Introduction to Parallel Distributed Processing models of Cognitive Development

Deon T. Benton

001-Basics

To do before next week

- Install Lens
  - First, clone the GitHub repository for this class
  - Second, once cloned, navigate to the folder called "simulation_software" install the version of Lens that is compatible with your machine
    - This should be done *outside* this class session

- Read the first chapter of Marr's book
  - You can download it here: https://1lib.us/book/1223444/8e5ca8

- Read Connectionist Models by Timothy Rogers
  - This can be found in the "readings" of the cloned GitHub folder

Notes on course structure

- The course is divided broadly into two parts
- Part I: Basics of the PDP modeling approach
- Part 2: How this approach has informed developmental science

Let’s start using GitHub

- Download and install the latest version of Git
  - https://git-scm.com/downloads

- Create a GitHub user account
  - https://github.com/join
    - Later, you’ll want to apply for a GitHub Student User account (for advanced features [e.g., having private repositories])

- You’re done!

Understanding complex information-processing systems

- **Marr (1982)**

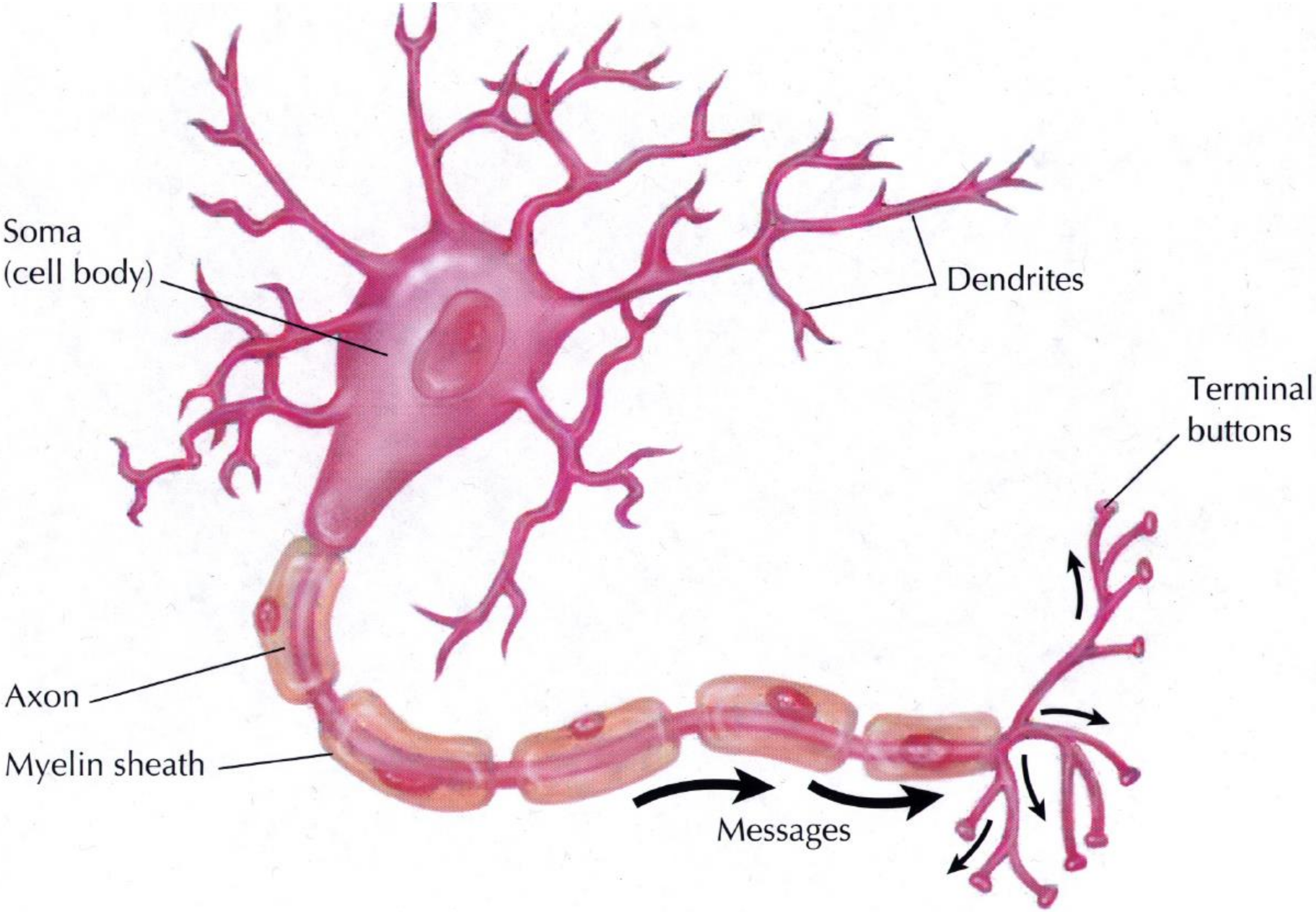
Understanding complex information-processing systems

