Introduction to Parallel Distributed Processing models

Deon T. Benton

001-Basics

To do before next week

- Install Lens
  - First, git clone https://github.com/dtbenton/pdp_summer_tutorial_2021 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

- What is Git?
  - It’s a version control system that is used to track changes to local computer files

- Why use Git?
  - Common tasks are tedious without it
  - All the cool kids are using it, duh!

- Why is Git relevant for this course?
  - All “course” materials will be housed on a remote GitHub repository, which will require you to ***git pull*** several times

Installing Git

- Linux (Debian)
  - $ sudo apt-get install git
- Linux (Fedora)
  - $ sudo yum install git
- Mac
  - https://git-scm.com/download/mac
- Windows
  - https://git-scm.com/download/win

Confirm that Git is installed

- In the command line (win) or terminal (Mac), type in: git –version
- Create a new project on your local machine and add a file to it
  - Once created and within that folder, initialize the folder as a local Git repository
    - git init
- Add name and email to Git (use the same email from GitHub)
  - git config --global user.name “Your Name Here”
  - git config --global user.email “Your Email Here”
- Stage the files in your local repository
  - git add *
  - git commit -m “some message here”

Working with a remote repository on GitHub

- Once logged into GitHub, click "+" in the top right of the screen to create a new remote repository**
    - Give it a name and make it "Public"
    - No need to initialize with a README
    - Click "Create repository"
- Follow the steps to add the remote repository to your local machine
    - Note that you will be prompted to enter the username and password you used to sign up to GitHub

**You must have a GitHub account to complete all of the above

Git Commands to Know

- git init – initializes a Local Git Repository
- git add <file name> OR git add * — adds files to a staging area
- git status — checks the status of the local directory
- git commit –m “message here” — captures a snapshot of the staging area
- git push — sends the snapshot of those changes to a remote repository on Github
- git pull — pulls latest “snapshots” from a remote repository to a local one
- git clone <URL of remote repository> — clones a remote repository to your local machine (what you need to do to access the content for this course)

A linear algebra primer

- Scalar: a single number (or dimension)

- Vector: A list of numbers where each number corresponds to a dimension in an n-dimensional space and the set of numbers corresponds to a point in that space
  - [2 3 1]
  - [3 4 5 6]
  - [4 5 3 … 10 5 40]

- Matrix: A collection of **row and column** vectors
  - $$\begin{matrix} 2 & 0 & 6 \\ 3 & 1 & 0 \\ 5 & 0 & 1 \end{matrix}$$

A linear algebra primer

- Scalar: a single number (or dimension)

- Vector: A list of numbers where each number corresponds to a dimension in an n-dimensional space and the set of numbers corresponds to a point in that space
  - [2 3 1]
  - [3 4 5 6]
  - [4 5 3 … 10 5 40]

- Matrix: A collection of **row and column** vectors
  - $$\begin{matrix} 2 & 0 & 6 \\ 3 & 1 & 0 \\ 5 & 0 & 1 \end{matrix}$$

- Span: the ability of the linear combination of two (or more) vectors to recreate any other vector in some n-dimensional space

A linear algebra primer

- Vector/vector addition

[3 4 5 6] + [10 2 0 3]

- Matrix/matrix addition

$$\begin{matrix} 2 & 0 & 6 \\ 3 & 1 & 0 \\ 5 & 0 & 1 \end{matrix} + \begin{matrix} 8 & 0 & 6 \\ 3 & 1 & 0 \\ 1 & 10 & 1 \end{matrix}$$

A linear algebra primer

- Vector/vector addition

[3 4 5 6] + [10 2 0 3]

- Matrix/matrix addition

$$\begin{matrix} 2 & 0 & 6 \\ 3 & 1 & 0 \\ 5 & 0 & 1 \end{matrix} + \begin{matrix} 8 & 0 & 6 \\ 3 & 1 & 0 \\ 1 & 10 & 1 \end{matrix}$$

A linear algebra primer

- Scalar/vector multiplication

21*[3 4 5 6]

