Marr's levels of analysis

3 levels of analysis for information processing systems

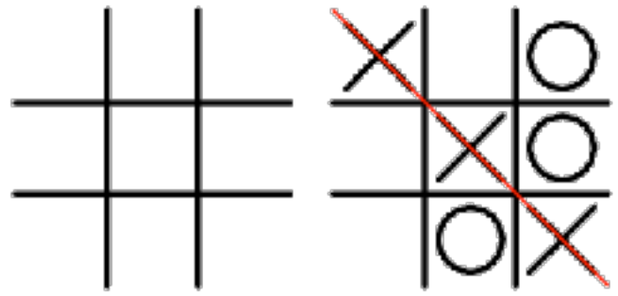
- **Computational level**
  - What is the goal of the computation, why is it appropriate, and what is the logic of the strategy by which it can be carried out?

- **Algorithmic level**
  - How can this computational theory be implemented?
  - In particular, what is the representation of the input and output, and what is the algorithm for the transformation?

- **Implementational level**
  - How can the representation and algorithm be realized physically?
  - e.g., neural mechanisms that implement the algorithm.

Marr, D. (1982). Vision: A computational investigation into the human representation and processing of visual information. MIT Press. *Cambridge, Massachusetts*.

Example: tic-tac-toe



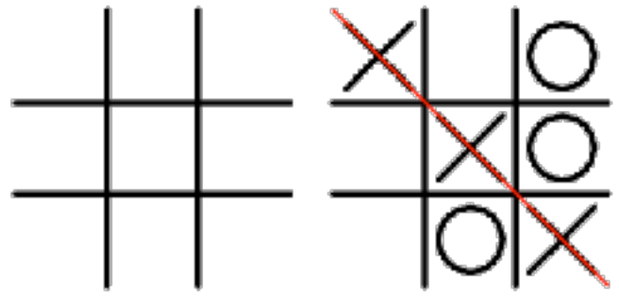
3 levels of analysis for information processing systems

- **Computational level**
  - 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?
    - *Goal: Achieve three in a row, while preventing your opponent from reaching three in a row.*

- **Algorithmic level**
  - How can this computational theory be implemented?
  - In particular, what is the representation of the input and output, and what is the algorithm for the transformation?

- **Implementational level**
  - How can the representation and algorithm be realized physically?
  - e.g., neural mechanisms that implement the algorithm.

Example: tic-tac-toe



3 different types of analysis of an information-processing system

- **Algorithmic level**
  - How can this computational theory be implemented?
  - In particular, what is the representation of the input and output, and what is the algorithm for the transformation?
    - *See rules below, or any other computer program*

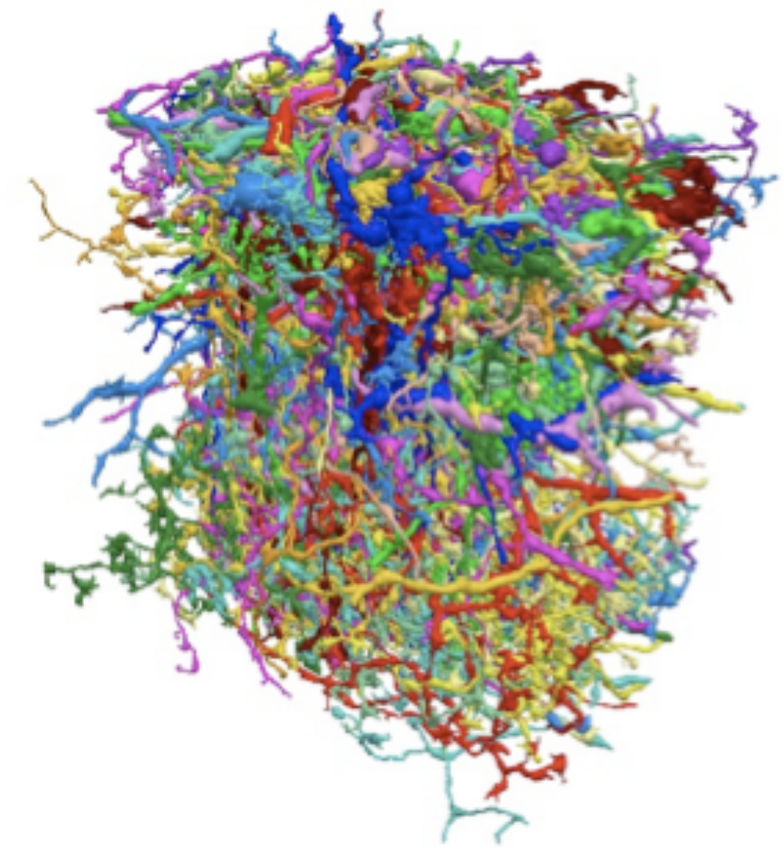
An example of one algorithm (but there are an infinite number of others)…

Simon's 1973 tic-tac-toe program for playing a perfect game

1. **Win**: If the player has two in a row, they can place a third to get three in a row.
2. **Block**: If the opponent has two in a row, the player must play the third themselves to block the opponent.
3. **Fork**: Create an opportunity where the player has two threats to win (two non-blocked lines of 2).
4. **Blocking an opponent's fork**: If there is only one possible fork for the opponent, the player should block it. Otherwise, the player should block any forks in any way that simultaneously allows them to create two in a row. Otherwise, the player should create a two in a row to force the opponent into defending, as long as it doesn't result in them creating a fork. For example, if "X" has two opposite corners and "O" has the center, "O" must not play a corner in order to win. (Playing a corner in this scenario creates a fork for "X" to win.)
5. **Center**: A player marks the center. (If it is the first move of the game, playing on a corner gives the second player more opportunities to make a mistake and may therefore be the better choice; however, it makes no difference between perfect players.)
6. **Opposite corner**: If the opponent is in the corner, the player plays the opposite corner.
7. **Empty corner**: The player plays in a corner square.
8. **Empty side**: The player plays in a middle square on any of the 4 sides.

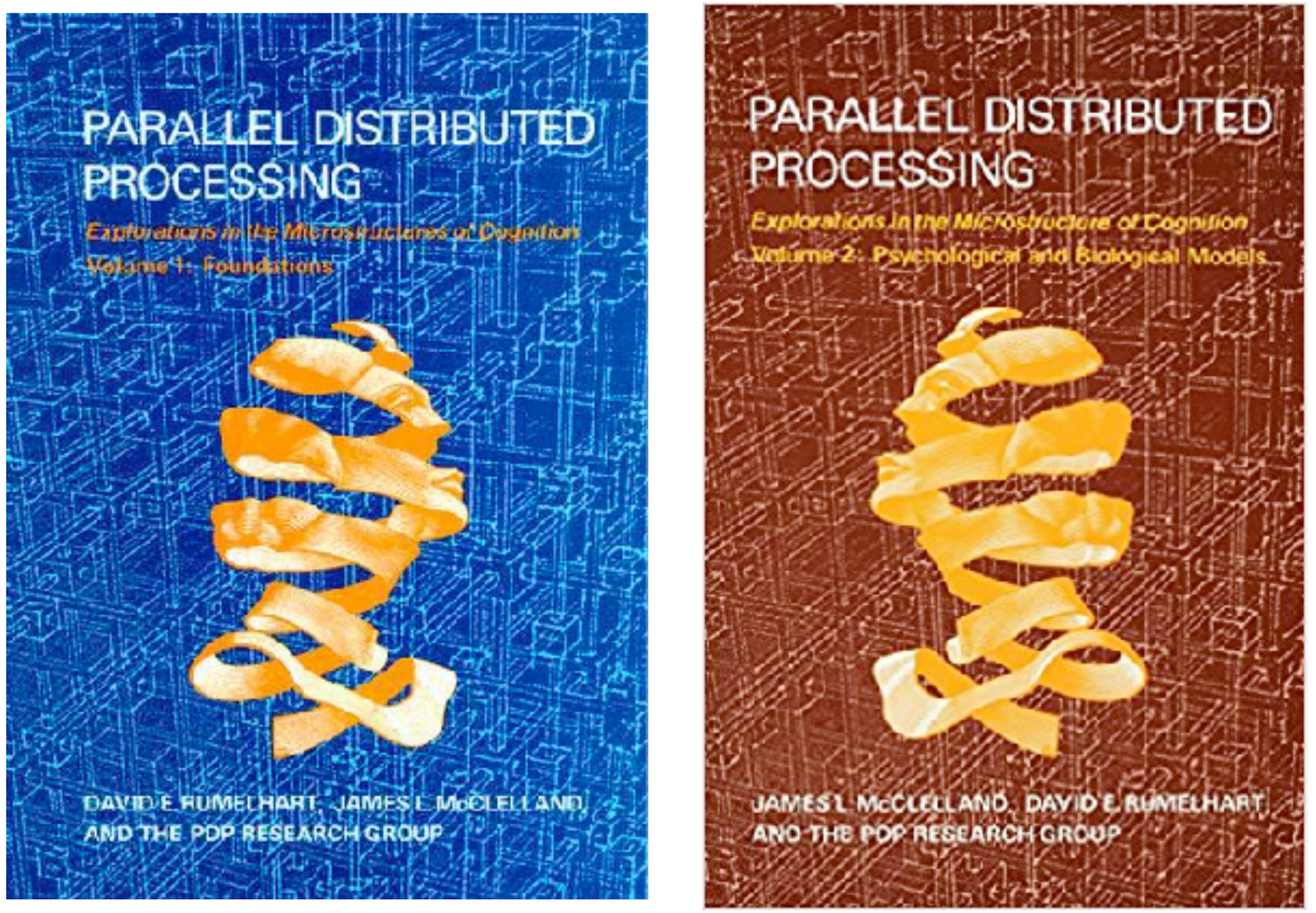
Example: tic-tac-toe

- **Implementational level**
  - How can the representation and algorithm be realized physically?
  - e.g., neural mechanisms that implement the algorithm.
    - *neurons, circuits, tinker toys, etc. are all possibilities*



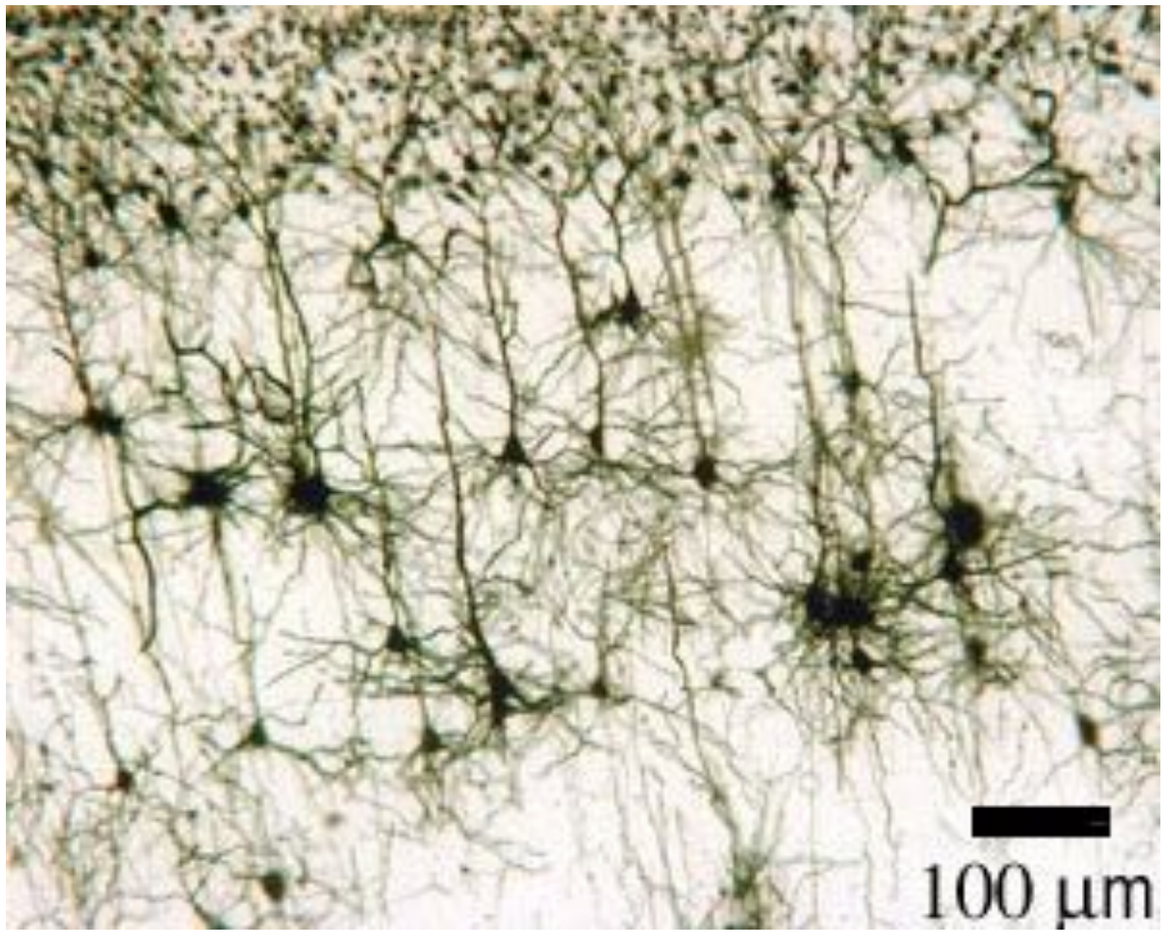
(Thanks Josh Tenenbaum for the example)

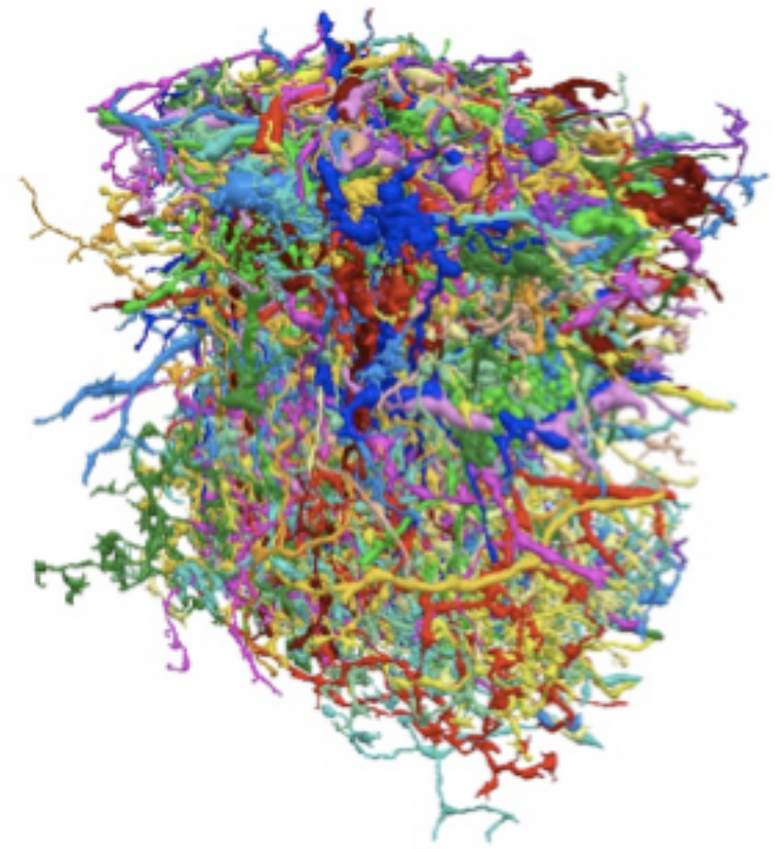
Parallel distributed processing (PDP): Neural network models of cognition

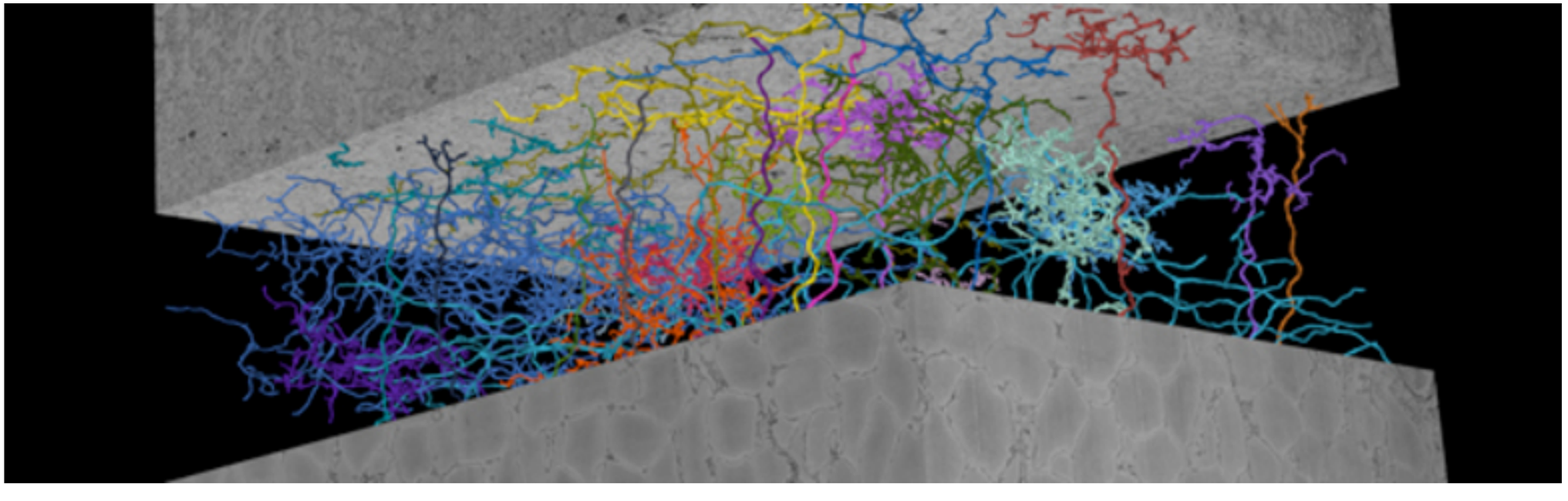


David Rumelhart, James McClelland, and the PDP Research Group (1986)

What kind of computer is the mind?







Key principles of neural network models of cognition

- *Neurally-inspired computation.* Taking inspiration from the low-level (how neurons compute) is key to understanding the high-level (intelligence and cognition)

- *Intelligence is an emergent phenomenon.* Complex behavior can emerge from a very large number of simple, interactive computations.

- *Simulation is central.* It's hard to predict how complexity will emerge. Computational modeling and simulation are essential for understanding intelligence.