- **Marr (1982)**

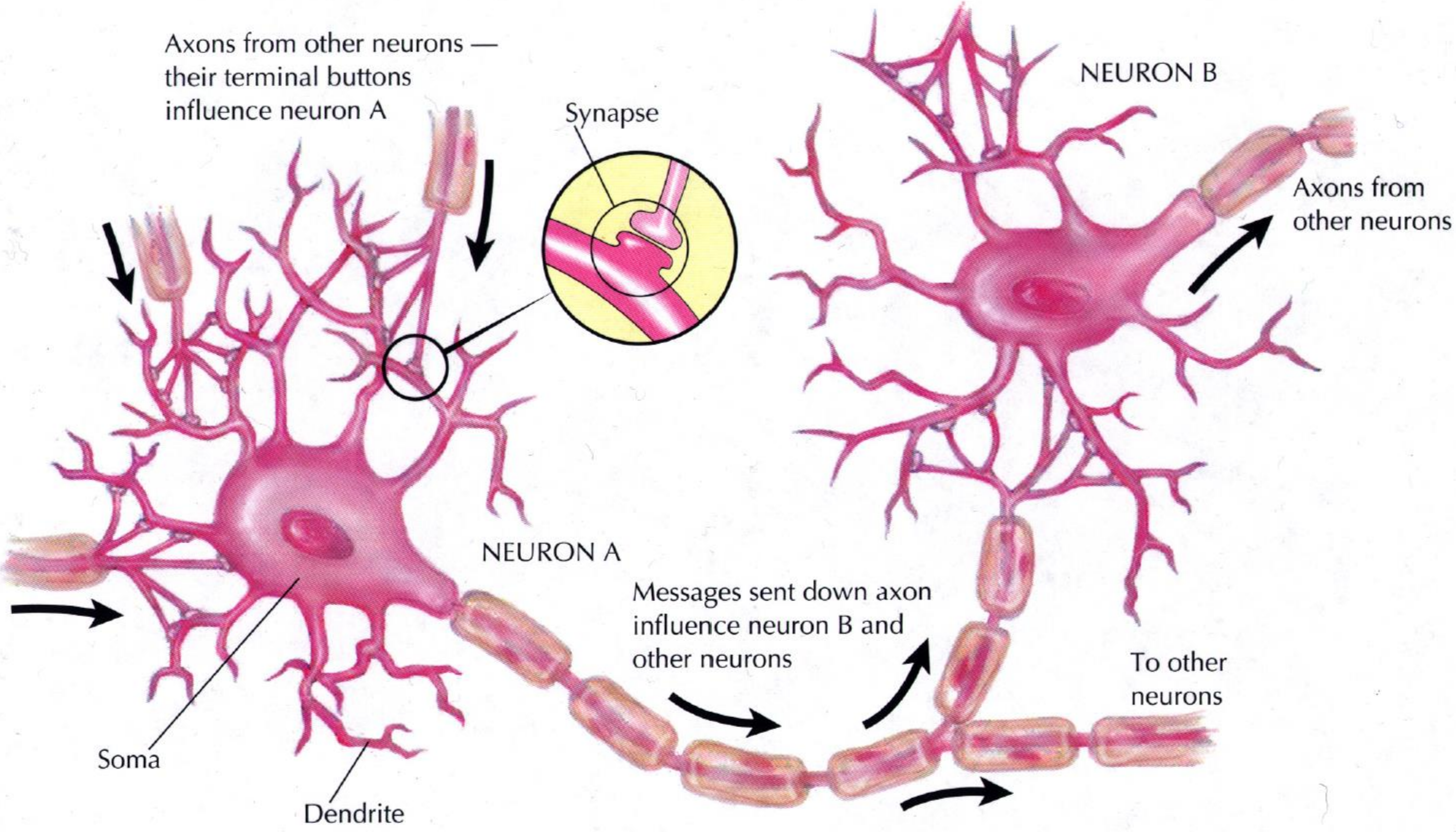
Computational theory
What is the goal of the computation, why is it appropriate, and what is the logic of the strategy by which it can be carried out?

Representation and algorithm
How can this computational theory be implemented? What is the representation for the input and output, and what is the algorithm for the transformation?