- Scalar/matrix multiplication

$$2* \begin{matrix} 8 & 0 & 6 \\ 3 & 1 & 0 \\ 1 & 10 & 1 \end{matrix}$$

- Matrix/matrix multiplication

$$\begin{matrix} 2 & 0 & 6 \\ 3 & 1 & 0 \\ 5 & 0 & 1 \end{matrix} \times \begin{matrix} 8 & 0 & 6 \\ 3 & 1 & 0 \\ 1 & 10 & 1 \end{matrix}$$

A linear algebra primer

- Vector/matrix multiplication

$$[3\ 4\ 5\ 6] * \begin{matrix} 2 & 0 & 6 \\ 3 & 1 & 0 \\ 5 & 0 & 1 \end{matrix}$$

Understanding complex information-processing systems

- **Marr (1982)**

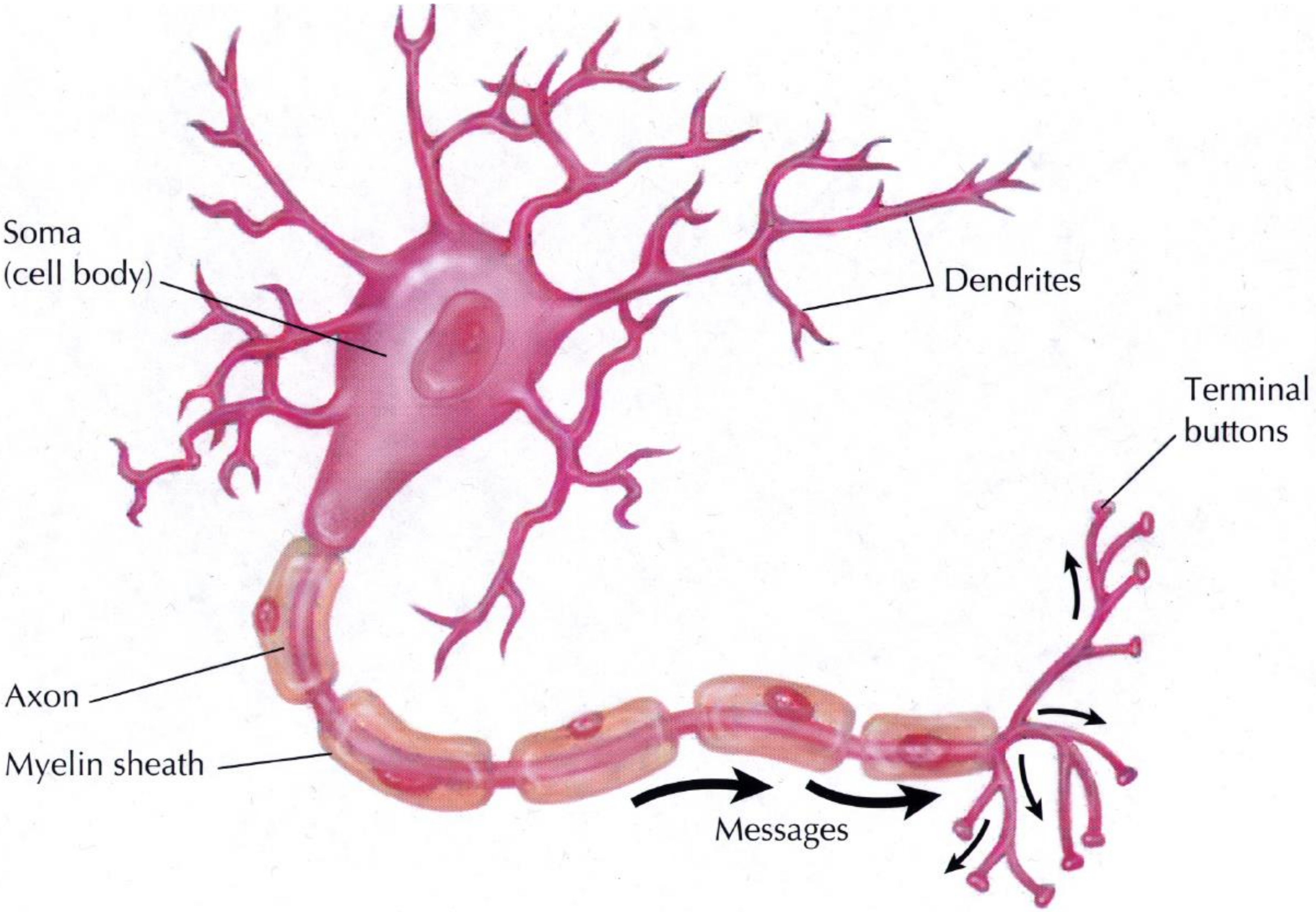
Understanding complex information-processing systems