A biological neuron

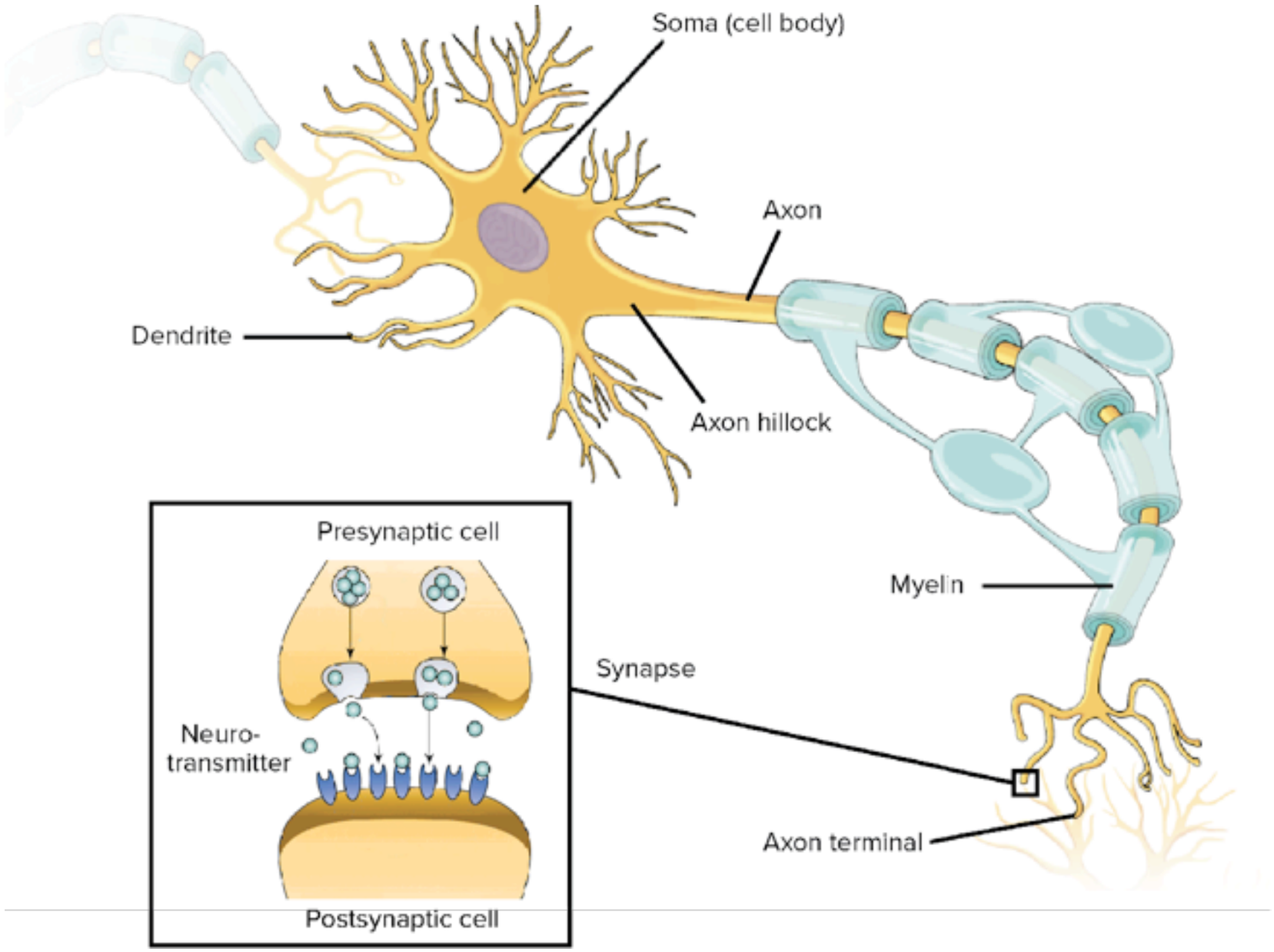
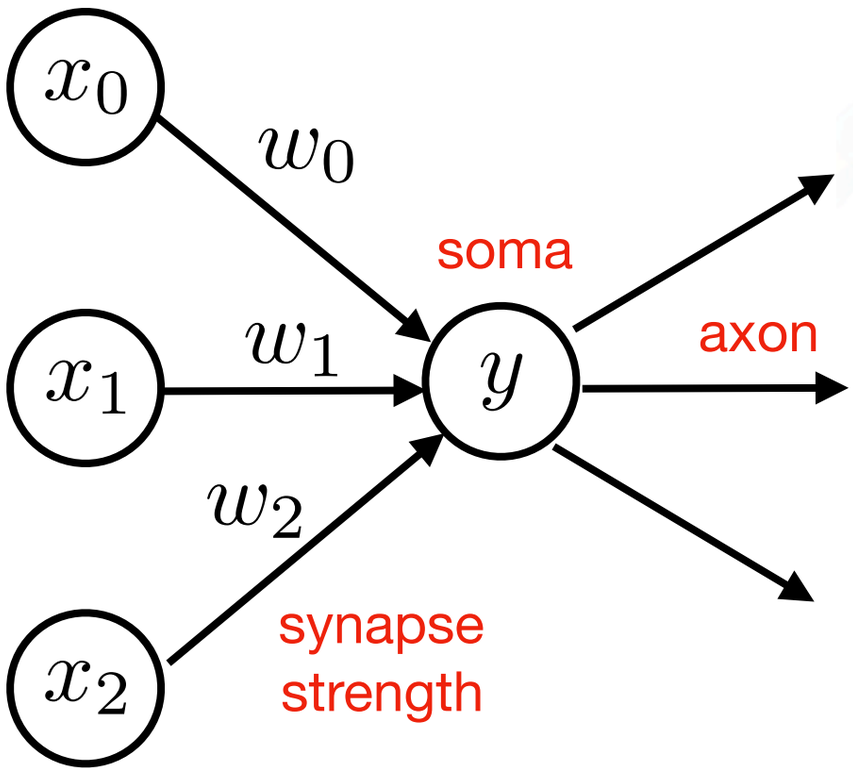
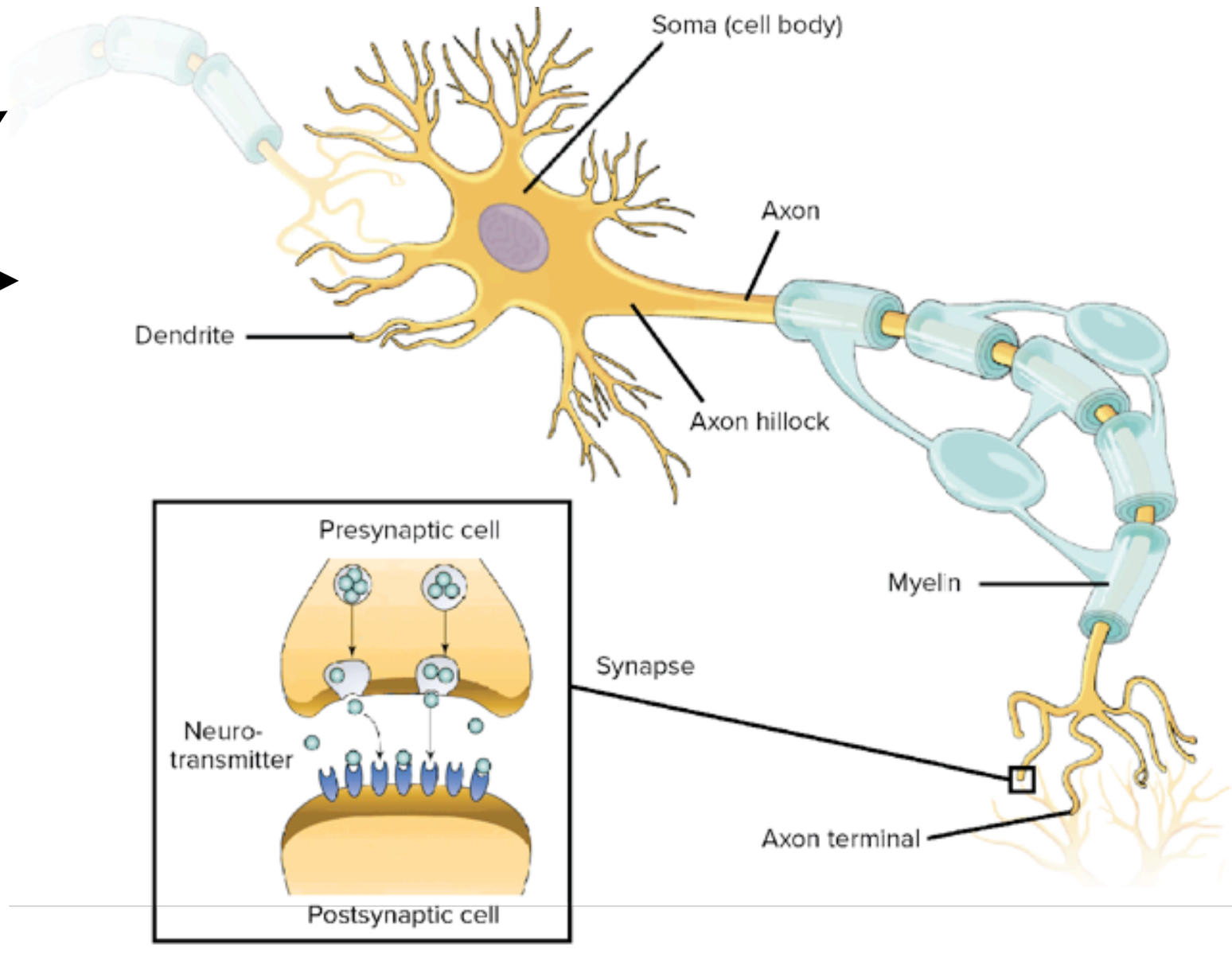


Image from OpenStax Biology

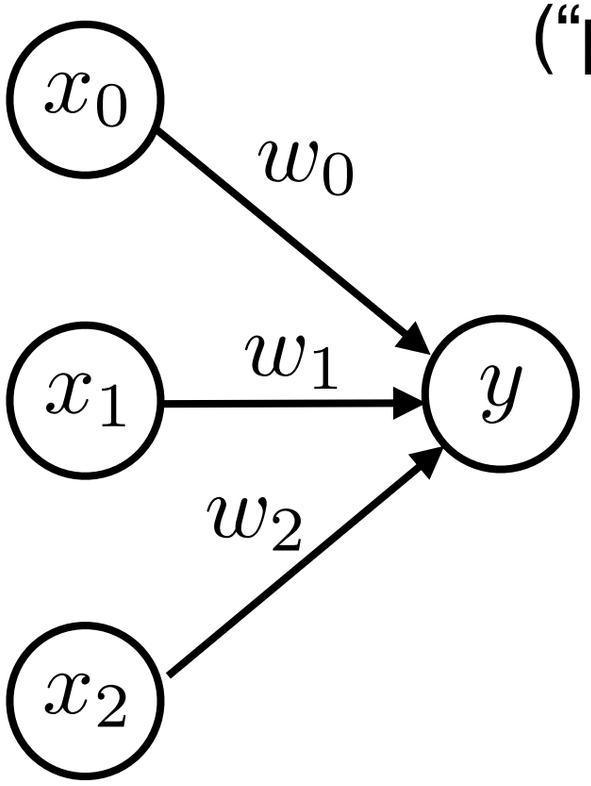
A simple artificial neuron





A simple artificial neuron (or “unit”)

(“perceptron”; Rosenblatt, 1958)



$$y = g(\sum_i x_i w_i + b)$$

x : input vector

w : weight vector

b : bias

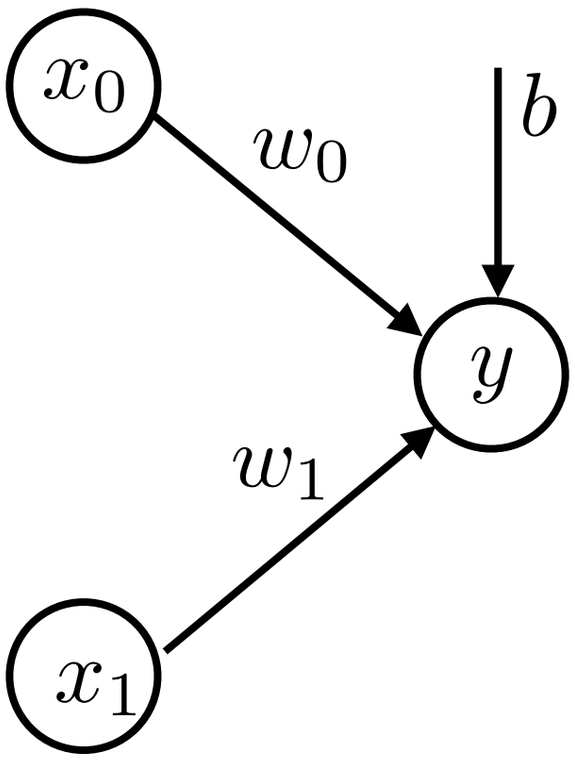
g : activation function

y : output

activation function:

$$g(\mathbf{net}) = \begin{cases} 1 & \mathbf{net} \geq 0 \\ 0 & \mathbf{net} < 0 \end{cases}$$

Computing the logical OR function



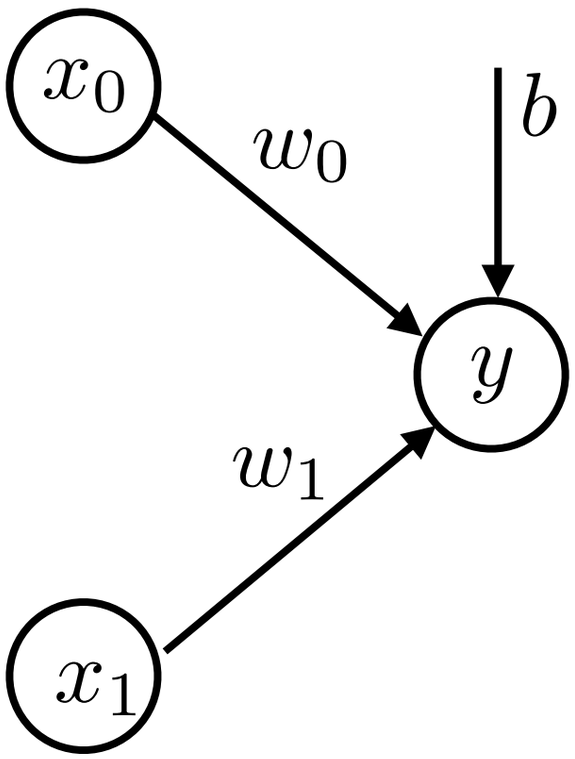
$$y = g(\sum_i x_i w_i + b)$$

$$g(\mathbf{net}) = \begin{cases} 1 & \mathbf{net} \geq 0 \\ 0 & \mathbf{net} < 0 \end{cases}$$

logical OR function

x_0	x_1	y
0	0	0
0	1	1
1	0	1
1	1	1

Computing the logical OR function

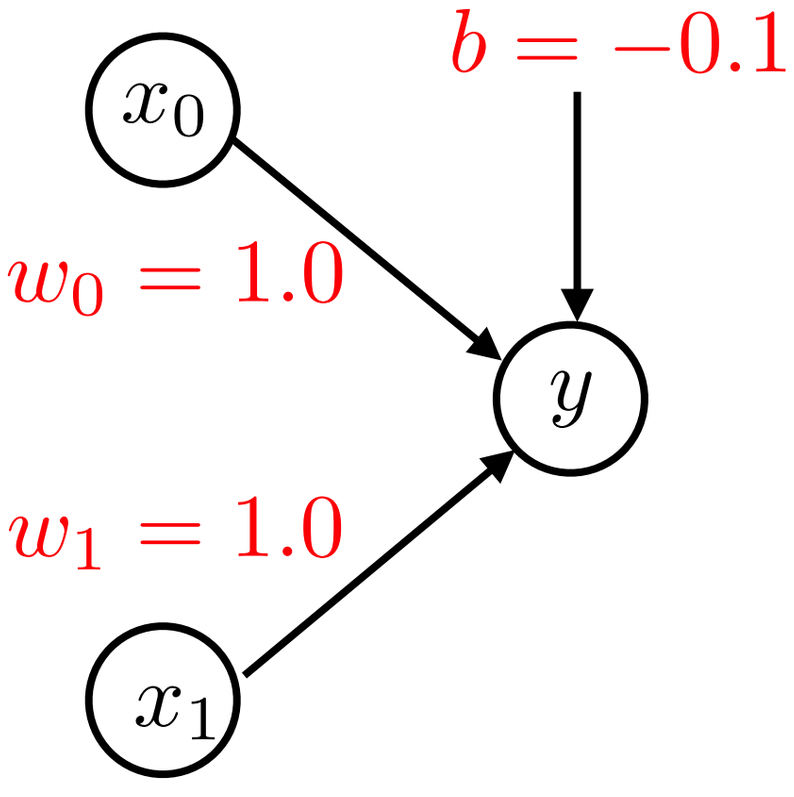


$$y = g(\sum_i x_i w_i + b)$$

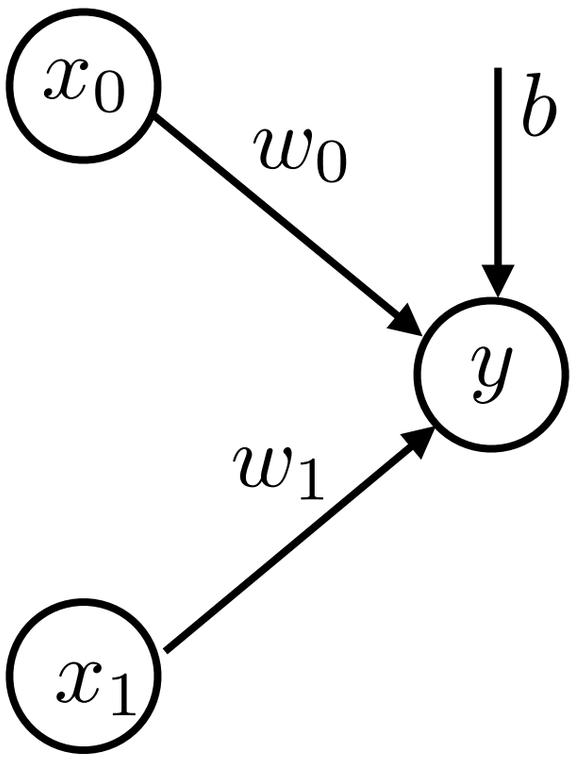
$$g(\mathbf{net}) = \begin{cases} 1 & \mathbf{net} \geq 0 \\ 0 & \mathbf{net} < 0 \end{cases}$$

logical OR function

x_0	x_1	y
0	0	0
0	1	1
1	0	1
1	1	1



Computing the logical OR function

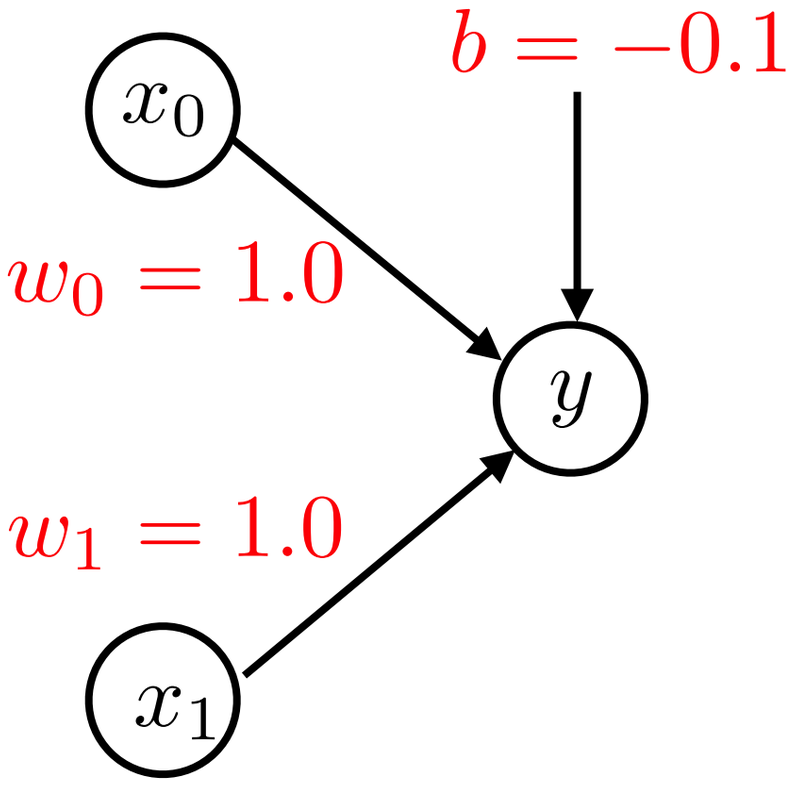


$$y = g(\sum_i x_i w_i + b)$$

$$g(\mathbf{net}) = \begin{cases} 1 & \mathbf{net} \geq 0 \\ 0 & \mathbf{net} < 0 \end{cases}$$

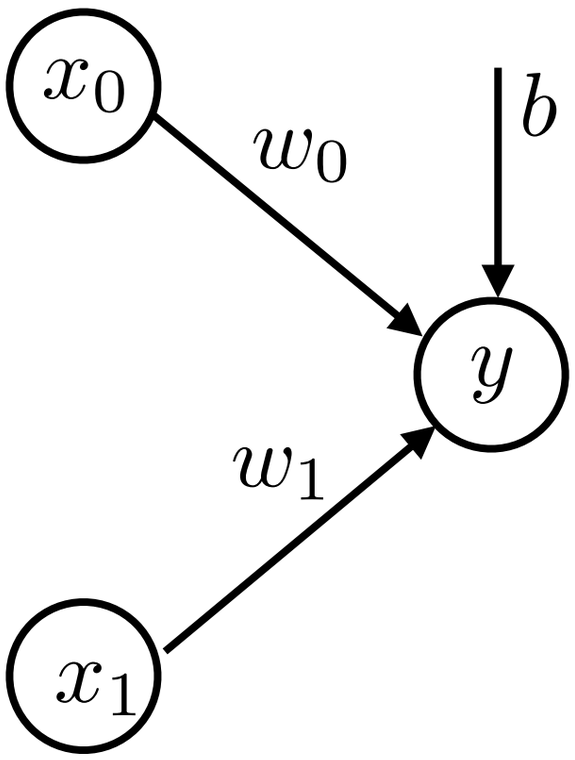
logical OR function

x_0	x_1	y
0	0	0
0	1	1
1	0	1
1	1	1



x_0w_0	x_1w_1	b	net	y
0×1	0×1	-0.1	-0.1	0

Computing the logical OR function

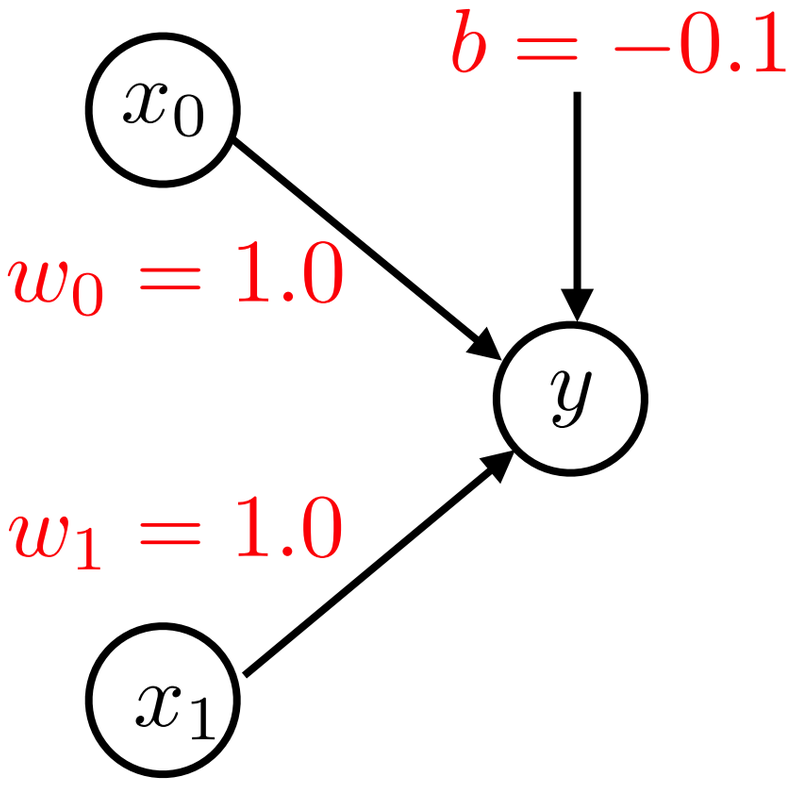


$$y = g(\sum_i x_i w_i + b)$$

$$g(\mathbf{net}) = \begin{cases} 1 & \mathbf{net} \geq 0 \\ 0 & \mathbf{net} < 0 \end{cases}$$