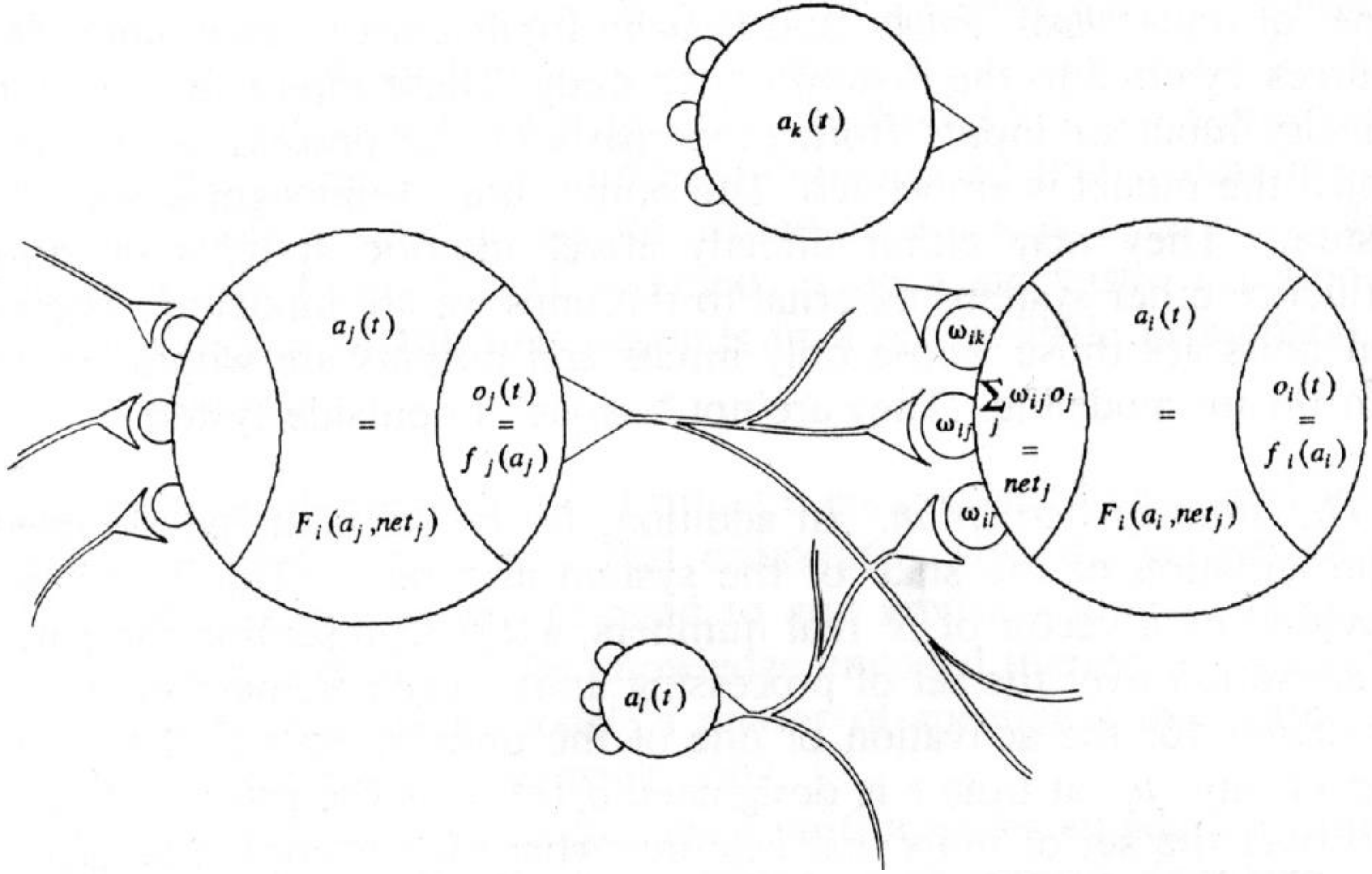
Hardware implementation
How can the representation and algorithm be realized physically?

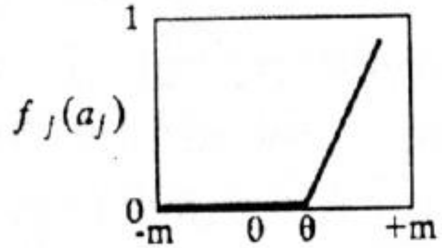


Soma
(cell body)
Dendrites
Terminal
buttons
Axon
Myelin sheath
Messages

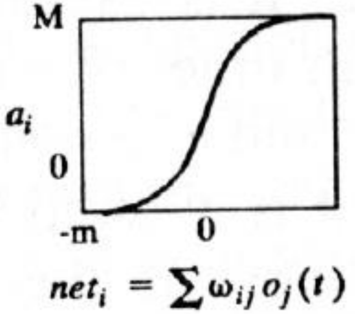


Axons from other neurons — their terminal buttons influence neuron A
Synapse
NEURON B
Axons from other neurons
NEURON A
Messages sent down axon influence neuron B and other neurons
To other neurons
Soma
Dendrite