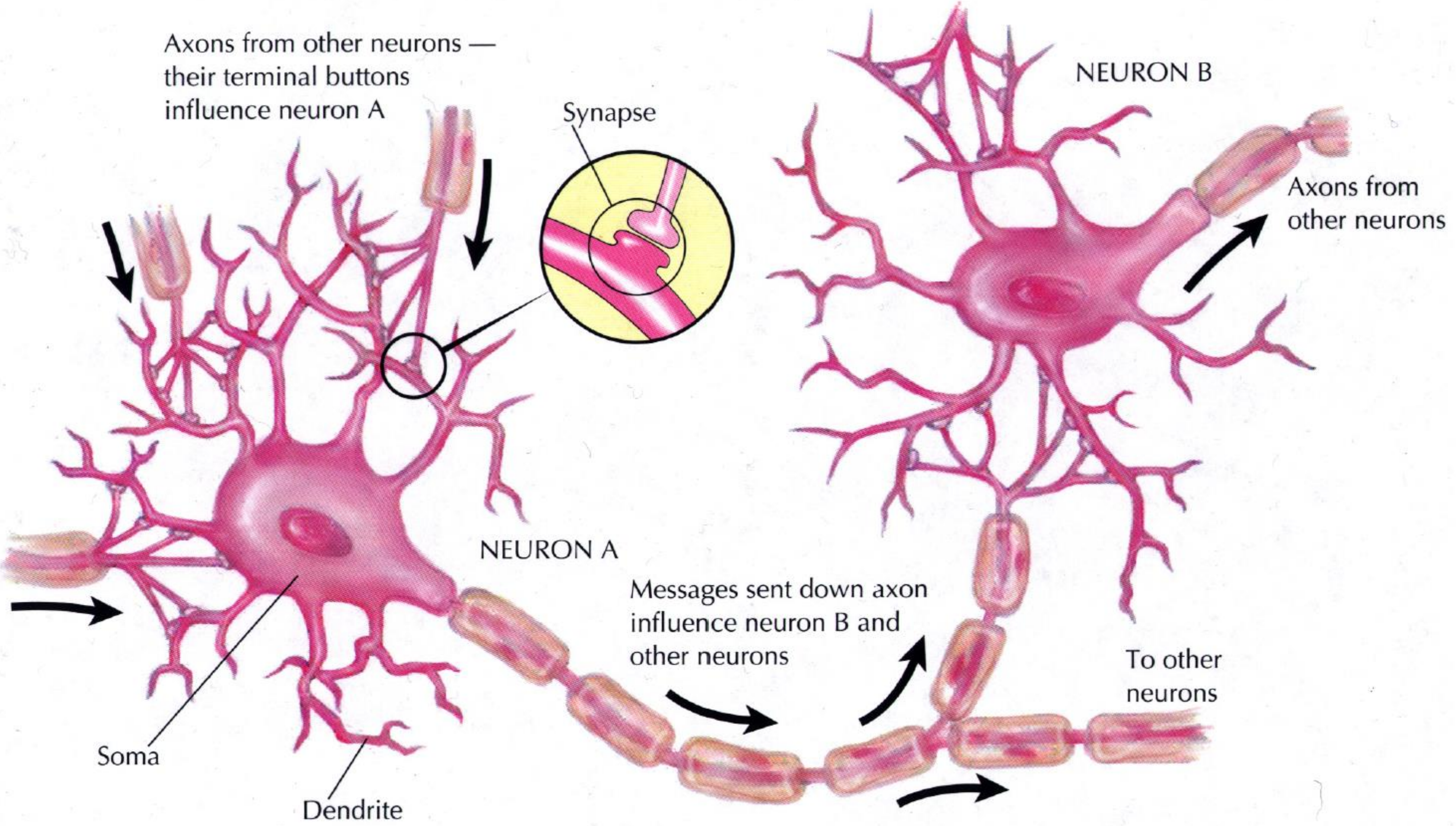
- **Marr (1982)**

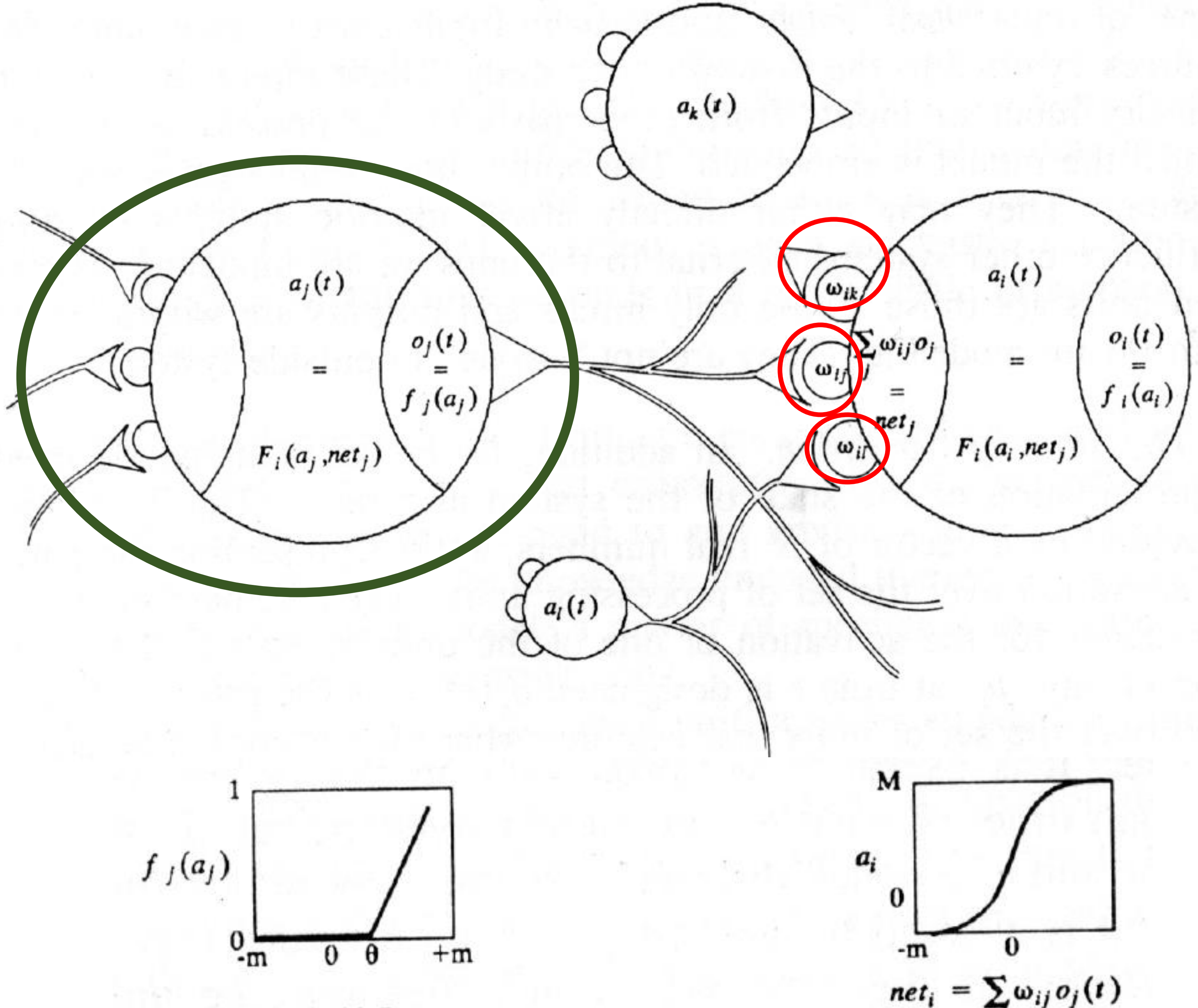
Computational theory
What is the goal of the computation, why is it appropriate, and what is the logic of the strategy by which it can be carried out?