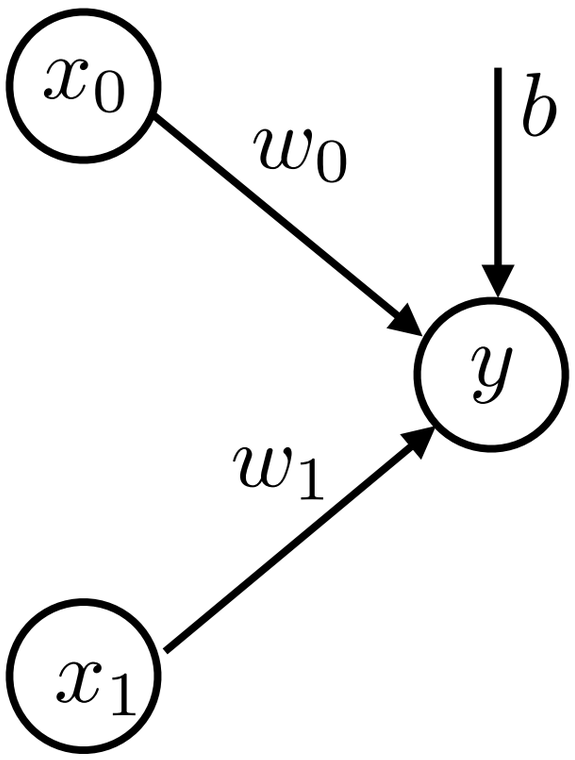
logical OR function

x_0	x_1	y
0	0	0
0	1	1
1	0	1
1	1	1



x_0w_0	x_1w_1	b	net	y
0×1	0×1	-0.1	-0.1	0
0×1	1×1	-0.1	0.9	1

Computing the logical OR function

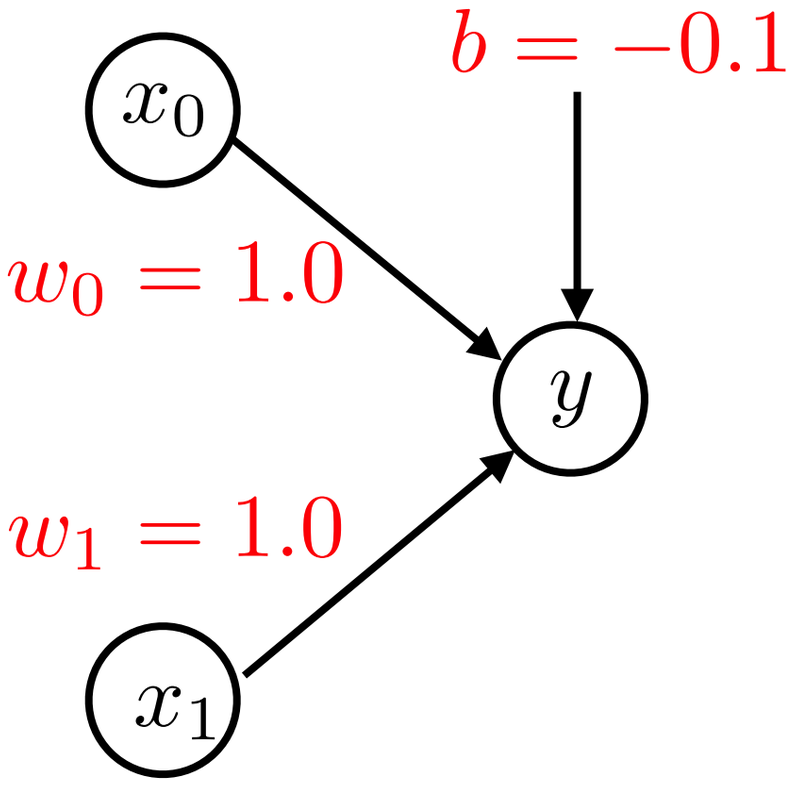


$$y = g(\sum_i x_i w_i + b)$$

$$g(\mathbf{net}) = \begin{cases} 1 & \mathbf{net} \geq 0 \\ 0 & \mathbf{net} < 0 \end{cases}$$

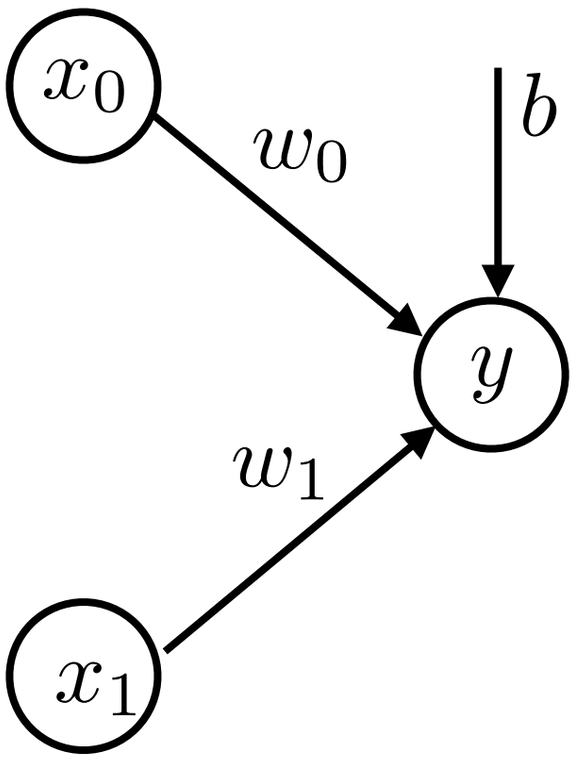
logical OR function

x_0	x_1	y
0	0	0
0	1	1
1	0	1
1	1	1



x_0w_0	x_1w_1	b	net	y
0×1	0×1	-0.1	-0.1	0
0×1	1×1	-0.1	0.9	1
1×1	0×1	-0.1	0.9	1

Computing the logical OR function

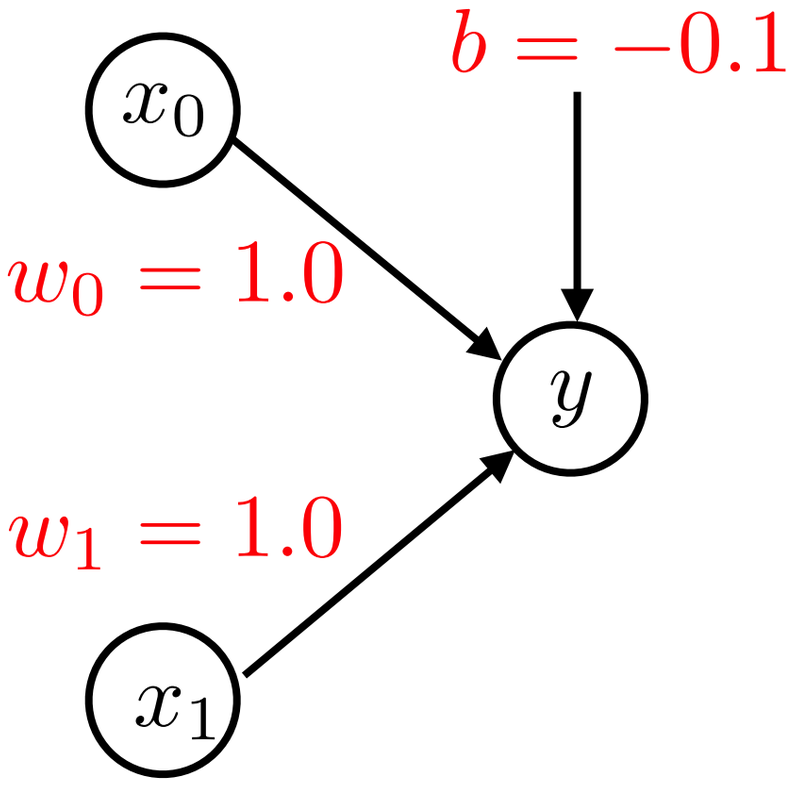


$$y = g(\sum_i x_i w_i + b)$$

$$g(\mathbf{net}) = \begin{cases} 1 & \mathbf{net} \geq 0 \\ 0 & \mathbf{net} < 0 \end{cases}$$

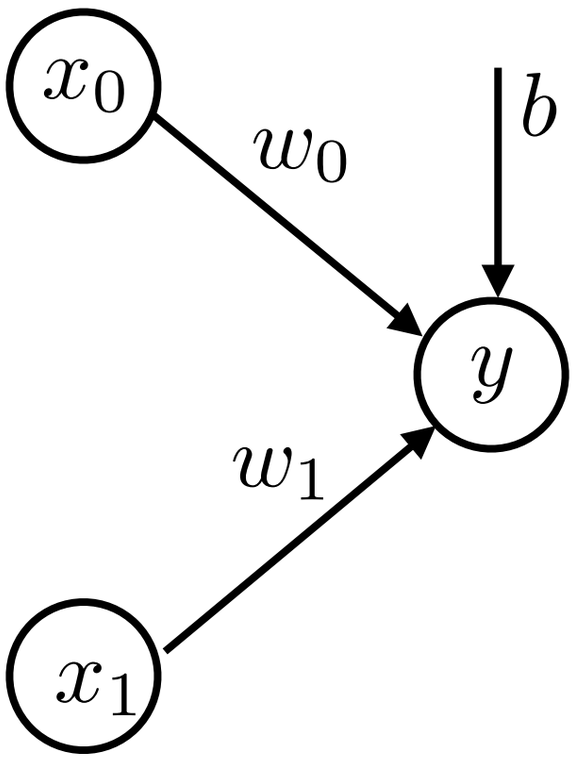
logical OR function

x_0	x_1	y
0	0	0
0	1	1
1	0	1
1	1	1



x_0w_0	x_1w_1	b	net	y
0×1	0×1	-0.1	-0.1	0
0×1	1×1	-0.1	0.9	1
1×1	0×1	-0.1	0.9	1
1×1	1×1	-0.1	1.9	1

Computing the logical AND function



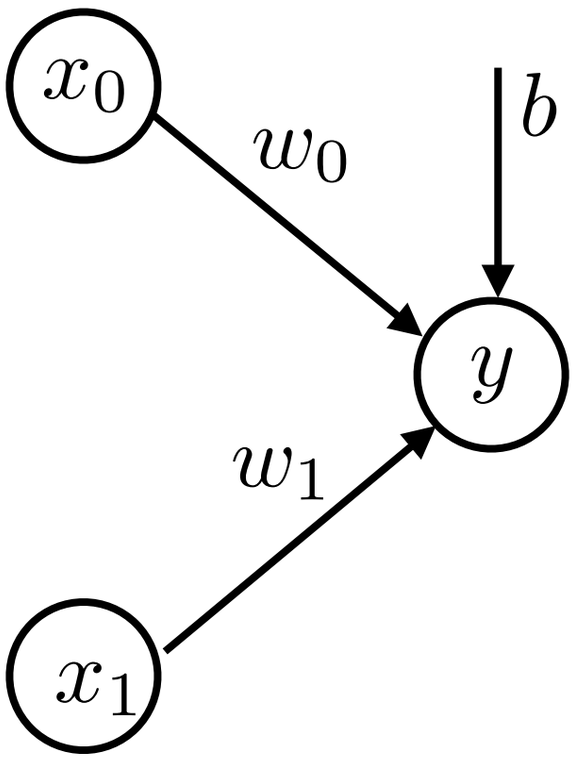
$$y = g(\sum_i x_i w_i + b)$$

$$g(\mathbf{net}) = \begin{cases} 1 & \mathbf{net} \geq 0 \\ 0 & \mathbf{net} < 0 \end{cases}$$

logical AND function

x_0	x_1	y
0	0	0
0	1	0
1	0	0
1	1	1

Computing the logical AND function

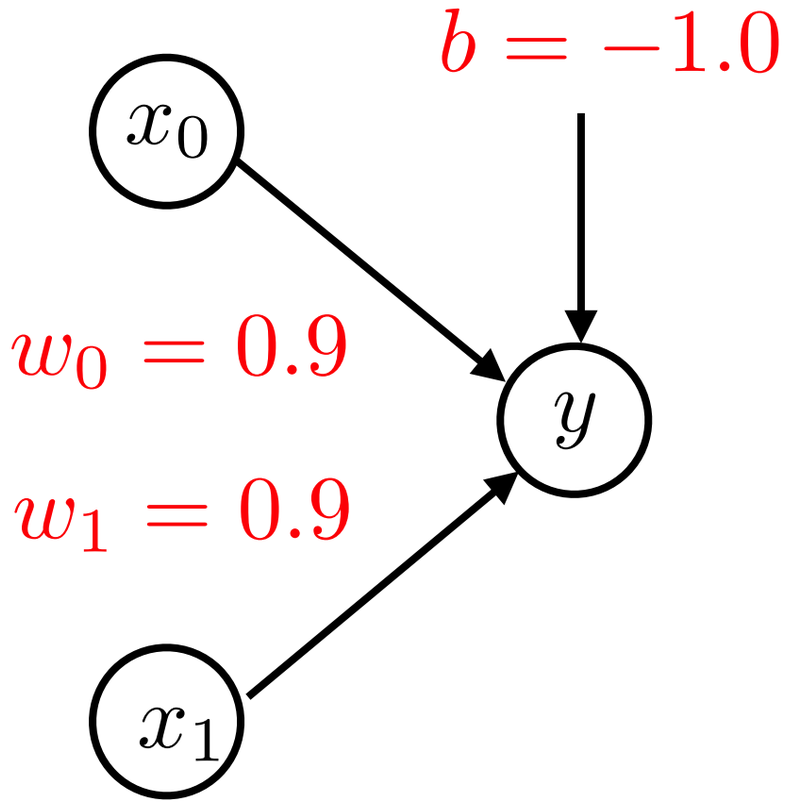


$$y = g(\sum_i x_i w_i + b)$$

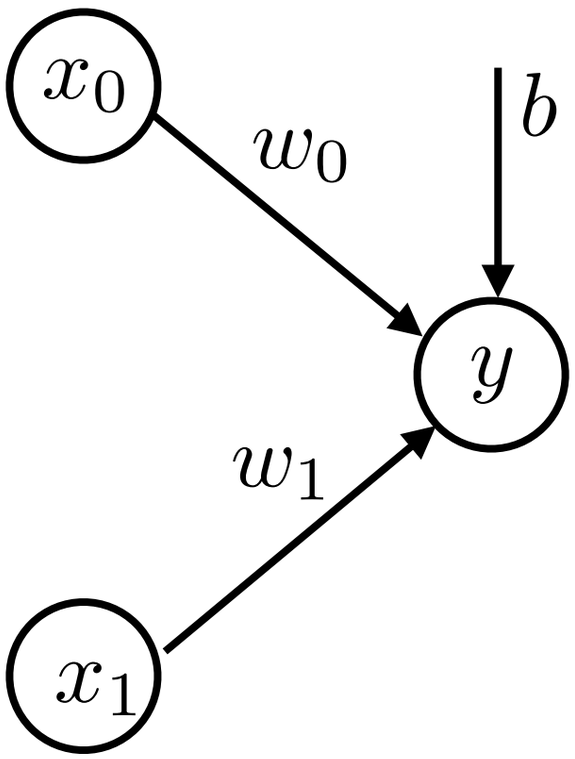
$$g(\mathbf{net}) = \begin{cases} 1 & \mathbf{net} \geq 0 \\ 0 & \mathbf{net} < 0 \end{cases}$$

logical AND function

x_0	x_1	y
0	0	0
0	1	0
1	0	0
1	1	1



Computing the logical AND function

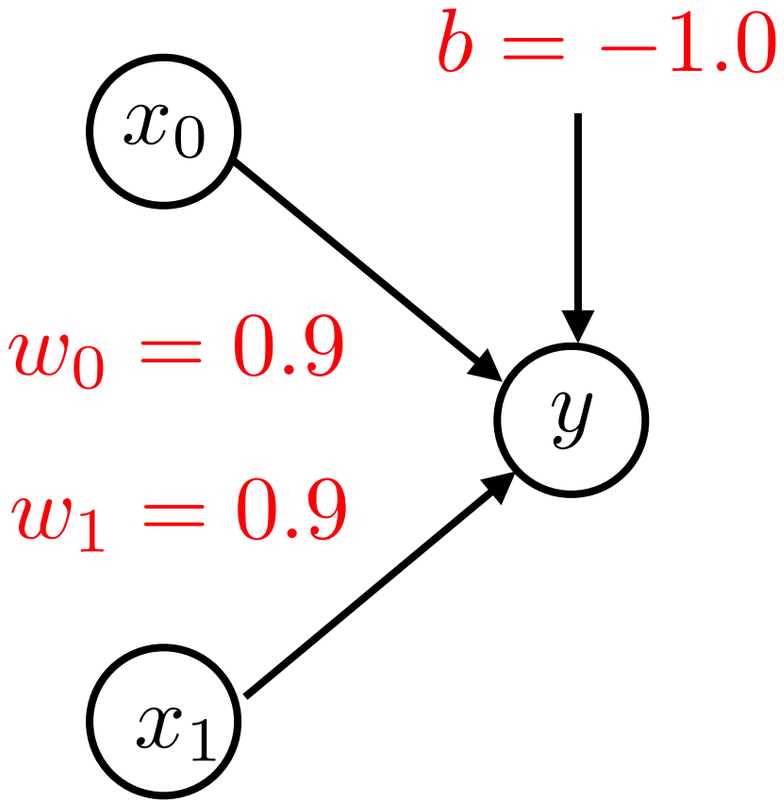


$$y = g(\sum_i x_i w_i + b)$$

$$g(\mathbf{net}) = \begin{cases} 1 & \mathbf{net} \geq 0 \\ 0 & \mathbf{net} < 0 \end{cases}$$

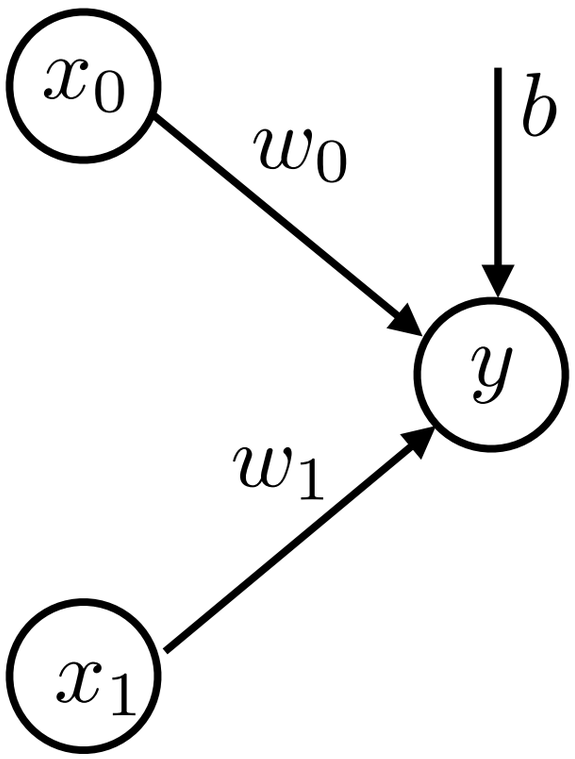
logical AND function

x_0	x_1	y
0	0	0
0	1	0
1	0	0
1	1	1



x_0w_0	x_1w_1	b	net	y
0×0.9	0×0.9	-1.0	-1.0	0

Computing the logical AND function

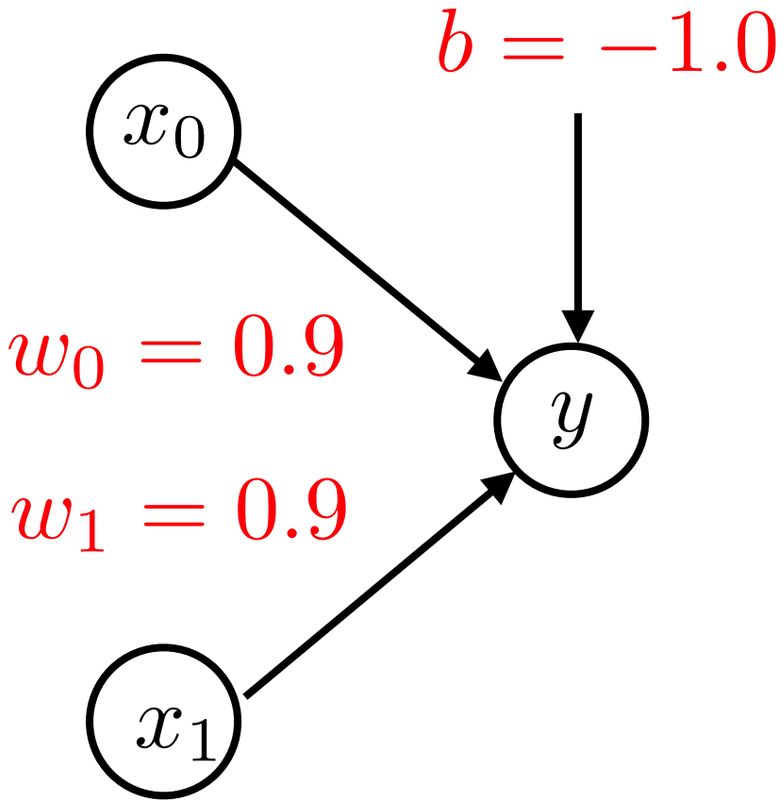


$$y = g(\sum_i x_i w_i + b)$$

$$g(\mathbf{net}) = \begin{cases} 1 & \mathbf{net} \geq 0 \\ 0 & \mathbf{net} < 0 \end{cases}$$