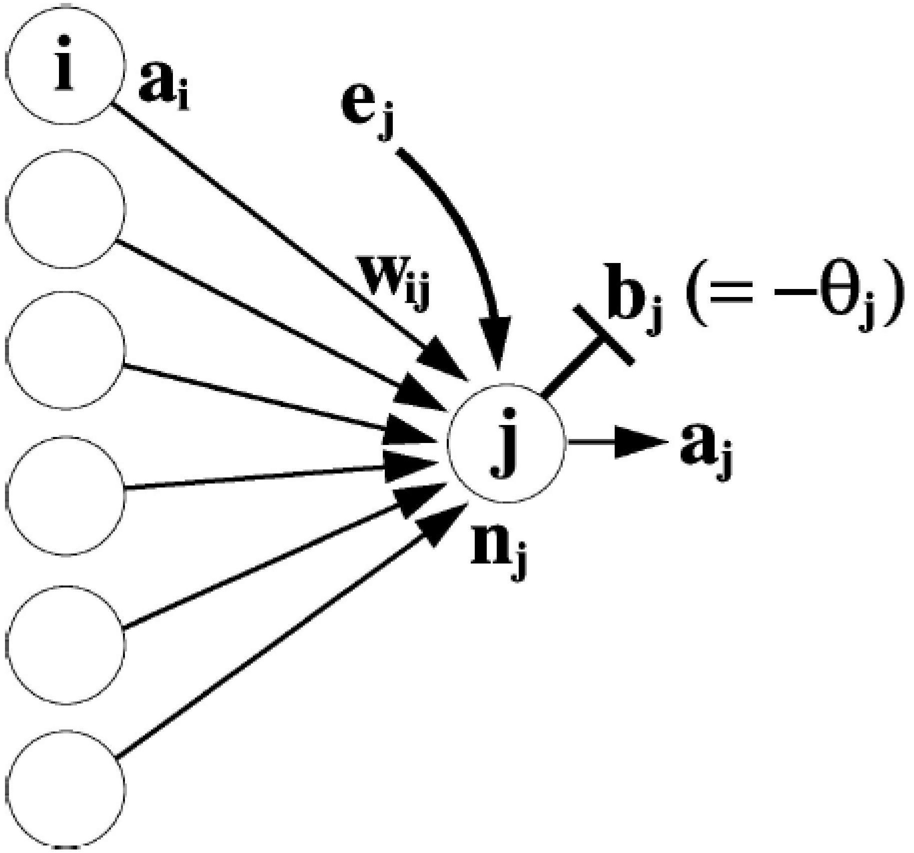


Threshold Output Function



Sigmoid Activation Function

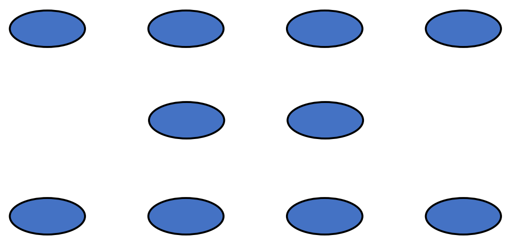
Notation



i, j indices of units (i sending, j receiving)
a_j activation of unit j
n_j summed net input to unit j
w_{ij} weight on connection from unit i to unit j
e_j external input to unit j
b_j bias (tonic input) to unit j (= $-\theta_j$)

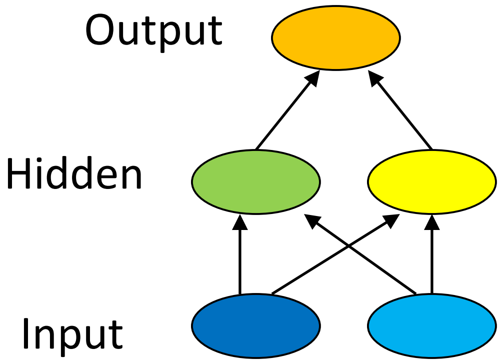
Six elements of connectionist models:

1. A set of units
2. A weight matrix
3. An input function
4. A transfer function
5. A model environment
6. A learning rule

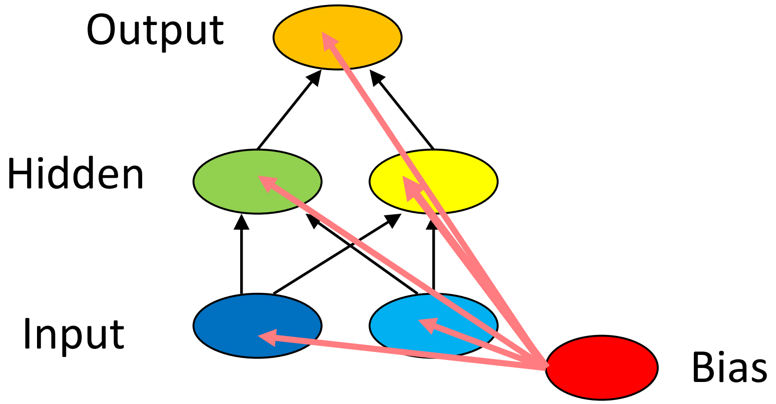


Six elements of connectionist models:

1. A set of units
    - Each unit is like a population of cells with a similar receptive field.
    - Think of all units in a model as a single vector, with each unit corresponding to one element of the vector.
    - At any point in time, each unit has an *activation state* analogous to the mean firing activity of the population of neurons.
    - These activation states are stored in an *activation vector*, with each element corresponding to one unit.