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

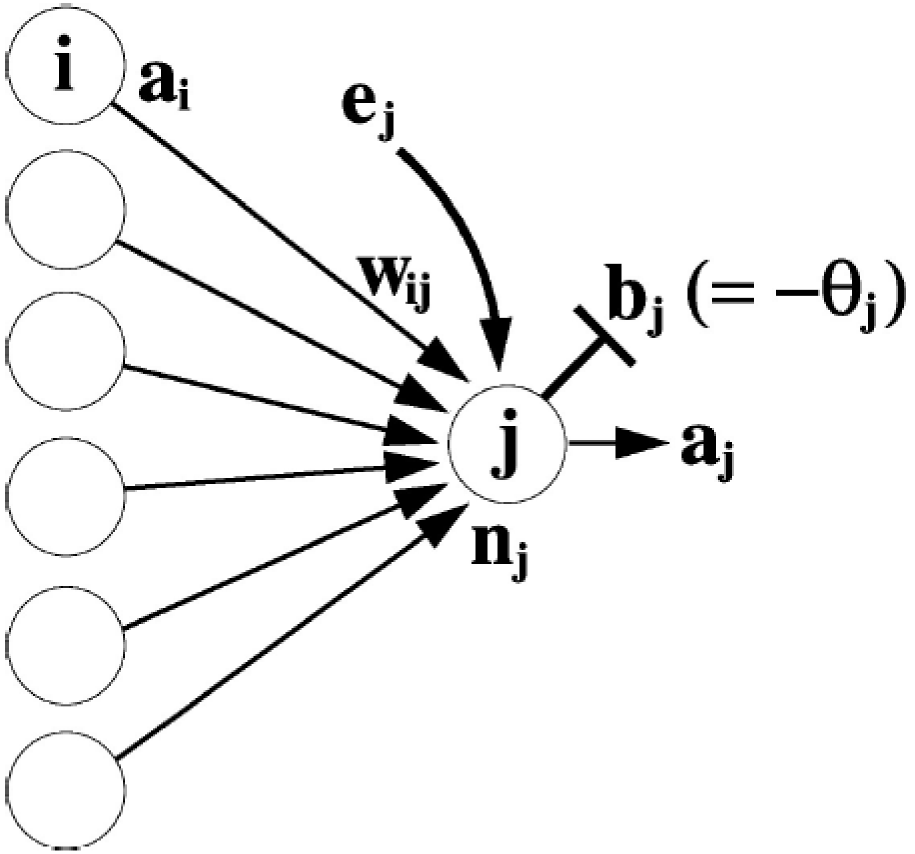

Axons from other neurons —
their terminal buttons
influence neuron A
Synapse
NEURON B
Axons from
other neurons
NEURON A
Messages sent down axon
influence neuron B and
other neurons
To other
neurons
Soma
Dendrite



Threshold Output Function

Sigmoid Activation Function

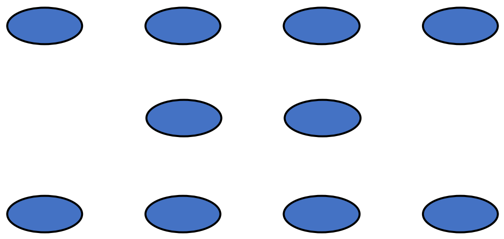
Notation



i, j indices of units (i sending, j receiving)
a_i activation of (sending) unit i
a_j activation of (receiving) unit j
n_j summed net input to (receiving) 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 (= $-\theta_j$)

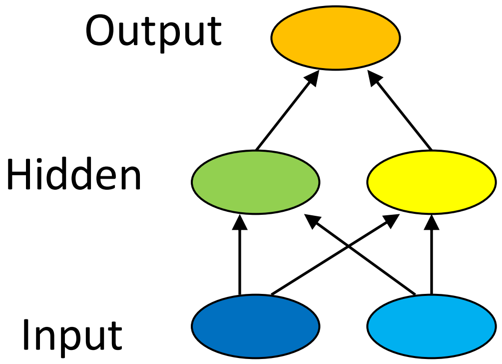
Six elements of connectionist models:

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

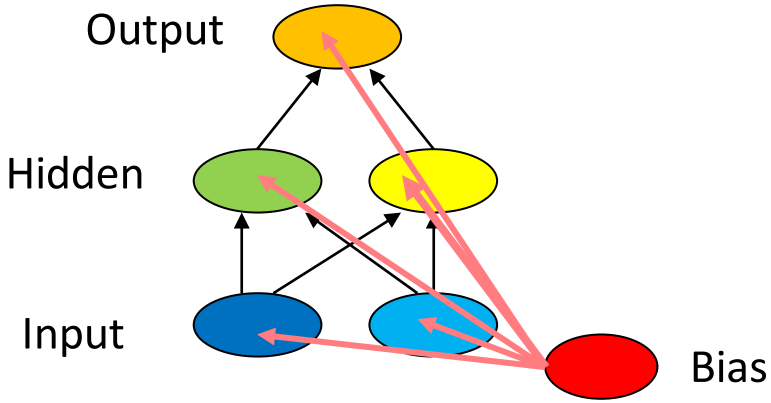


Six elements of connectionist models:

1. A set of units
   - Units are organized into layers
   - Think of the *collection* of 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 the activation of one unit.



[1 0 .51 .52 .45]



[1 0 .51 .52 .45 1]

Types of units

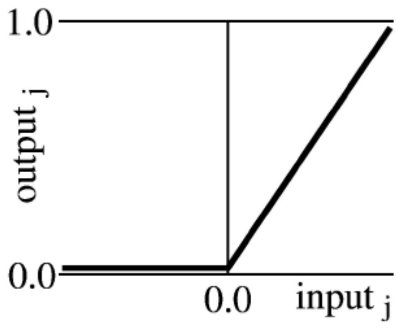
$$n_j = \sum_i a_i w_{ij} + e_j + b_j$$

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

Types of units

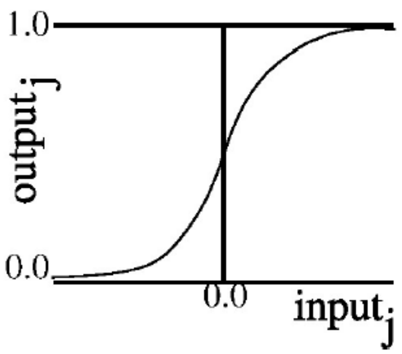
Linear units

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



Rectified linear units (ReLUs)

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



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

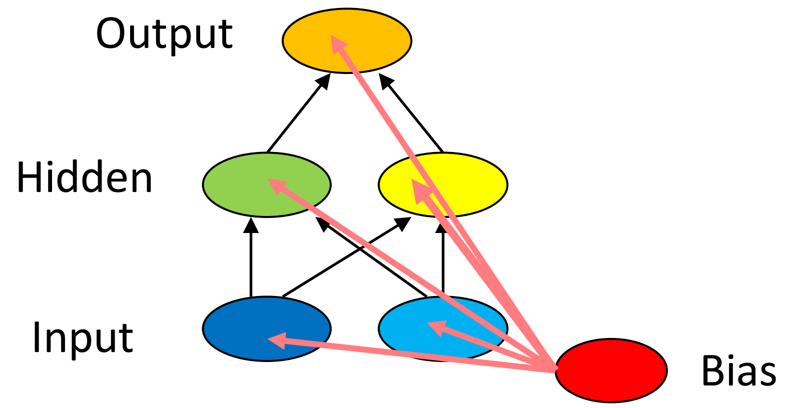
$$a_j = \sigma(n_j) = \frac{1}{1 + \exp(-n_j)}$$

Binary stochastic units

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

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]

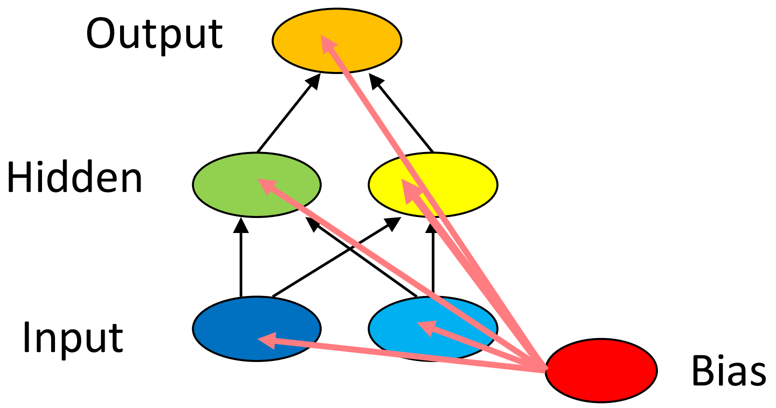
Receiving

Sending

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

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]