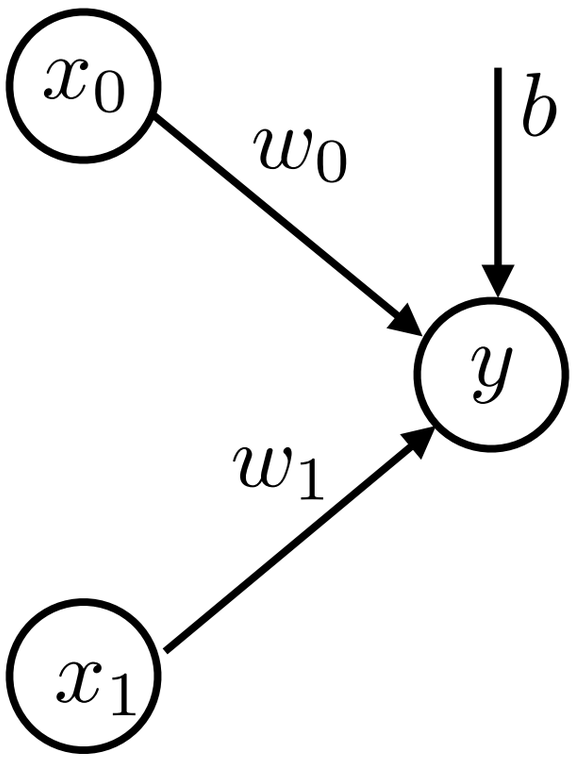
logical AND function

x_0	x_1	y
0	0	0
0	1	0
1	0	0
1	1	1



x_0w_0	x_1w_1	b	net	y
0×0.9	0×0.9	-1.0	-1.0	0
0×0.9	1×0.9	-1.0	-0.1	0

Computing the logical AND function

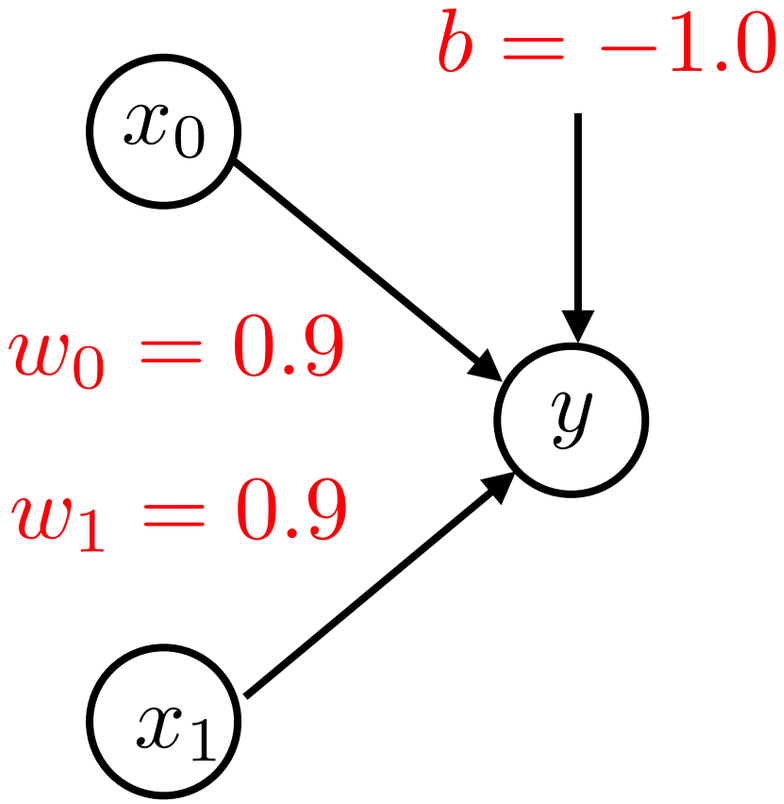


$$y = g(\sum_i x_i w_i + b)$$

$$g(\mathbf{net}) = \begin{cases} 1 & \mathbf{net} \geq 0 \\ 0 & \mathbf{net} < 0 \end{cases}$$

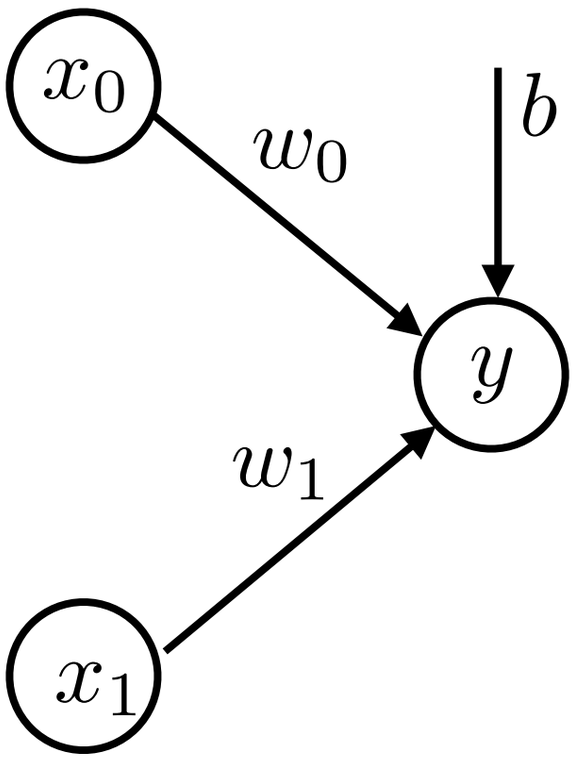
logical AND function

x_0	x_1	y
0	0	0
0	1	0
1	0	0
1	1	1



x_0w_0	x_1w_1	b	net	y
0×0.9	0×0.9	-1.0	-1.0	0
0×0.9	1×0.9	-1.0	-0.1	0
1×0.9	0×0.9	-1.0	-0.1	0

Computing the logical AND function

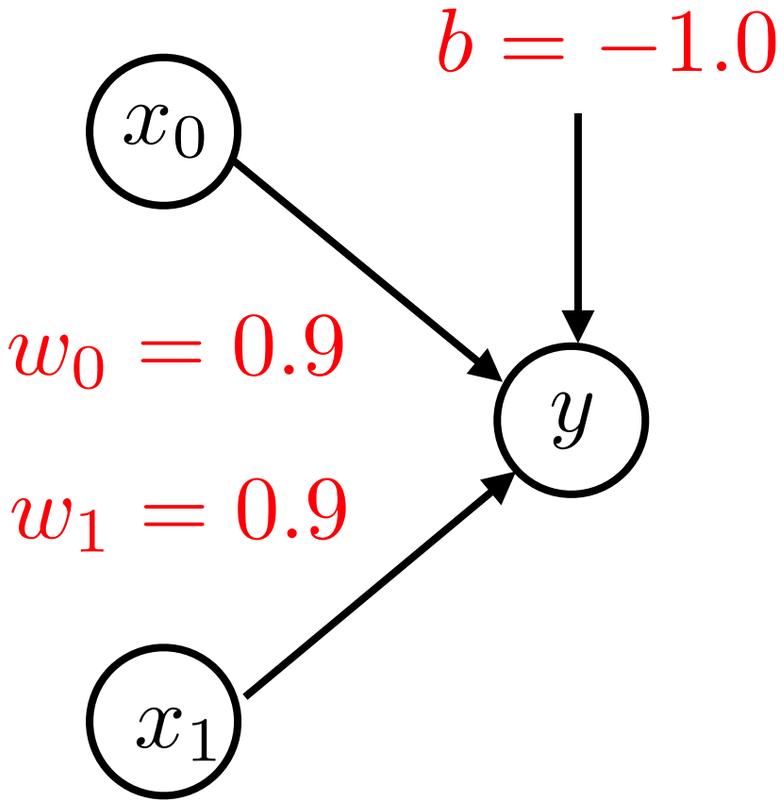


$$y = g(\sum_i x_i w_i + b)$$

$$g(\mathbf{net}) = \begin{cases} 1 & \mathbf{net} \geq 0 \\ 0 & \mathbf{net} < 0 \end{cases}$$

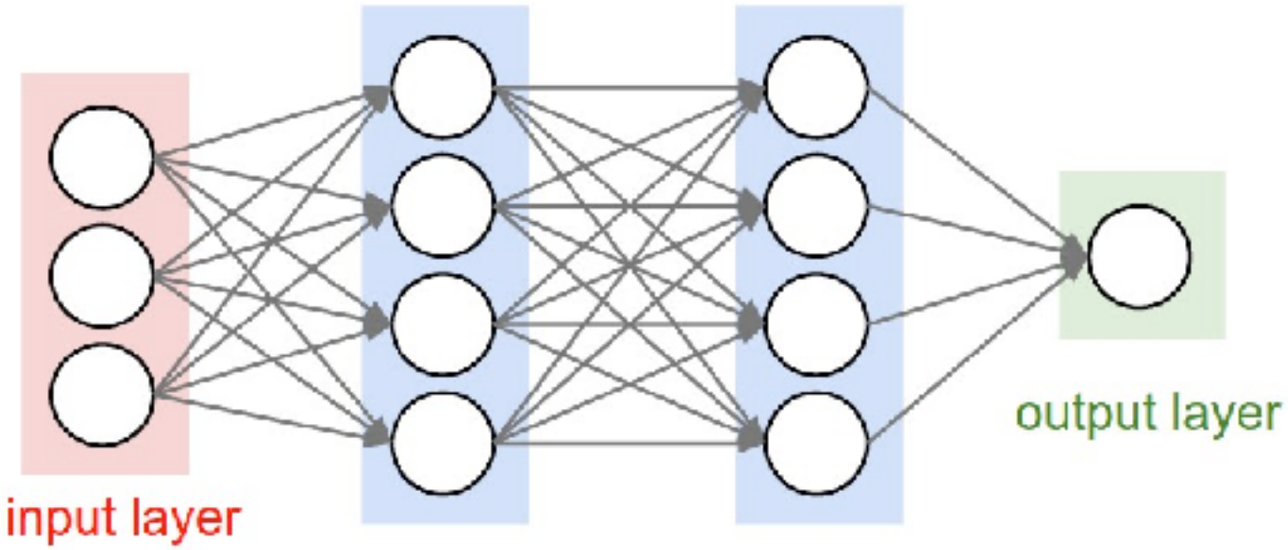
logical AND function

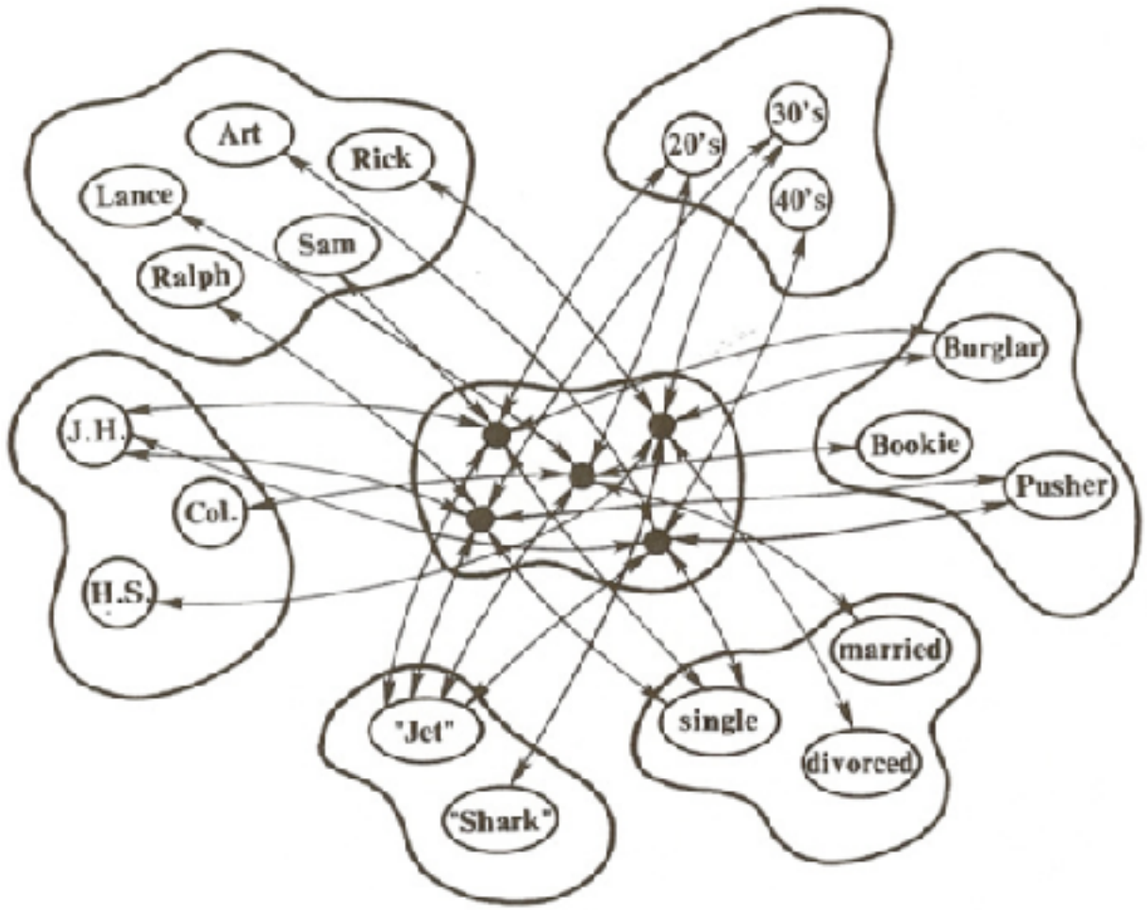
x_0	x_1	y
0	0	0
0	1	0
1	0	0
1	1	1

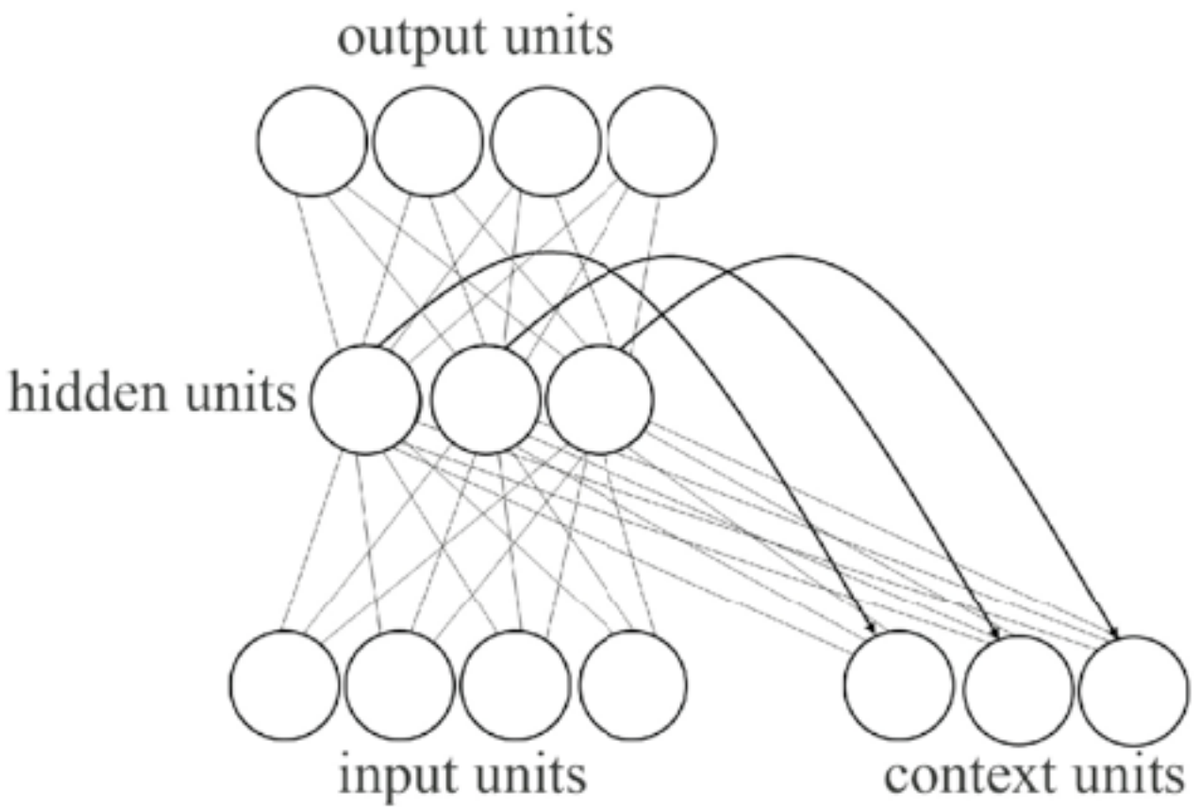


x_0w_0	x_1w_1	b	net	y
0×0.9	0×0.9	-1.0	-1.0	0
0×0.9	1×0.9	-1.0	-0.1	0
1×0.9	0×0.9	-1.0	-0.1	0
1×0.9	1×0.9	-1.0	0.8	1

Artificial neural networks







Example: Retrieving information from memory

A key property of memory is that it's content addressable:

- "I met one of your friends the other day. He has short black hair, glasses, he goes to school with you…"

Memory could be structured like a database table…

	Gender	Hair color	Hair length	Glasses	School
Susan	F	black	long	N	Princeton
Sally	F	blonde	long	Y	NYU
Pablo	M	brown	short	N	Columbia
Bo	M	black	short	Y	NYU

. . .

. . .

. . .

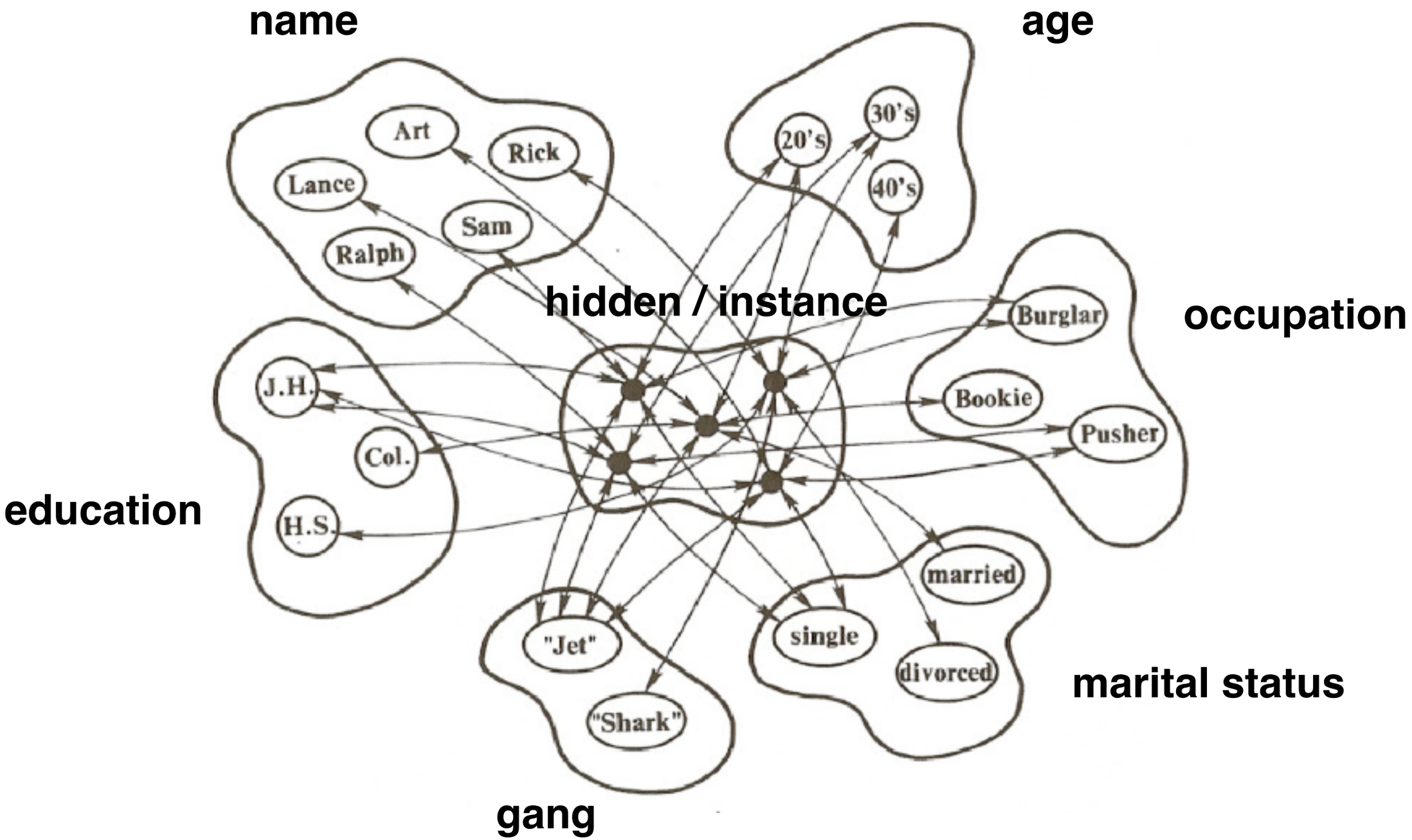
What are some weaknesses of structuring memory this way for content addressable retrieval?

