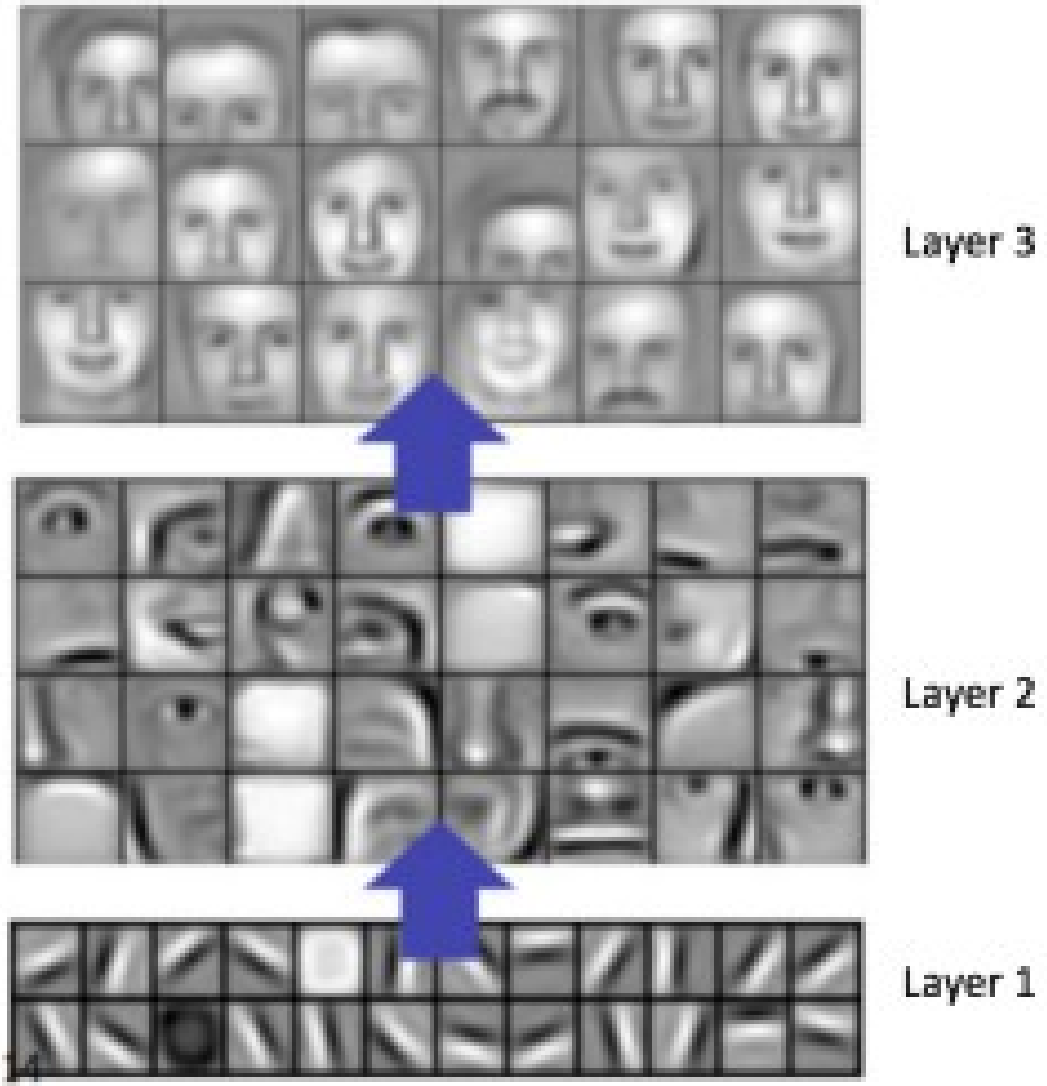


(a) Recurrent neural network



(b) Forward neural network


(a)
OA
PG
FDΔ
FDv
OB
OA
OC
TEO
TE
(b)
V2
V4d
VLPFC
V1
OFC
TEO
V4v
TE
(c)
RF size (°)
Latency (ms)
12
TE
80-100
5.8
TEO
70-90
4.8
V4
60-80
1.4
V2
50-70
1
V1
40-60
(d)
Animal
vs.
Non-Animal
Increasing complexity, RF size,
invariance to visual transformations
TRENDS in Cognitive Sciences


Layer 3
Layer 2
Layer 1