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

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

$= -1.7$

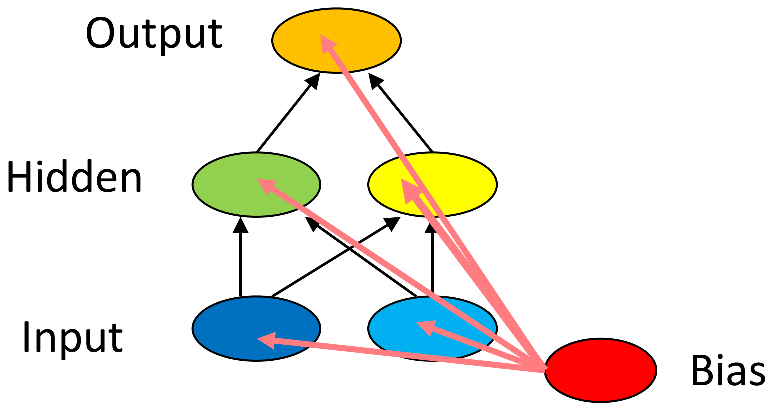
Receiving

Sending

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

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…



[1 0 ?? ?? ?? 1]

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

Receiving

Sending

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

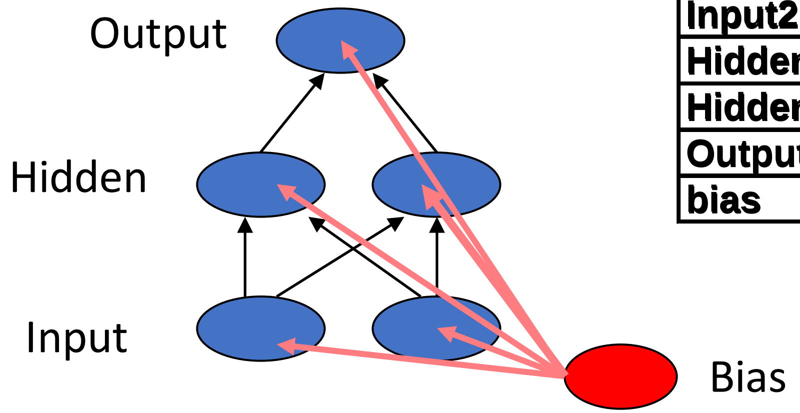
$$a_i = f(net_i)$$
= sig(-1.7)
= 0.15

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					

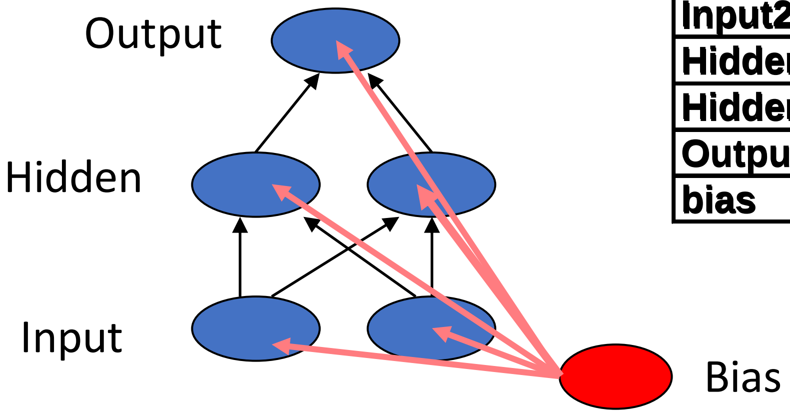
- Note that the model environment is always theoretically important!
- It amounts to a theoretical statement about the nature of the information available to the system from perception and action or prior cognitive processing.
- Many models sink or swim on the basis of their assumptions about the nature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