- You might have to search through each row in sequence
- You could use an indexing scheme, but it may not recover well from errors and noise

Interactive activation model

(McClelland, 1981)

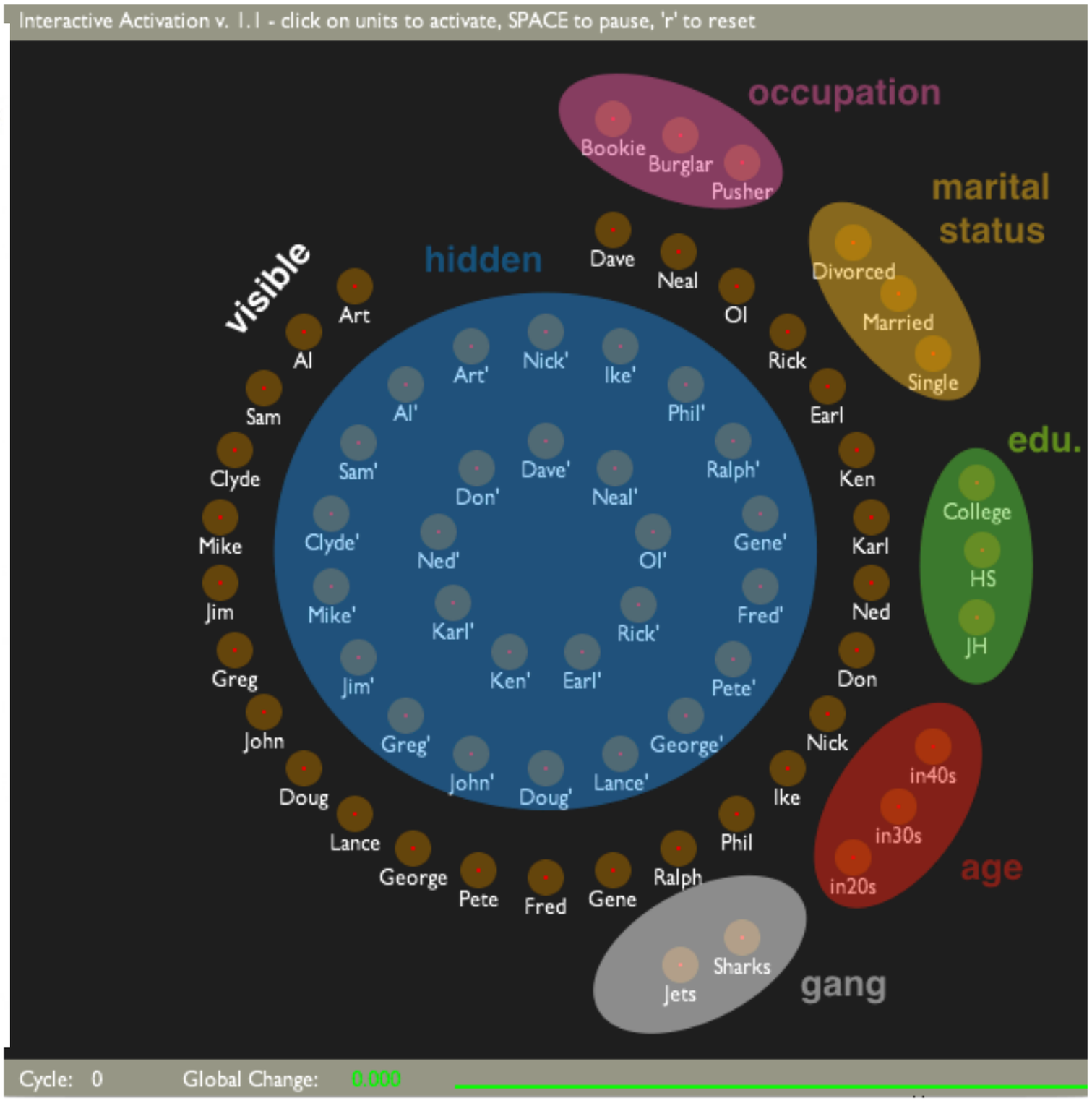
- Each item is a unit with mutually excitatory connections to its properties
- Properties are organized into pools of mutually inhibitory units (e.g., since a person can't be both in their 20's and in their 30's)



Interactive activation model - Jets and Sharks (software for Homework 1)

The Jets and The Sharks

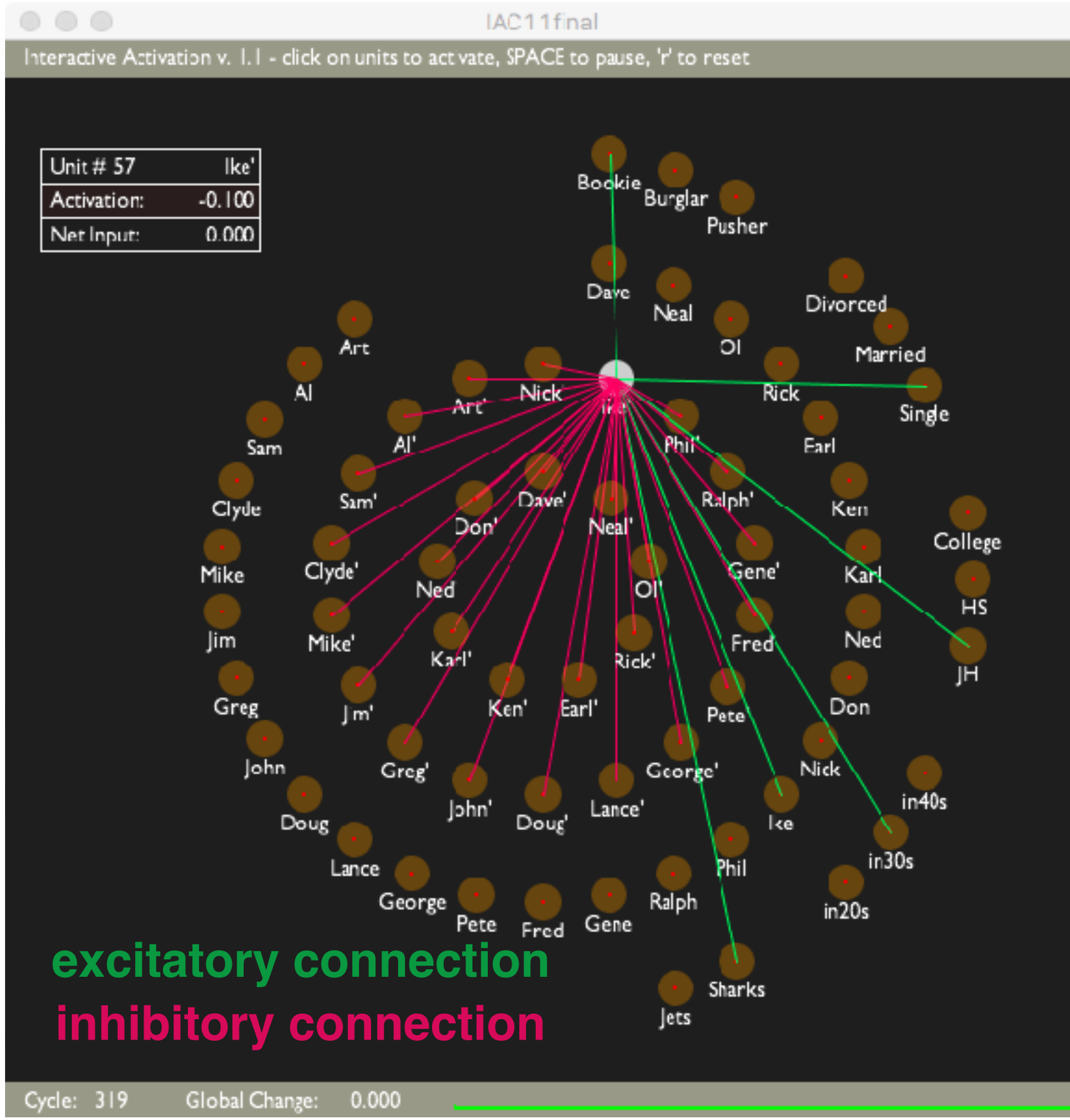
Name	Gang	Age	Edu	Mar	Occupation
Art	Jets	40's	J.H.	Sing.	Pusher
Al	Jets	30's	J.H.	Mar.	Burglar
Sam	Jets	20's	COL.	Sing.	Bookie
Clyde	Jets	40's	J.H.	Sing.	Bookie
Mike	Jets	30's	J.H.	Sing.	Bookie
Jim	Jets	20's	J.H.	Div.	Burglar
Greg	Jets	20's	H.S.	Mar.	Pusher
John	Jets	20's	J.H.	Mar.	Burglar
Doug	Jets	30's	H.S.	Sing.	Bookie
Lance	Jets	20's	J.H.	Mar.	Burglar
George	Jets	20's	J.H.	Div.	Burglar
Pete	Jets	20's	H.S.	Sing.	Bookie
Fred	Jets	20's	H.S.	Sing.	Pusher
Gene	Jets	20's	COL.	Sing.	Pusher
Ralph	Jets	30's	J.H.	Sing.	Pusher
Phil	Sharks	30's	COL.	Mar.	Pusher
Ike	Sharks	30's	J.H.	Sing.	Bookie
Nick	Sharks	30's	H.S.	Sing.	Pusher
Don	Sharks	30's	COL.	Mar.	Burglar
Ned	Sharks	30's	COL.	Mar.	Bookie
Karl	Sharks	40's	H.S.	Mar.	Bookie
Ken	Sharks	20's	H.S.	Sing.	Burglar
Earl	Sharks	40's	H.S.	Mar.	Burglar
Rick	Sharks	30's	H.S.	Div.	Burglar
Ol	Sharks	30's	COL.	Mar.	Pusher
Neal	Sharks	30's	H.S.	Sing.	Bookie
Dave	Sharks	30's	H.S.	Div.	Pusher



Network structure: connecting an instance unit with its properties

The Jets and The Sharks

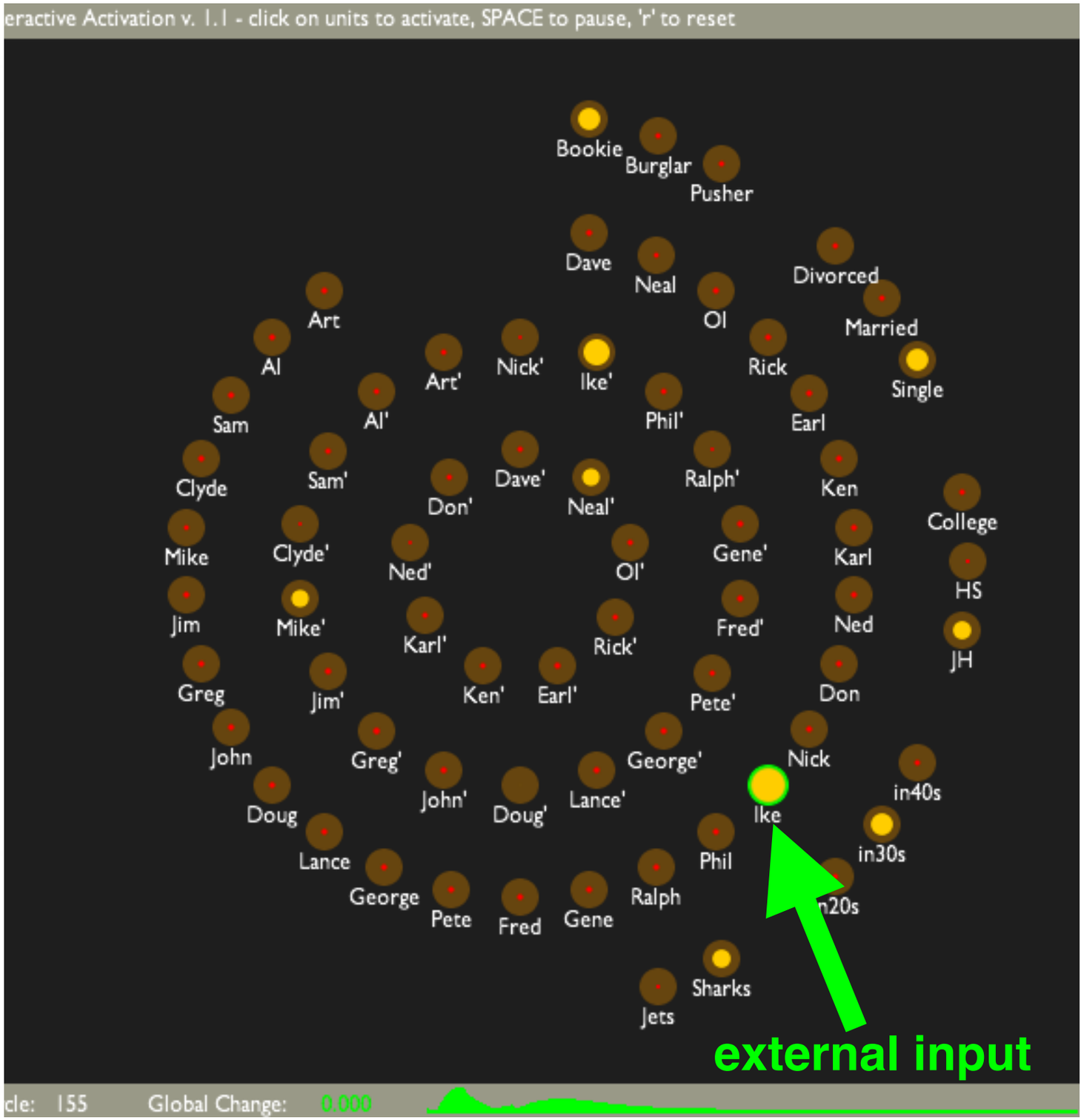
Name	Gang	Age	Edu	Mar	Occupation
Art	Jets	40's	J.H.	Sing.	Pusher
Al	Jets	30's	J.H.	Mar.	Burglar
Sam	Jets	20's	COL.	Sing.	Bookie
Clyde	Jets	40's	J.H.	Sing.	Bookie
Mike	Jets	30's	J.H.	Sing.	Bookie
Jim	Jets	20's	J.H.	Div.	Burglar
Greg	Jets	20's	H.S.	Mar.	Pusher
John	Jets	20's	J.H.	Mar.	Burglar
Doug	Jets	30's	H.S.	Sing.	Bookie
Lance	Jets	20's	J.H.	Mar.	Burglar
George	Jets	20's	J.H.	Div.	Burglar
Pete	Jets	20's	H.S.	Sing.	Bookie
Fred	Jets	20's	H.S.	Sing.	Pusher
Gene	Jets	20's	COL.	Sing.	Pusher
Ralph	Jets	30's	J.H.	Sing.	Pusher
Phil	Sharks	30's	COL.	Mar.	Pusher
Ike	Sharks	30's	J.H.	Sing.	Bookie
Nick	Sharks	30's	H.S.	Sing.	Pusher
Don	Sharks	30's	COL.	Mar.	Burglar
Ned	Sharks	30's	COL.	Mar.	Bookie
Karl	Sharks	40's	H.S.	Mar.	Bookie
Ken	Sharks	20's	H.S.	Sing.	Burglar
Earl	Sharks	40's	H.S.	Mar.	Burglar
Rick	Sharks	30's	H.S.	Div.	Burglar
Ol	Sharks	30's	COL.	Mar.	Pusher
Neal	Sharks	30's	H.S.	Sing.	Bookie
Dave	Sharks	30's	H.S.	Div.	Pusher



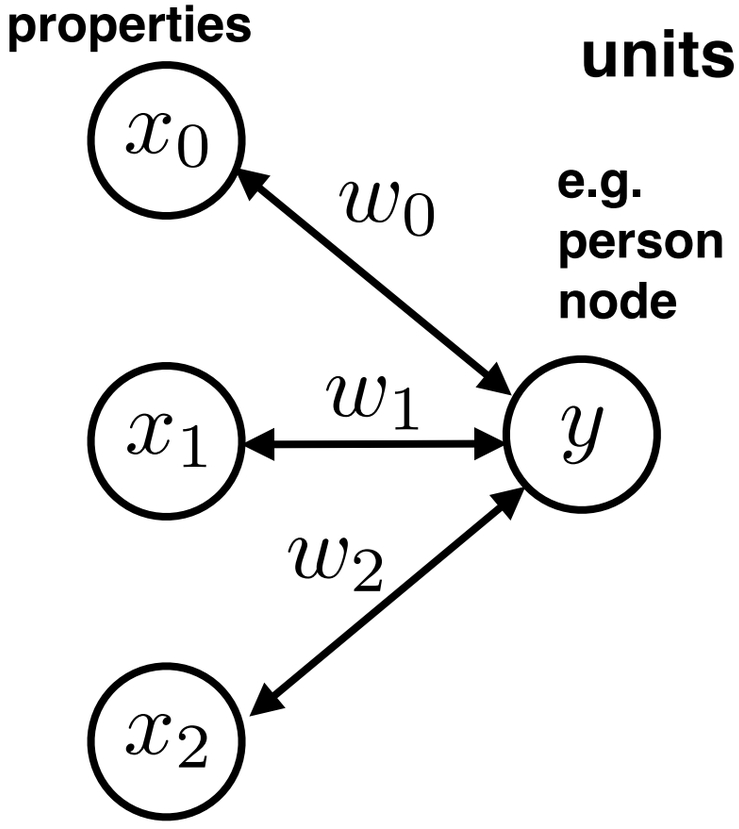
Retrieving Ike's properties from his name

The Jets and The Sharks

Name	Gang	Age	Edu	Mar	Occupation
Art	Jets	40's	J.H.	Sing.	Pusher
Al	Jets	30's	J.H.	Mar.	Burglar
Sam	Jets	20's	COL.	Sing.	Bookie
Clyde	Jets	40's	J.H.	Sing.	Bookie
Mike	Jets	30's	J.H.	Sing.	Bookie
Jim	Jets	20's	J.H.	Div.	Burglar
Greg	Jets	20's	H.S.	Mar.	Pusher
John	Jets	20's	J.H.	Mar.	Burglar
Doug	Jets	30's	H.S.	Sing.	Bookie
Lance	Jets	20's	J.H.	Mar.	Burglar
George	Jets	20's	J.H.	Div.	Burglar
Pete	Jets	20's	H.S.	Sing.	Bookie
Fred	Jets	20's	H.S.	Sing.	Pusher
Gene	Jets	20's	COL.	Sing.	Pusher
Ralph	Jets	30's	J.H.	Sing.	Pusher
Phil	Sharks	30's	COL.	Mar.	Pusher
Ike	Sharks	30's	J.H.	Sing.	Bookie
Nick	Sharks	30's	H.S.	Sing.	Pusher
Don	Sharks	30's	COL.	Mar.	Burglar
Ned	Sharks	30's	COL.	Mar.	Bookie
Karl	Sharks	40's	H.S.	Mar.	Bookie
Ken	Sharks	20's	H.S.	Sing.	Burglar
Earl	Sharks	40's	H.S.	Mar.	Burglar
Rick	Sharks	30's	H.S.	Div.	Burglar
Ol	Sharks	30's	COL.	Mar.	Pusher
Neal	Sharks	30's	H.S.	Sing.	Bookie
Dave	Sharks	30's	H.S.	Div.	Pusher



Interactive activation model - details



units take continuous (rather than binary) values

This is a recurrent neural network:

- activity propagates forward to y,

$$y^{(t+1)} = g(y^{(t)}, \sum_i x_i^{(t)} w_i + b)$$

- at the same time, activity propagates backward to x_i

$$x_i^{(t+1)} = g(x_i^{(t)}, y^{(t)} w_i + b)$$

This activation function is more complex and depends on the current activation $u^{(t)}$:

$$g(u^{(t)}, \mathbf{net}) = \begin{cases} u^{(t)} + (max - u^{(t)})\mathbf{net} - decay(u^{(t)} - rest) & \mathbf{net} > 0 \\ u^{(t)} + (u^{(t)} - min)\mathbf{net} - decay(u^{(t)} - rest) & \mathbf{net} \leq 0 \end{cases}$$

max : max activation for a unit

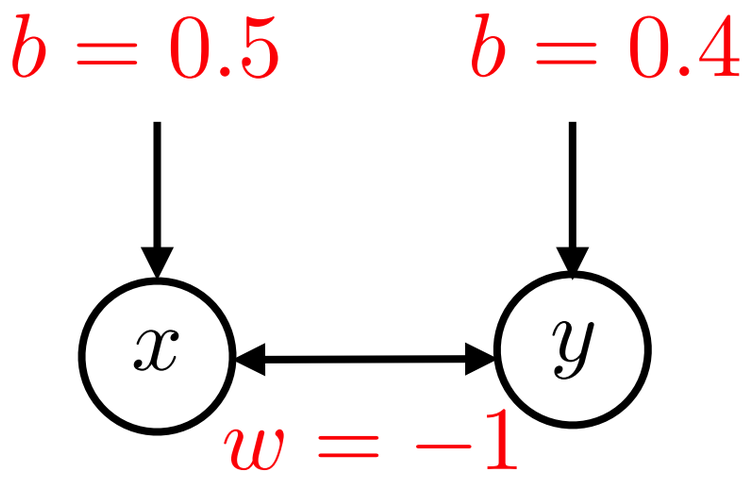
min : minimum activation for a unit

$rest$: resting activation

$decay$: how fast we decay to rest

“Rich get richer” dynamics

If two units are competing (mutual inhibition), the unit with stronger net input will tend to dominate over time.



$$y^{(t+1)} = g(y^{(t)}, \sum_i x_i^{(t)} w_i + b)$$

$$g(u^{(t)}, \mathbf{net}) = \begin{cases} u^{(t)} + (max - u^{(t)})\mathbf{net} - decay(u^{(t)} - rest) & \mathbf{net} > 0 \\ u^{(t)} + (u^{(t)} - min)\mathbf{net} - decay(u^{(t)} - rest) & \mathbf{net} \leq 0 \end{cases}$$

$$\begin{aligned} max &= 1 \\ min &= -1 \\ rest &= -0.1 \\ decay &= 0.1 \end{aligned}$$

unit activity over time

step 01 : x 0.56 y 0.45
step 02 : x 0.52 y 0.16
step 03 : x 0.62 y 0.00
step 04 : x 0.74 y -0.23
step 05 : x 0.84 y -0.47
step 06 : x 0.90 y -0.67
step 07 : x 0.92 y -0.78
step 08 : x 0.92 y -0.83
step 09 : x 0.92 y -0.84
step 10 : x 0.92 y -0.85
step 11 : x 0.92 y -0.85
step 12 : x 0.92 y -0.86

reaches stable state

⋮

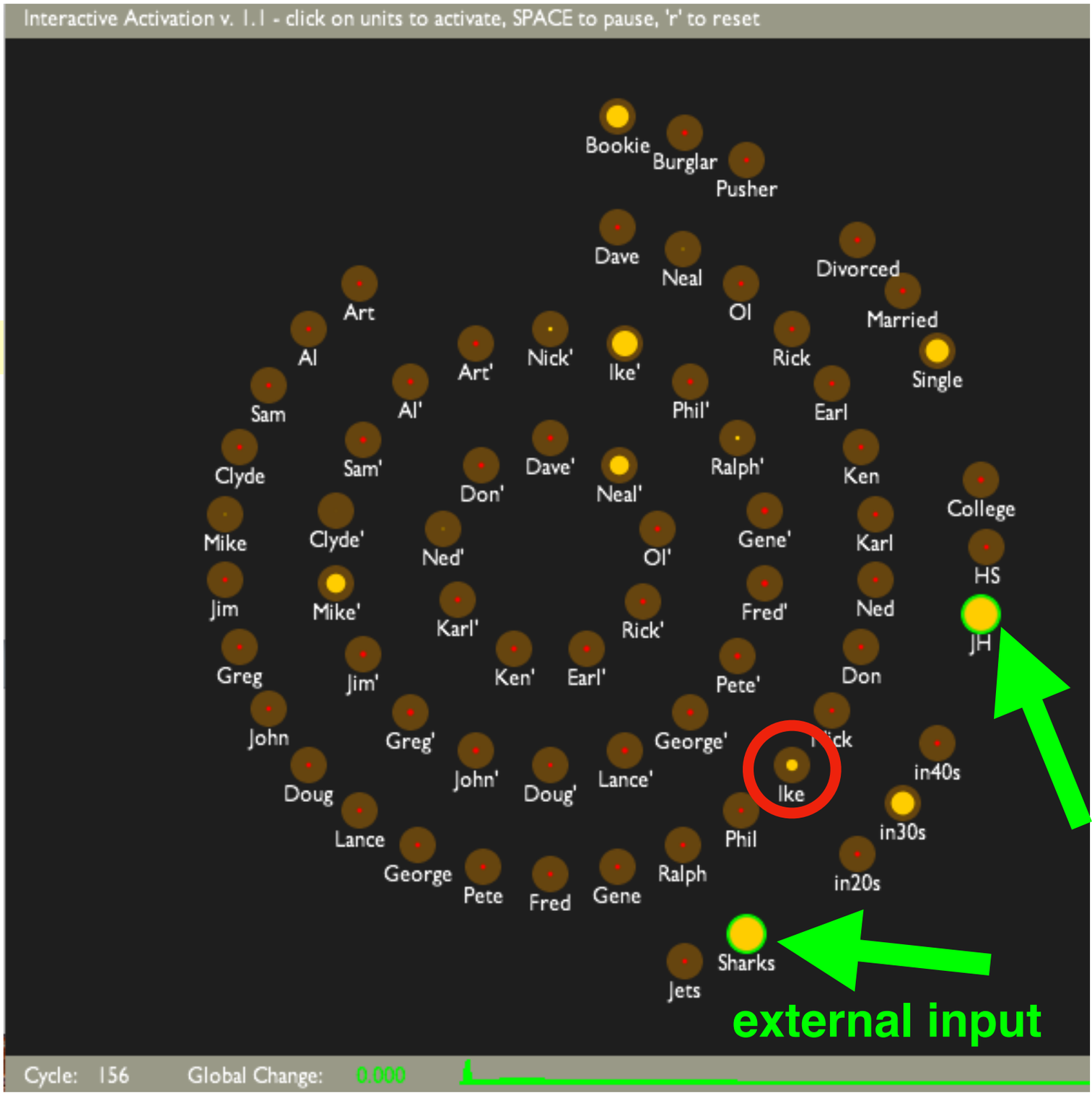
step 100 : x 0.92 y -0.86

Unit x wins!

Content addressability: Retrieving Ike's name from only partial information

"Who do you know who is a Shark with a junior high education?"

Phil	Sharks	30's	COL.	Mar.	Pusher
Ike	Sharks	30's	J.H.	Sing.	Bookie
Nick	Sharks	30's	H.S.	Sing.	Pusher
Don	Sharks	30's	COL.	Mar.	Burglar
Ned	Sharks	30's	COL.	Mar.	Bookie
Karl	Sharks	40's	H.S.	Mar.	Bookie
Ken	Sharks	20's	H.S.	Sing.	Burglar
Earl	Sharks	40's	H.S.	Mar.	Burglar
Rick	Sharks	30's	H.S.	Div.	Burglar
Ol	Sharks	30's	COL.	Mar.	Pusher
Neal	Sharks	30's	H.S.	Sing.	Bookie
Dave	Sharks	30's	H.S.	Div.	Pusher



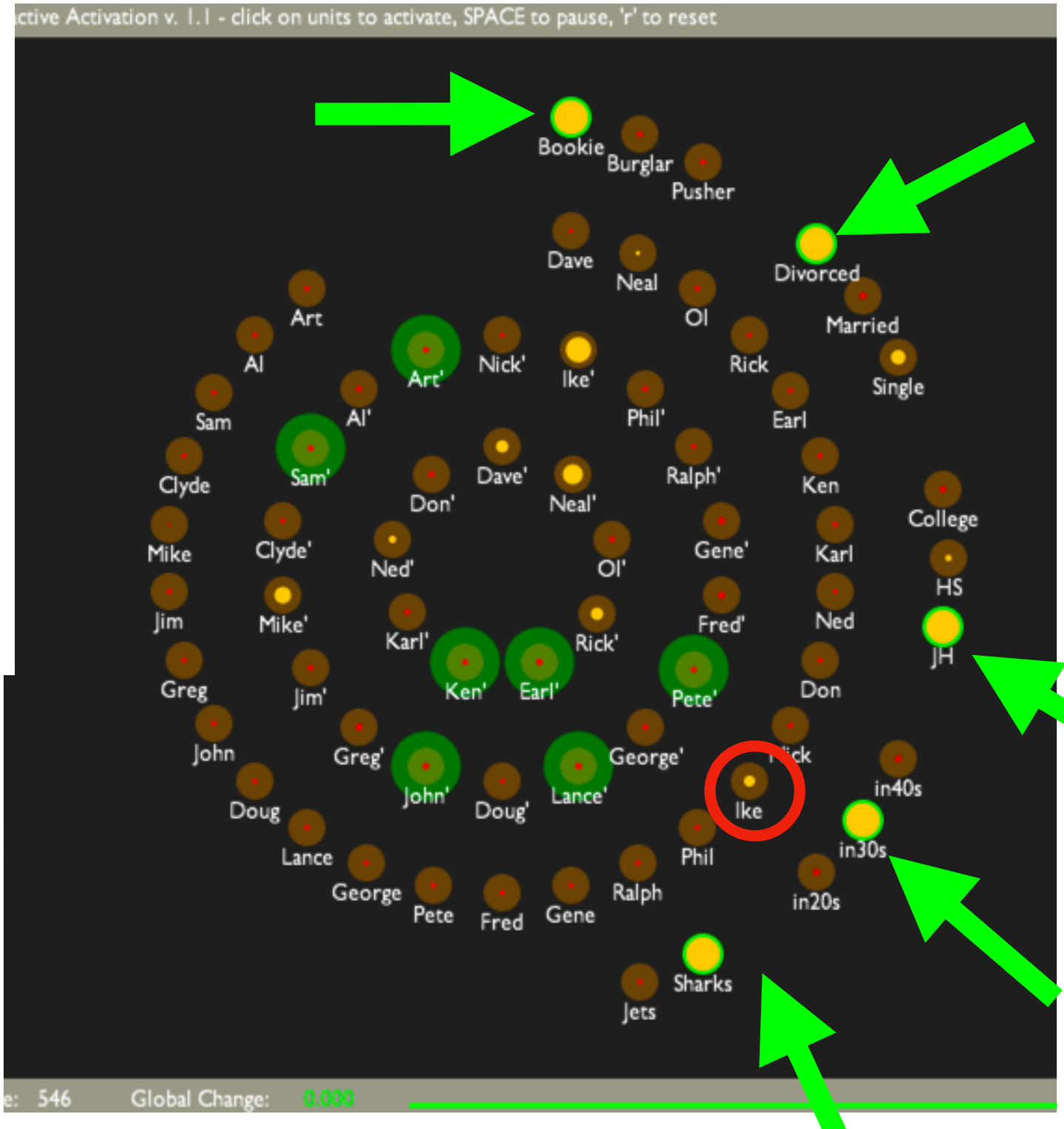
Rich get richer dynamics:
If multiple units are competing, the one with stronger activation tends to dominate over time

Graceful degradation: Retrieving Ike's name from imperfect information

"Who is the Shark in their 30s, with a junior high education, that is a bookie and divorced?

(where Ike otherwise matches, except he is single)

Phil	Sharks	30's	COL.	Mar.	Pusher
Ike	Sharks	30's	J.H.	Sing.	Bookie
Nick	Sharks	30's	H.S.	Sing.	Pusher
Don	Sharks	30's	COL.	Mar.	Burglar
Ned	Sharks	30's	COL.	Mar.	Bookie
Karl	Sharks	40's	H.S.	Mar.	Bookie
Ken	Sharks	20's	H.S.	Sing.	Burglar
Earl	Sharks	40's	H.S.	Mar.	Burglar
Rick	Sharks	30's	H.S.	Div.	Burglar
Ol	Sharks	30's	COL.	Mar.	Pusher
Neal	Sharks	30's	H.S.	Sing.	Bookie
Dave	Sharks	30's	H.S.	Div.	Pusher



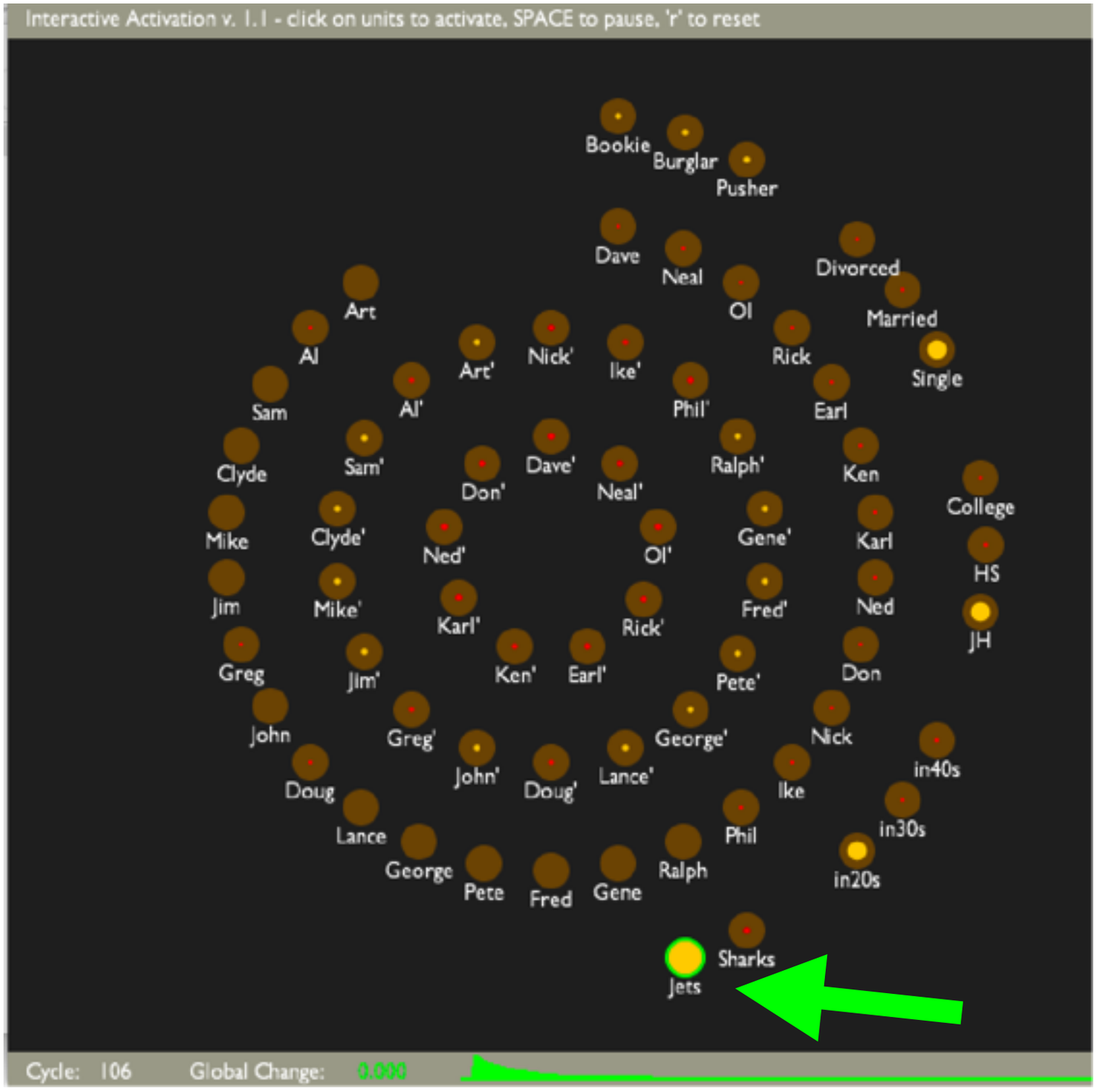
Spontaneous generalization

“What are Jets usually like?”

Name	Gang	Age	Edu	Mar	Occupation
Art	Jets	40’s	J.H.	Sing.	Pusher
Al	Jets	30’s	J.H.	Mar.	Burglar
Sam	Jets	20’s	COL.	Sing.	Bookie
Clyde	Jets	40’s	J.H.	Sing.	Bookie
Mike	Jets	30’s	J.H.	Sing.	Bookie
Jim	Jets	20’s	J.H.	Div.	Burglar
Greg	Jets	20’s	H.S.	Mar.	Pusher
John	Jets	20’s	J.H.	Mar.	Burglar
Doug	Jets	30’s	H.S.	Sing.	Bookie
Lance	Jets	20’s	J.H.	Mar.	Burglar
George	Jets	20’s	J.H.	Div.	Burglar
Pete	Jets	20’s	H.S.	Sing.	Bookie
Fred	Jets	20’s	H.S.	Sing.	Pusher
Gene	Jets	20’s	COL.	Sing.	Pusher
Ralph	Jets	30’s	J.H.	Sing.	Pusher

Common attributes:

20’s	J.H.	Single
9 of 15	9 of 15	9 of 15



Interactive activation model - Summary

Instead of entries in a database, entries and properties are implemented as simple neuron-like units, connected by mutually excitatory connections.

This simple model has complex behavior, and displays several important (and non-obvious properties!) that match well with human memory:

- **Retrieval by name -** Given a name, you can retrieve an entry's properties
- **Content addressability -** Given a few properties, you can retrieve an entry's name
- **Graceful degradation -** Retrieval is still possible if some properties are misspecified
- **Spontaneous generalization -** The model can make inferences about a typical member of a class

Review: Key principles of neural network models of cognition

- *Neurally-inspired computation.* Taking inspiration from the low-level (how neurons compute) is key to understanding the high-level (intelligence and cognition)

- *Intelligence is an emergent phenomenon.* Complex behavior can emerge from a very large number of simple, interactive computations.

- *Simulation is central.* It's hard to predict how complexity will emerge. Computational modeling and simulation are essential for understanding intelligence.

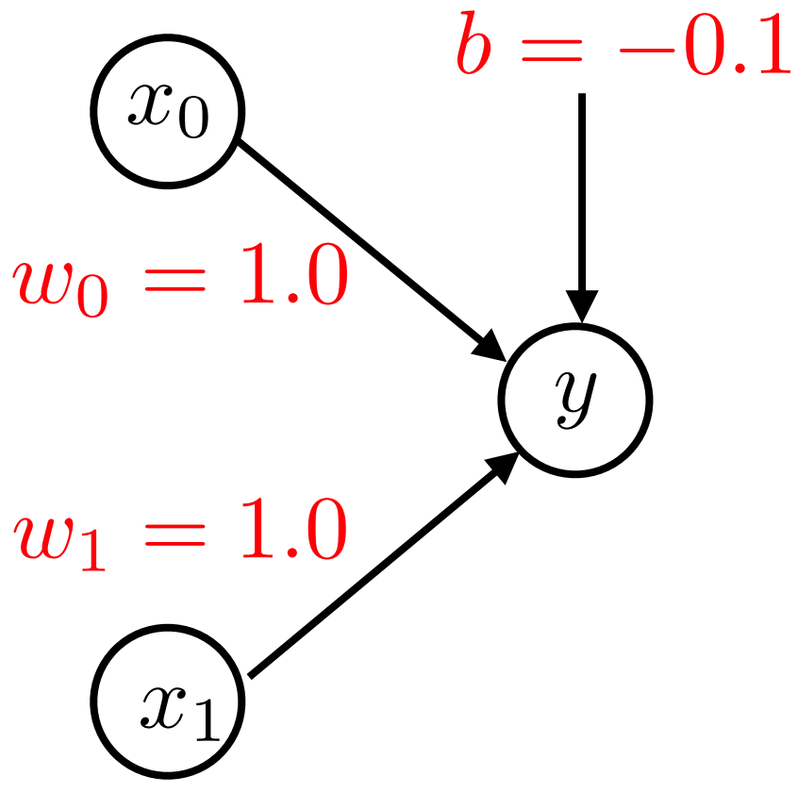
What about learning?