[1 0 .51 .52 .45]



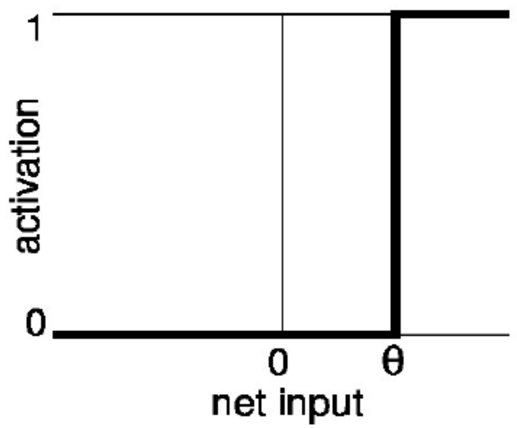
[1 0 .51 .52 .45 1]

Types of units

Binary threshold unit

$$n_j = \sum_i a_i w_{ij} + e_j$$

$$a_j = \begin{cases} 1 & \text{if } n_j > \theta_j \\ 0 & \text{otherwise} \end{cases}$$



If "bias" $b_j = -\theta_j$, this is the same as

$$n_j = \sum_i a_i w_{ij} + e_j + b_j \qquad a_j = \begin{cases} 1 & \text{if } n_j > 0 \\ 0 & \text{otherwise} \end{cases}$$

Will generally omit b_j and e_j in equations

- Bias b_j can be treated as weight w_{ij} from special unit with fixed activation $a_i = 1$.
- External input e_j can be treated as incoming activation a_i across connection with fixed weight $w_{ii} = 1$.

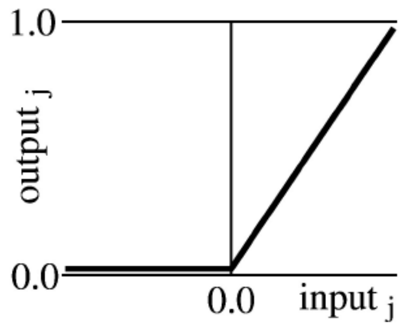
Types of units

Linear units

$$a_j = n_j = \sum_i a_i w_{ij}$$

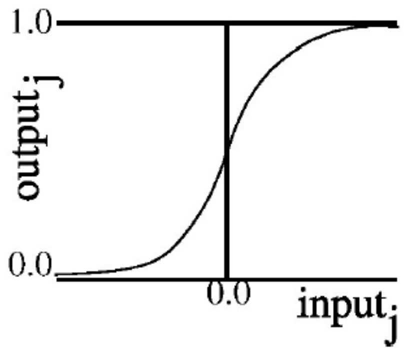
Rectified linear units (ReLUs)

$$a_j = \max(0.0, n_j)$$



Sigmoidal ("logistic", "semi-linear") units

$$a_j = \sigma(n_j) = \frac{1}{1 + \exp(-n_j)}$$



Binary stochastic units

$$p(a_j = 1) = \frac{1}{1 + \exp(-n_j)}$$

Types of units

Continuous time-averaged (cascaded) units [two alternatives]

$$
\begin{aligned}
n_j^{[t]} &= \tau \sum_i a_i^{[t-1]} w_{ij} + (1-\tau)\, n_j^{[t-1]} \\
a_j^{[t]} &= \tau\, \sigma\!\left(n_j^{[t]}\right) + (1-\tau)\, a_j^{[t-1]}
\end{aligned}
$$

Interactive activation

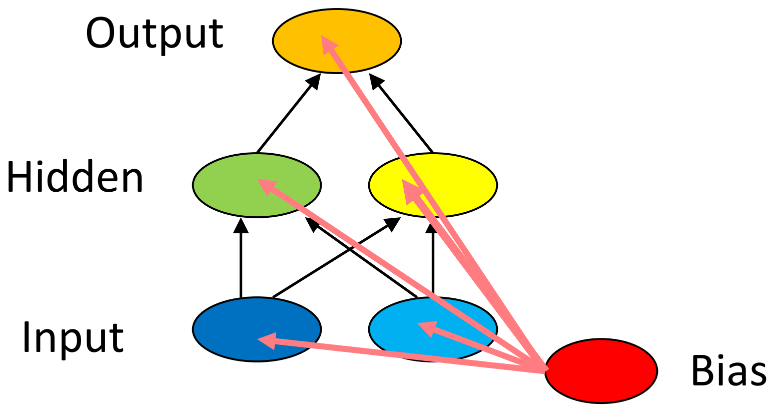
(Jets & Sharks model; Schema model; McClelland & Rumelhart letter/word model)

$$
\begin{aligned}
n_j^{[t]} &= \sum_i a_i^{[t-1]} w_{ij} + e_j^{[t]} \\
a_j^{[t]} &= (1-\mathrm{decay})\, a_j^{[t-1]} + \begin{cases} n_j^{[t]} \left(\max - a_j^{[t-1]}\right) & \text{if } n_j^{[t]} > 0 \\ n_j^{[t]} \left(a_j^{[t-1]} - \min\right) & \text{otherwise} \end{cases}
\end{aligned}
$$