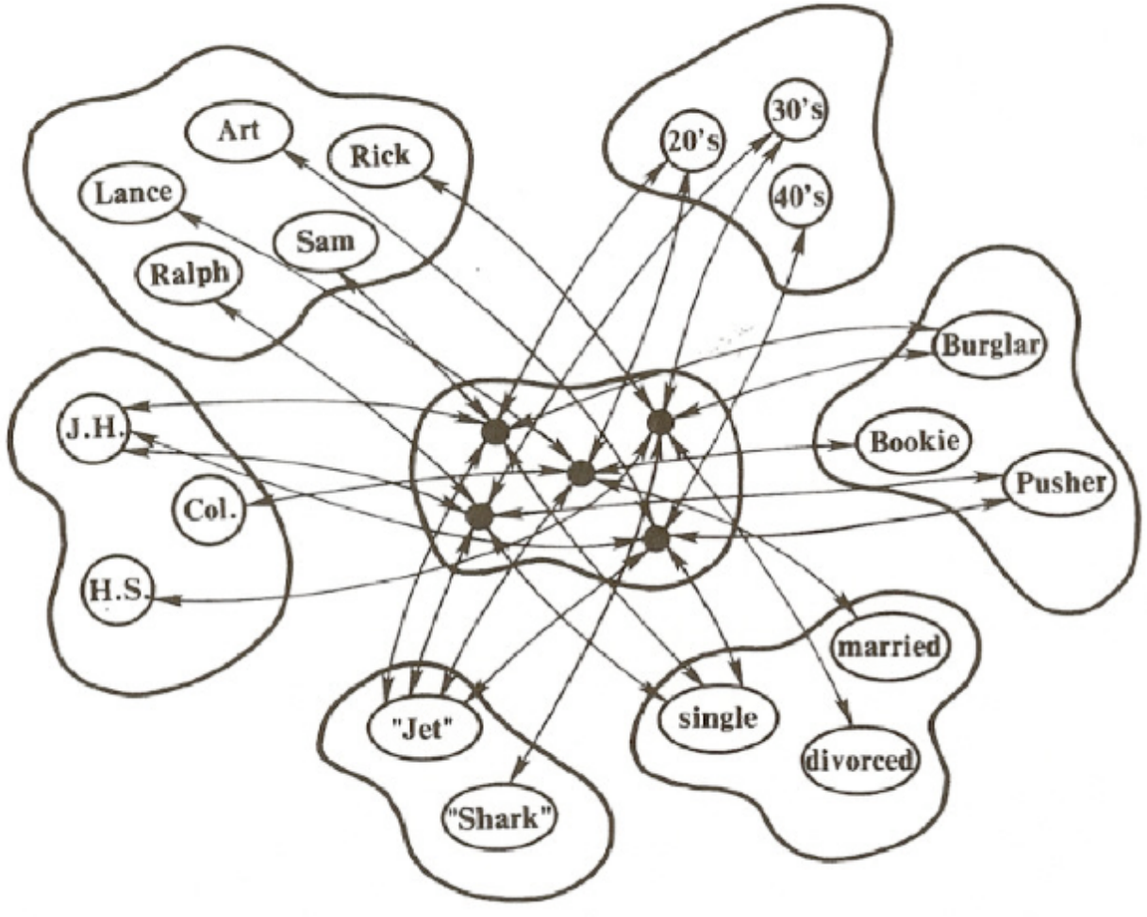
- In the previous examples, the neural networks were simple enough that we could set the connection strengths by hand to perform the desired computations.

- This is not a feasible general strategy, since models get much more complex

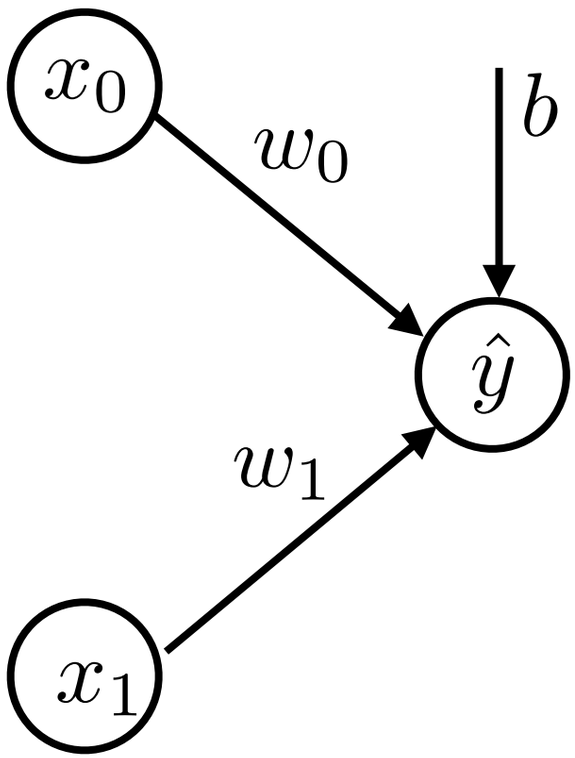
- Moreover, we would like computational tools for trying to understand learning and cognitive development

In practice, connection strengths are almost never set by hand.





Learning by optimizing an objective function

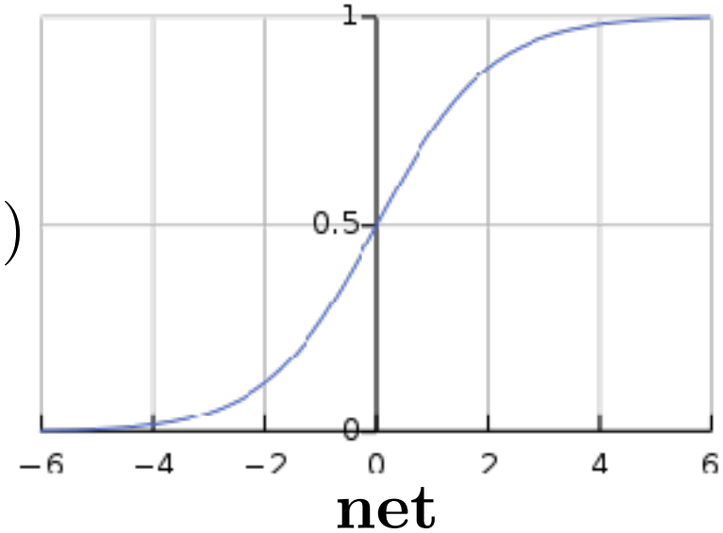


$$\hat{y} = g(\sum_i x_i w_i + b)$$

activation function:

"sigmoid" or "logistic function"

$$g(\mathbf{net}) = \frac{1}{1 + e^{-\mathbf{net}}}$$



x : input vector

w : weight vector

b : bias

g : activation function

$\hat{y}$: predicted output

y : target output

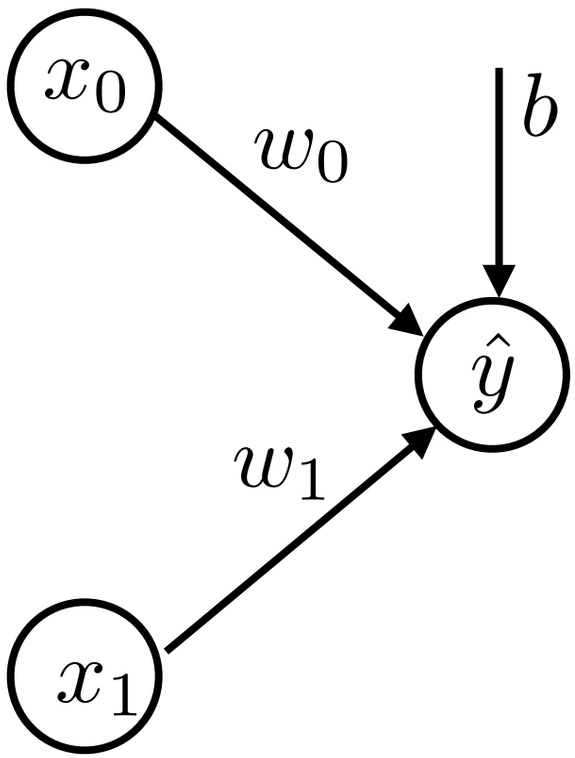
Error/loss function:

"squared error"

$$E(w, b) = (\hat{y} - y)^2$$

We want to minimize the squared difference between the predicted output and the target output (also could use "sum squared error", or "mean squared error", across multiple different predictions)

Logistic regression = single connectionist neuron

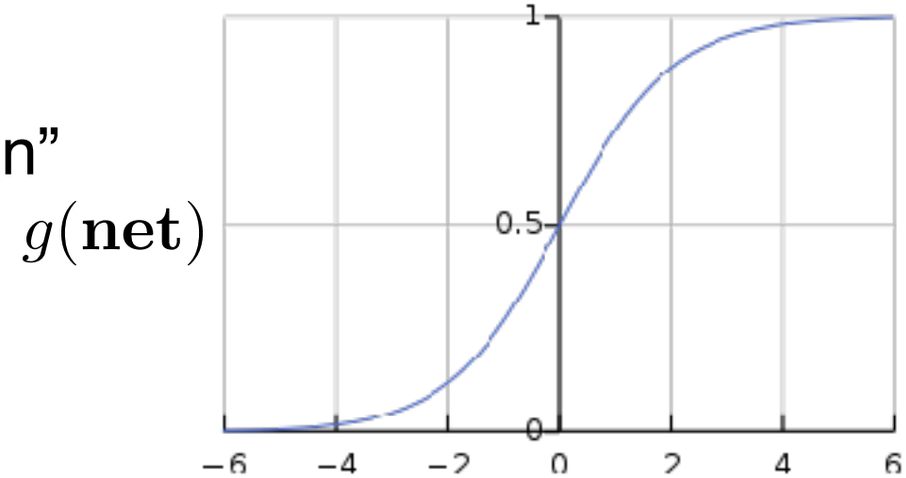


$$\hat{y} = g(\sum_i x_i w_i + b)$$

activation function:

"sigmoid" or "logistic function"

$$g(\mathbf{net}) = \frac{1}{1 + e^{-\mathbf{net}}}$$



x : input vector
w : weight vector
b : bias
g : activation function
$\hat{y}$: predicted output
y : target output

Error/loss functions:

"squared error"

$$E(w, b) = (\hat{y} - y)^2$$

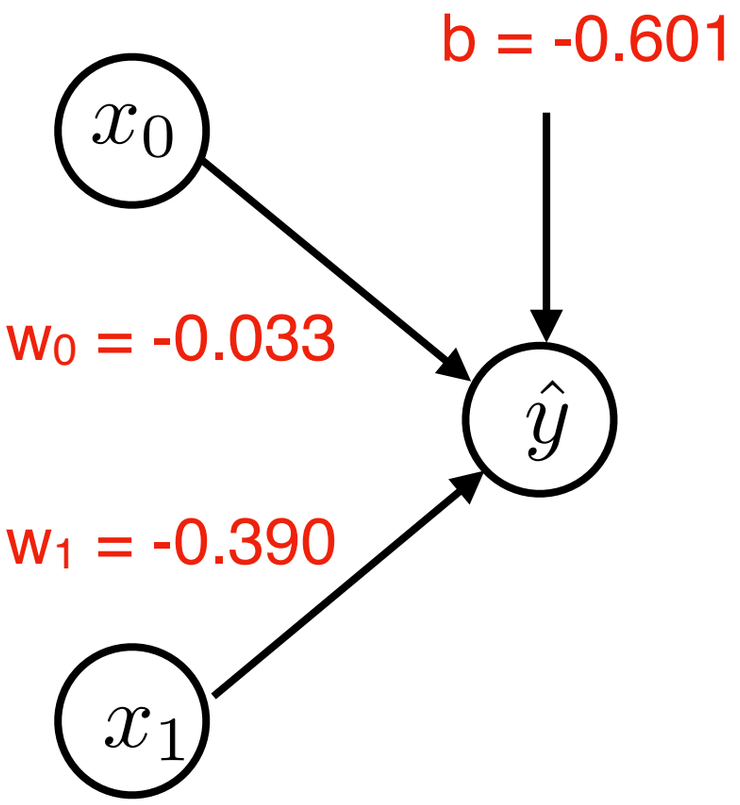
"negative log-likelihood"

$$E(w, b) = -\log[\hat{y}^y (1 - \hat{y})^{1-y}]$$

Some differences:

- Logistic regression only accepts 0 or 1 discrete output, whereas this formulation can accept 0-1 continuous output
- Logistic regression uses a negative log-likelihood loss, so it is fitting a maximum likelihood estimate

A naive algorithm: Parameter wiggling



$$\hat{y} = g(\sum_i x_i w_i + b) \quad g(\mathbf{net}) = \frac{1}{1 + e^{-\mathbf{net}}}$$

$$E(w, b) = (\hat{y} - y)^2$$

Computing the error:

x = [1, 1]

$\hat{y}$ = 0.26 (predicted output)

y = 1.0 (target)

E(w,b) = 0.54 (squared error)

logical OR

x_0	x_1	y
0	0	0
0	1	1
1	0	1
1	1	1

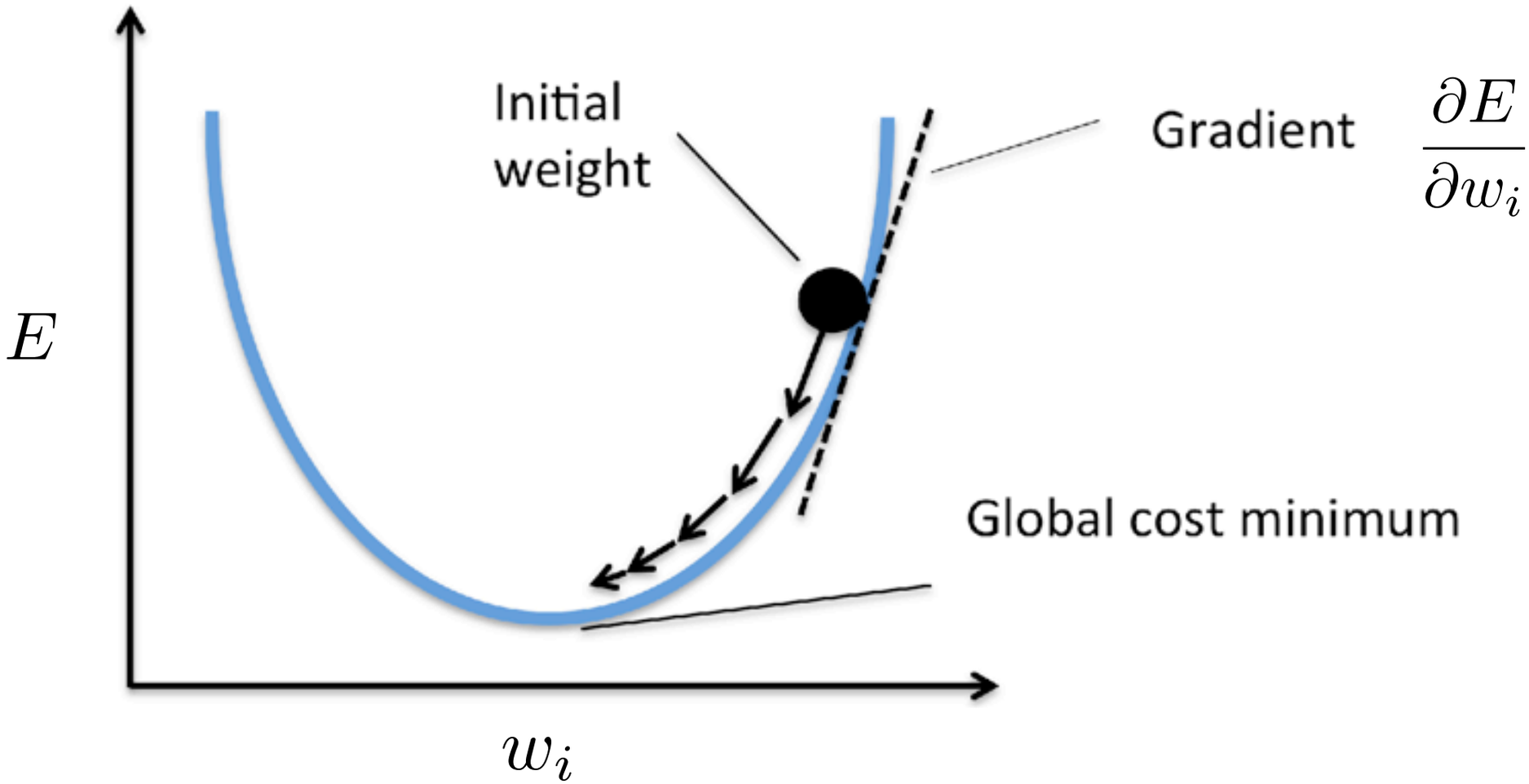
What if we try wiggling each parameter?

If we *increase* w_0 by 0.02, our error E(w,b) becomes 0.51

If we *decrease* w_0 by 0.02, our error E(w,b) becomes 0.55

Thus, let's increase w_0 by 0.02!
Now we have improved our error

A smarter algorithm: stochastic gradient descent

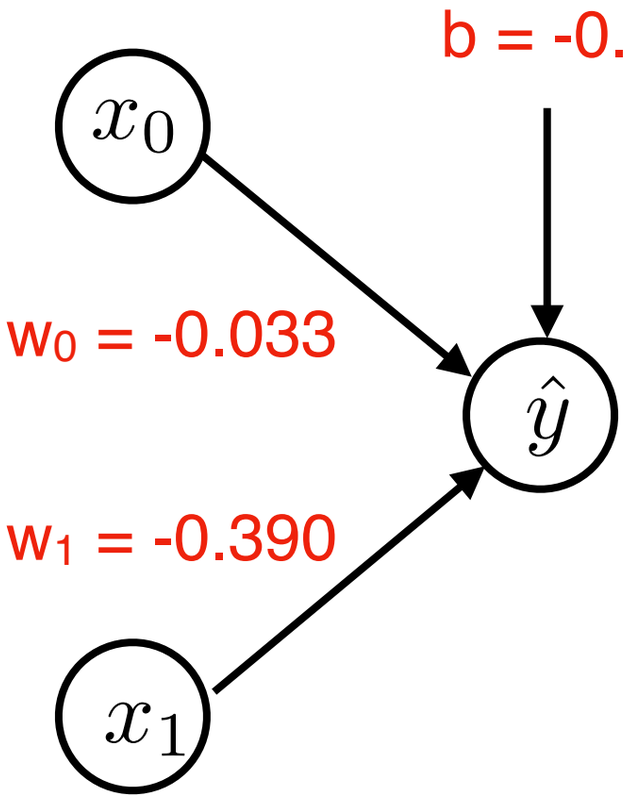


Computing the gradient tells us which direction to go for steepest descent:

$$w_i \leftarrow w_i - \alpha \frac{\partial E}{\partial w_i} \qquad \alpha : \text{learning rate}$$

Credit https://rasbt.github.io/mlxtend/

Learning via stochastic gradient descent



x : input vector

w : weight vector

b : bias

g : activation function

$\hat{y}$: predicted output

y : target output

$$\hat{y} = g(\sum_i x_i w_i + b) \quad g(\mathbf{net}) = \frac{1}{1 + e^{-\mathbf{net}}}$$

$$E(w, b) = (\hat{y} - y)^2$$

logical OR

x_0	x_1	y
0	0	0
0	1	1
1	0	1
1	1	1

Computing the error:

x = [1, 1]

$\hat{y}$ = 0.26 (predicted output)

E(w,b) = 0.54 (error)

Computing the gradient:

$$\frac{\partial E(w, b)}{\partial w_i} = 2(\hat{y} - y)g(\mathbf{net})(1 - g(\mathbf{net}))x_i$$

$$= 2(0.26 - 1)(.26)(1 - .26)1$$

$$= -0.285$$ (if we increase weight, error goes down)