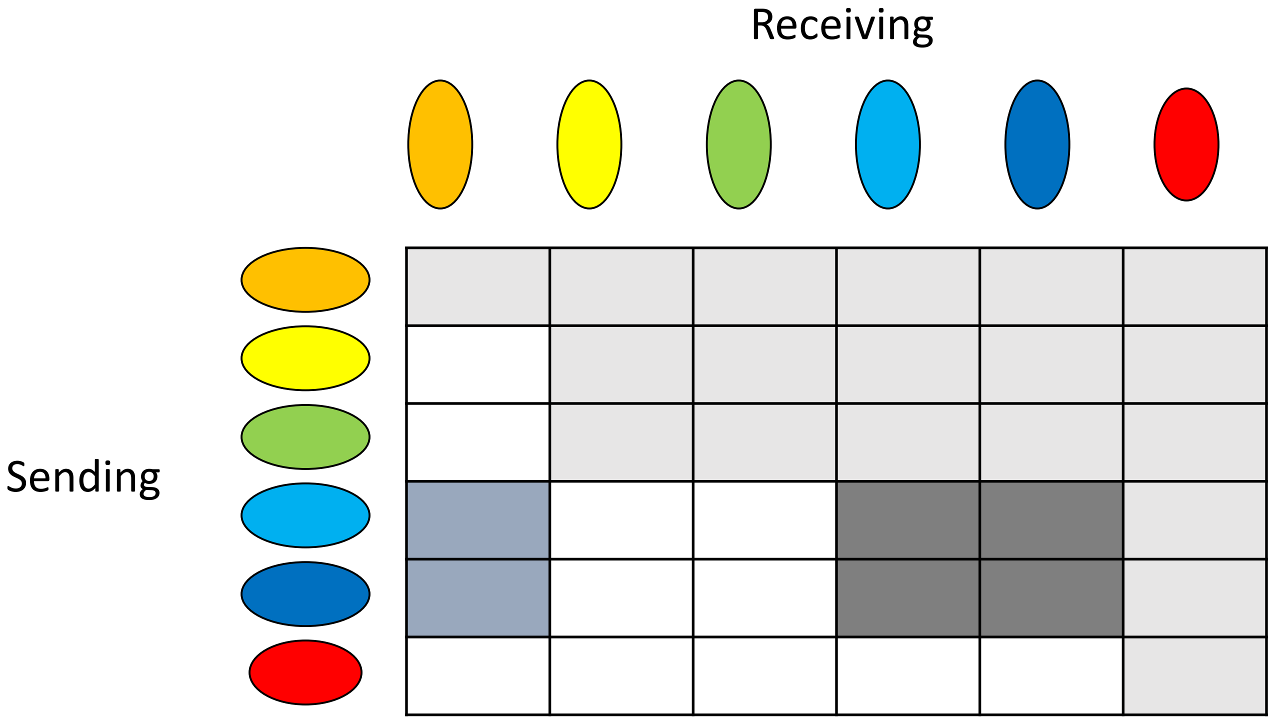
$$
\mathrm{decay} = 0.1 \qquad \max = 1.0 \qquad \min = -0.2
$$

Six elements of connectionist models:

2. A weight matrix
   - Each unit sends and receives a weighted connection to/from some other subset of units.
   - These *weights* are analogous to synapses: they are the means by which one units transmits information about its activation state to another unit.
   - Weights are stored in a *weight matrix*

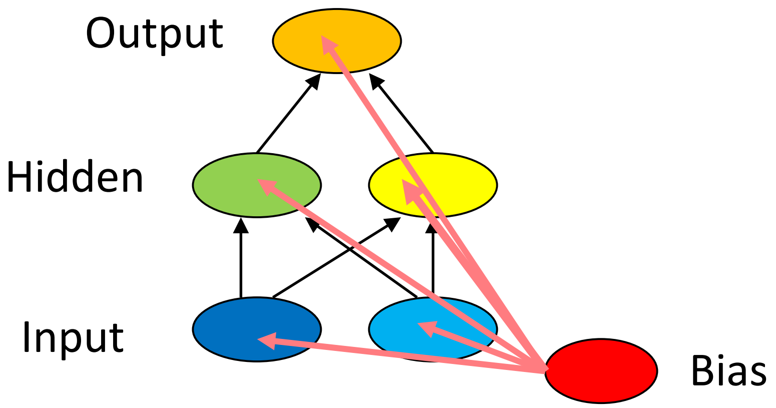


[1 0 .51 .52 .45 1]



Six elements of connectionist models:

3. An *input* function
   - For any given receiving unit, there needs to be some way of determining how to combine weights and sending activations to determine the unit's net input
   - This is almost always the dot product (ie weighted sum) of the sending activations and the weights.



[1 0 ?? ?? ?? 1]

Receiving

Sending

-.2					
-1.8					
	0.5	0.2			
	1.1	-2.2			
.1	.2	.3	.4	.2	

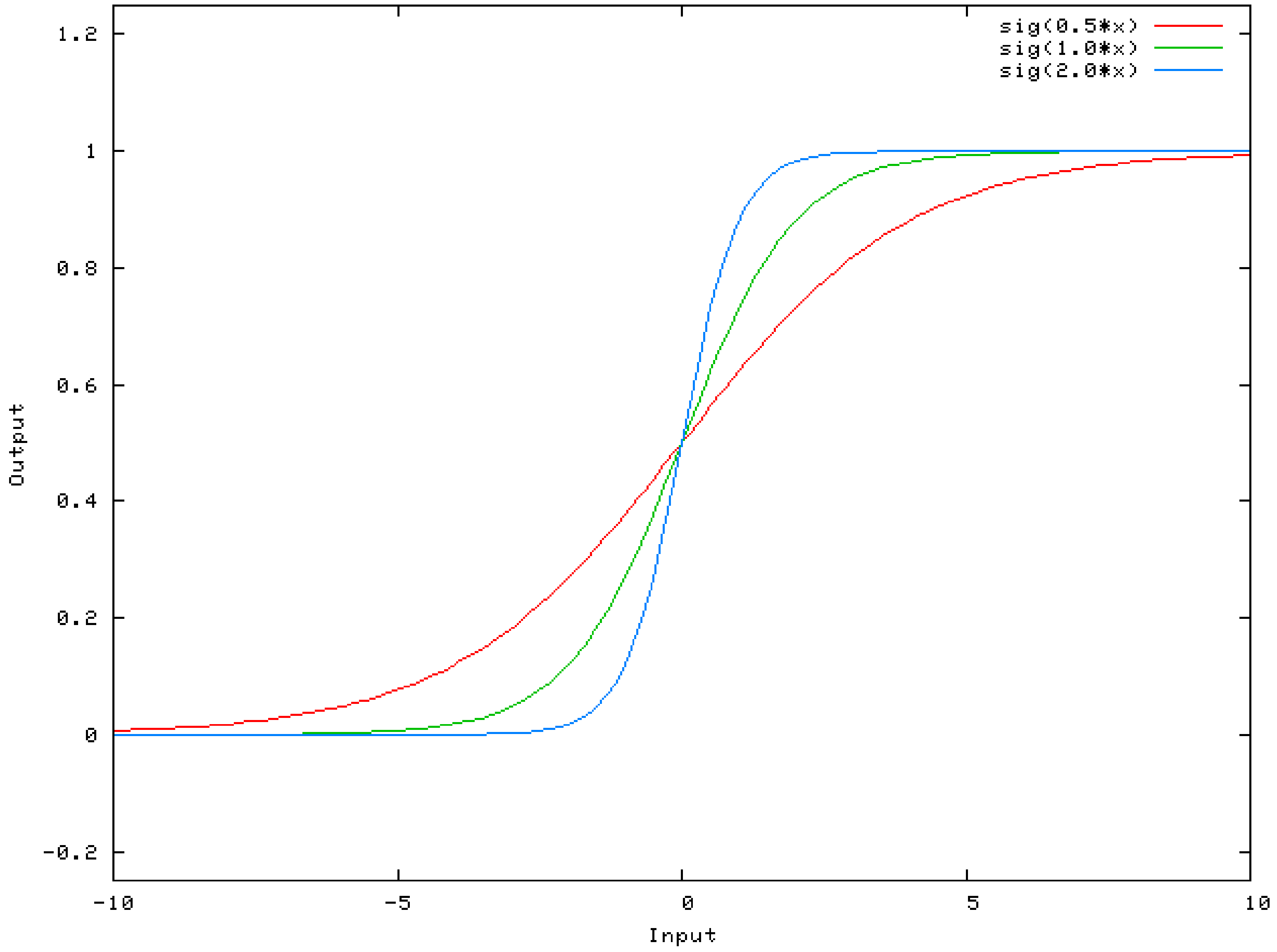
$$net_j = \sum_i a_i w_{ij}$$

$= 1 * .3 + 1 * -2.2 + 0 * 0.22$

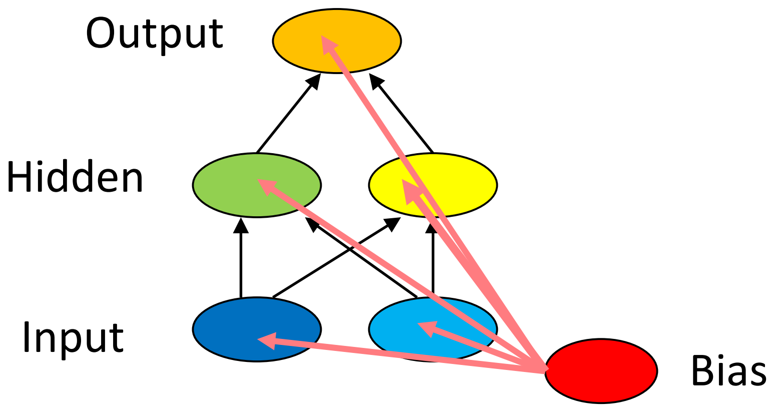
$= -1.7$

Six elements of connectionist models:

4. An *activation* function (or *transfer* function)
   - To determine how a unit should set its activation state for different net inputs, you need to specify an *activation function* $f(net_i)$
   - Lots of possible activation functions:
     - Linear: a = i + c neti
     - Threshold: if net > thresh then a = 1, else a = 0
     - Sigmoid: $\frac{1}{1+ e^{-c\, net_i}}$
     - Etc…



sig(0.5*x)
sig(1.0*x)
sig(2.0*x)
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
Output
-10
-5
0
5
10
Input



[1 0 ?? ?? ?? 1]

Receiving

Sending

-.2					
-1.8					
	0.5	0.22			
	1.1	-2.2			
.1	.2	.3	.4	.2	

$$net_j = \sum_i a_i w_{ij}$$
= 1 * .3 + 1 * -2.2 + 0 * 0.22
= -1.7

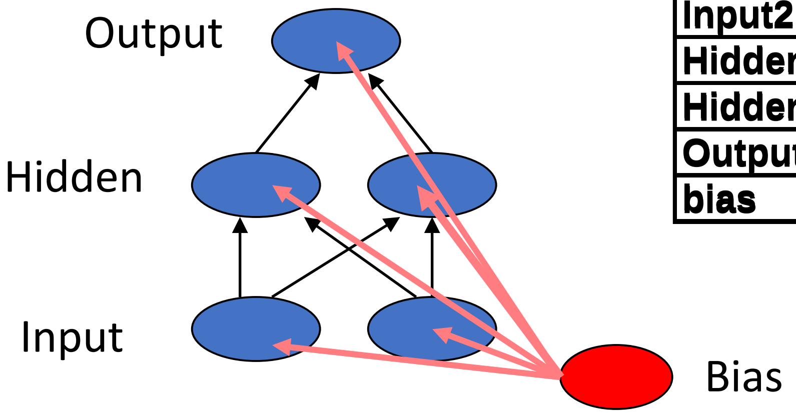
$$a_i = f(net_i)$$
= sig(-1.7)
= 0.15

Six elements of connectionist models:

5. A *model environment*
    - All the models do is compute activation states over units, given the preceding elements and some partial input.
    - The model environment specifies how events in the world are encoded in unit activation states, typically across a subset of units.
    - It consists of vectors that describe the input activations corresponding to different events, and sometimes the “target” activations that the network should generate for each input.

X-OR function

In1	In2	Out
0	0	0
0	1	1
1	0	1
1	1	0



	Input1	Input2	Hidden1	Hidden2	Output
Input1					
Input2					
Hidden1					
Hidden2					
Output					
bias					

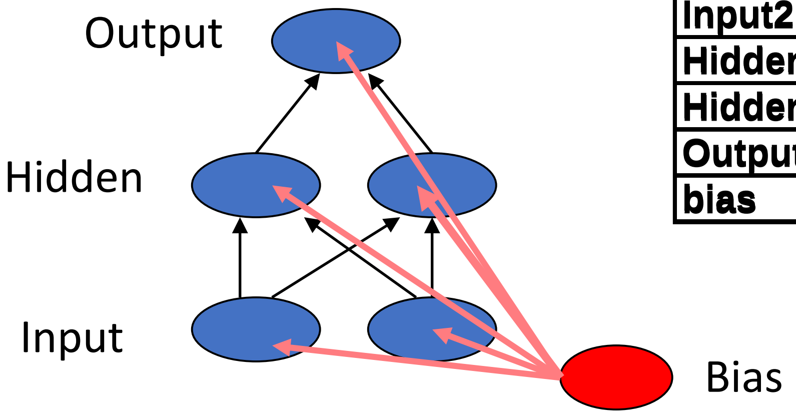
- Note that the model environment is always theoretically important!
- It amounts to a theoretical statement about the nature of the information available to the system from perception and action or prior cognitive processing.
- Many models sink or swim on the basis of their assumptions about the nature of inputs / outputs.

Six elements of connectionist models:

6. A *learning rule*
   - Only specified for models that learn (obviously)
   - Specifies how the values stored in the weight matrix should change as the network processes patterns
   - Many different varieties that we will see:
     - Hebbian
     - Error-correcting (e.g. backpropagation)
     - Competitive / self-organizing
     - Reinforcement-based

X-OR function

In1	In2	Out
0	0	0
0	1	1
1	0	1
1	1	0



	Input1	Input2	Hidden1	Hidden2	Output
Input1					
Input2					
Hidden1					
Hidden2					
Output					
bias					