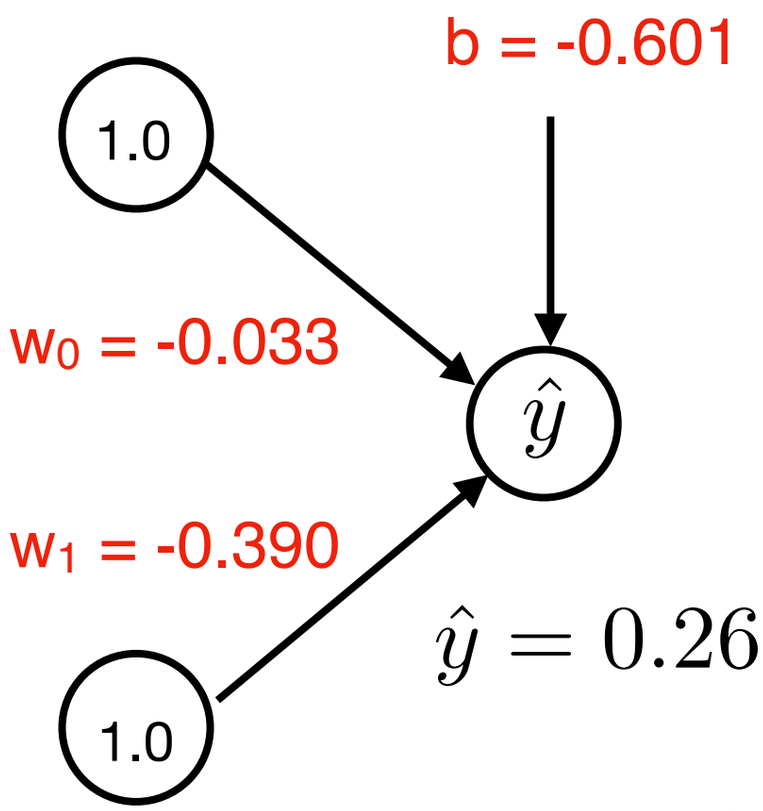
Update with gradient descent

$$w_i \leftarrow w_i - \alpha \frac{\partial E}{\partial w_i}$$

α : learning rate

Visualizing an update to the weights

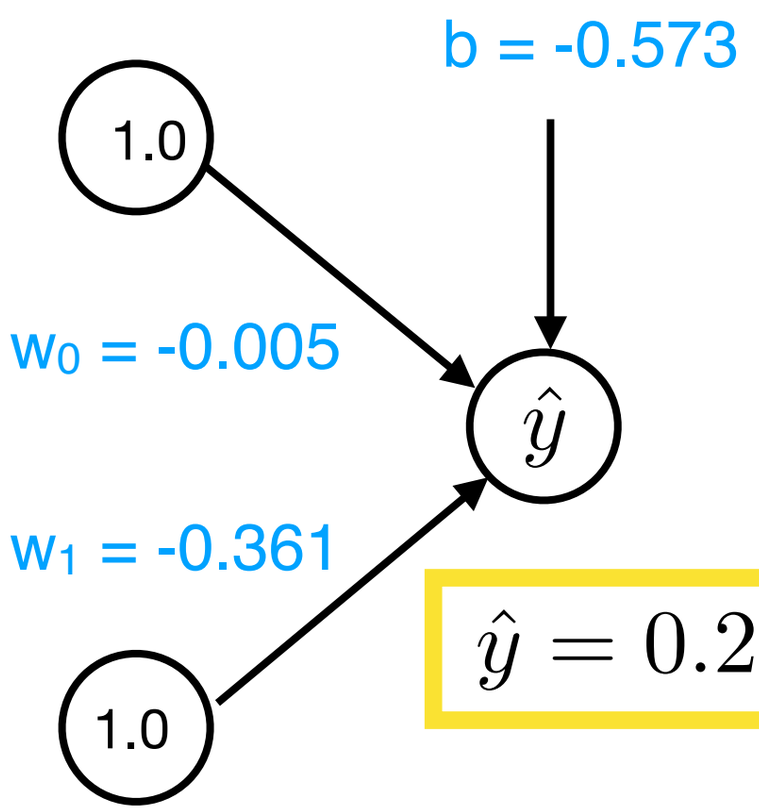
Before update



$$\hat{y} = 0.26$$

Update

After update
(all parameters increased)



$$\hat{y} = 0.28$$

Update rules

$$w_0 \leftarrow w_0 - \alpha(-0.285)$$
$$w_1 \leftarrow w_1 - \alpha(-0.285)$$

(similar update for bias)

Learning rate

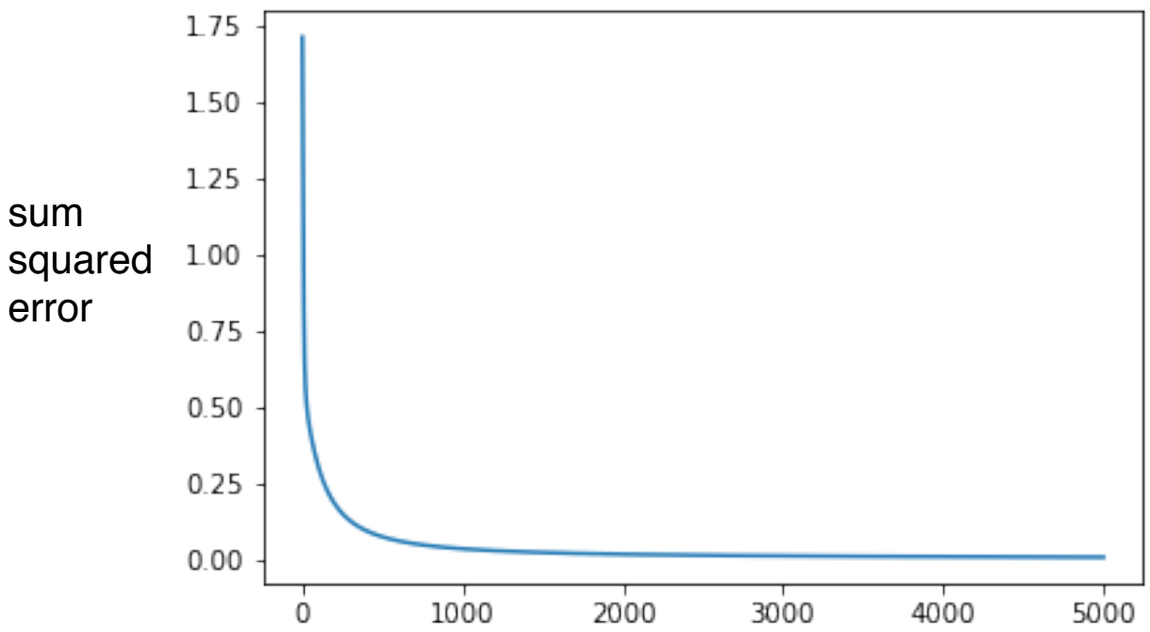
$$\alpha = 0.1$$

Notice that the new prediction is a little bit better! (closer to the target *y = 1.0*)

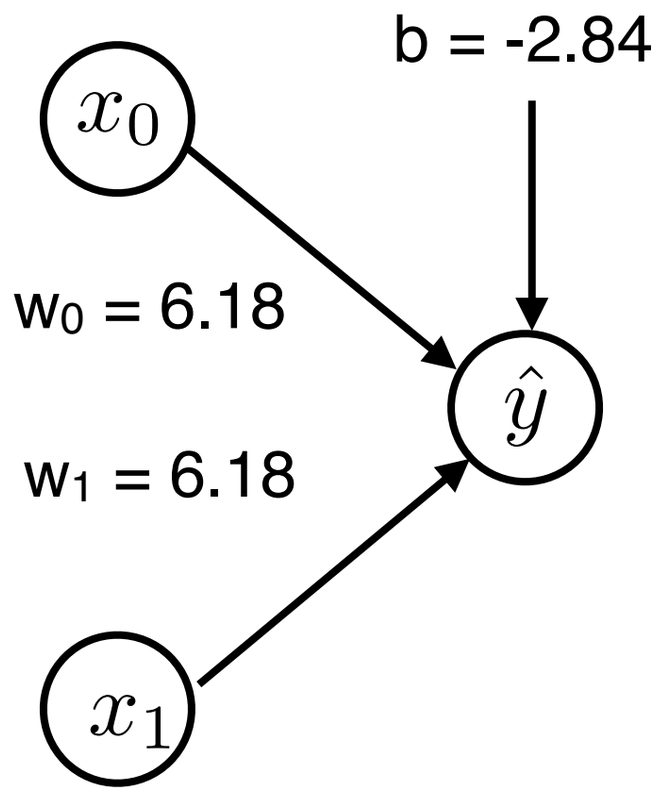
Learning via stochastic gradient descent

We repeatedly cycle through each pattern in the training set, making updates after each pattern as in the previous slide.

Error over time



Final network

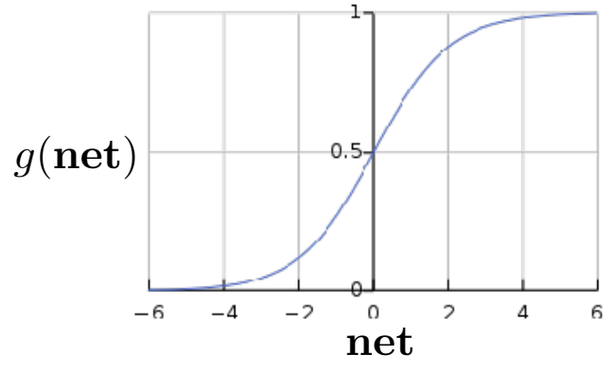


Learned function (logical OR)

x_0	x_1	$\hat{y}$	y
0	0	0.05	0
0	1	0.97	1
1	0	0.97	1
1	1	0.99	1

Pretty good fit!

$$g(\mathbf{net}) = \frac{1}{1 + e^{-\mathbf{net}}}$$



Computing the gradient

Definitions

$$E(w,b) = (\hat{y} - y)^2$$
$$= (g(\text{net}) - y)^2$$

$$\hat{y} = g(\mathbf{net}) = g(\sum_i x_i w_i + b)$$

For logistic function, we have this useful property:

$$\frac{\partial g(\mathbf{net})}{\partial \mathbf{net}} = g(\mathbf{net})(1 - g(\mathbf{net}))$$

Chain rule

$$\frac{\partial E(u)}{\partial w} = \frac{\partial E(u)}{\partial u}\frac{\partial u}{\partial w}$$

where $u = f(w)$

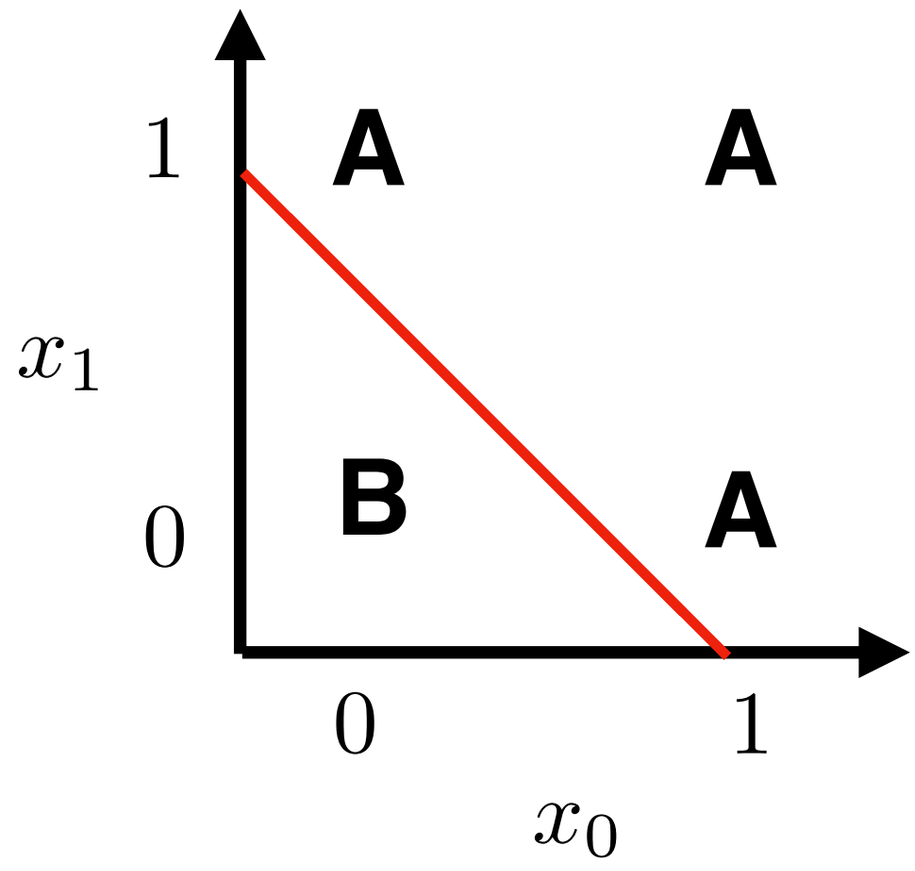
Computing the gradient of the error with respect to the weight:

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial g(\mathbf{net})}\frac{\partial g(\mathbf{net})}{\partial \mathbf{net}}\frac{\partial \mathbf{net}}{\partial w_i}$$
$$= 2(\hat{y} - y)\frac{\partial g(\mathbf{net})}{\partial \mathbf{net}}\frac{\partial \mathbf{net}}{\partial w_i}$$
$$= 2(\hat{y} - y)g(\mathbf{net})(1 - g(\mathbf{net}))\frac{\partial \mathbf{net}}{\partial w_i}$$
$$= 2(\hat{y} - y)g(\mathbf{net})(1 - g(\mathbf{net}))x_i$$

Limits of linear classifiers

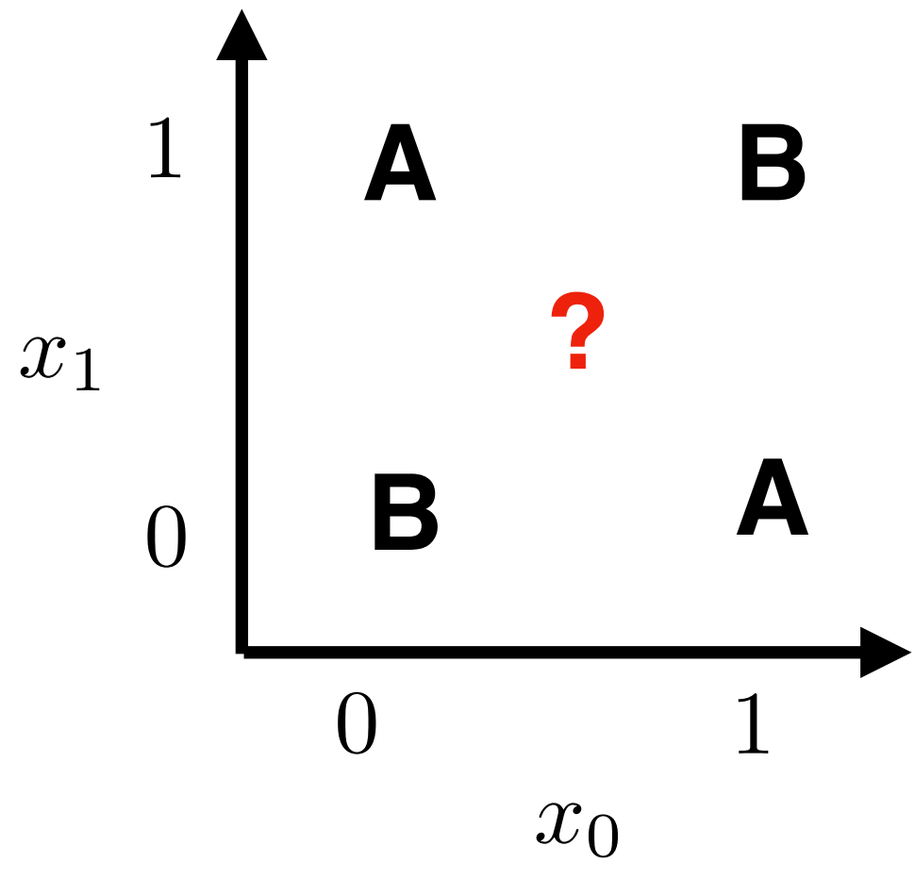
Linearly separable

Class A vs. B (logical OR)



Non-linearly separable

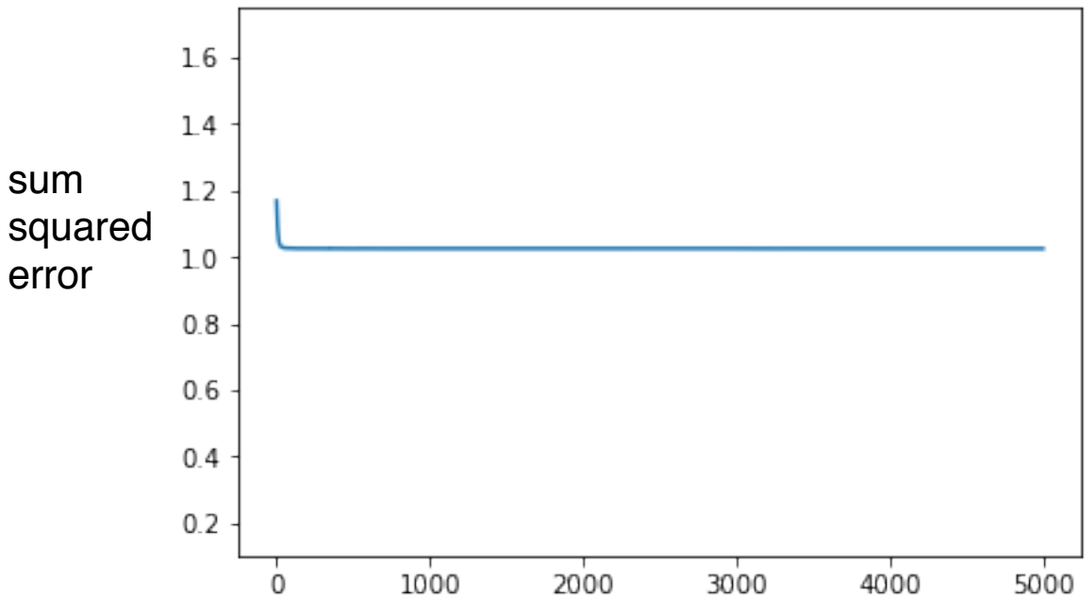
Class A vs. B (logical XOR)



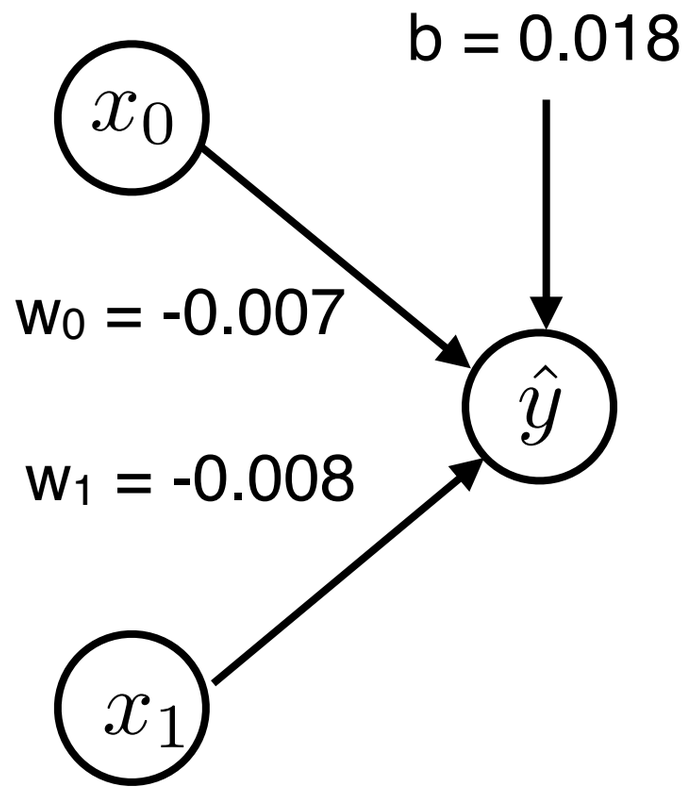
Failing to learn XOR with a linear classifier

("one or the other, but not both")

Error over time



Final network



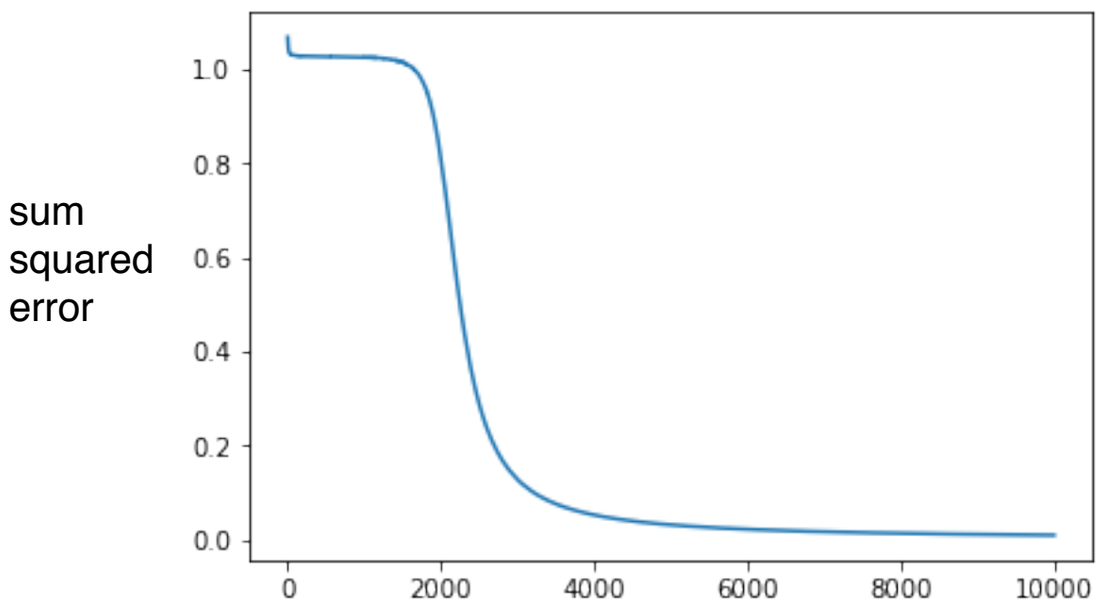
Learned function (logical XOR)

x_0	x_1	$\hat{y}$	y
0	0	0.50	0
0	1	0.50	1
1	0	0.50	1
1	1	0.50	0

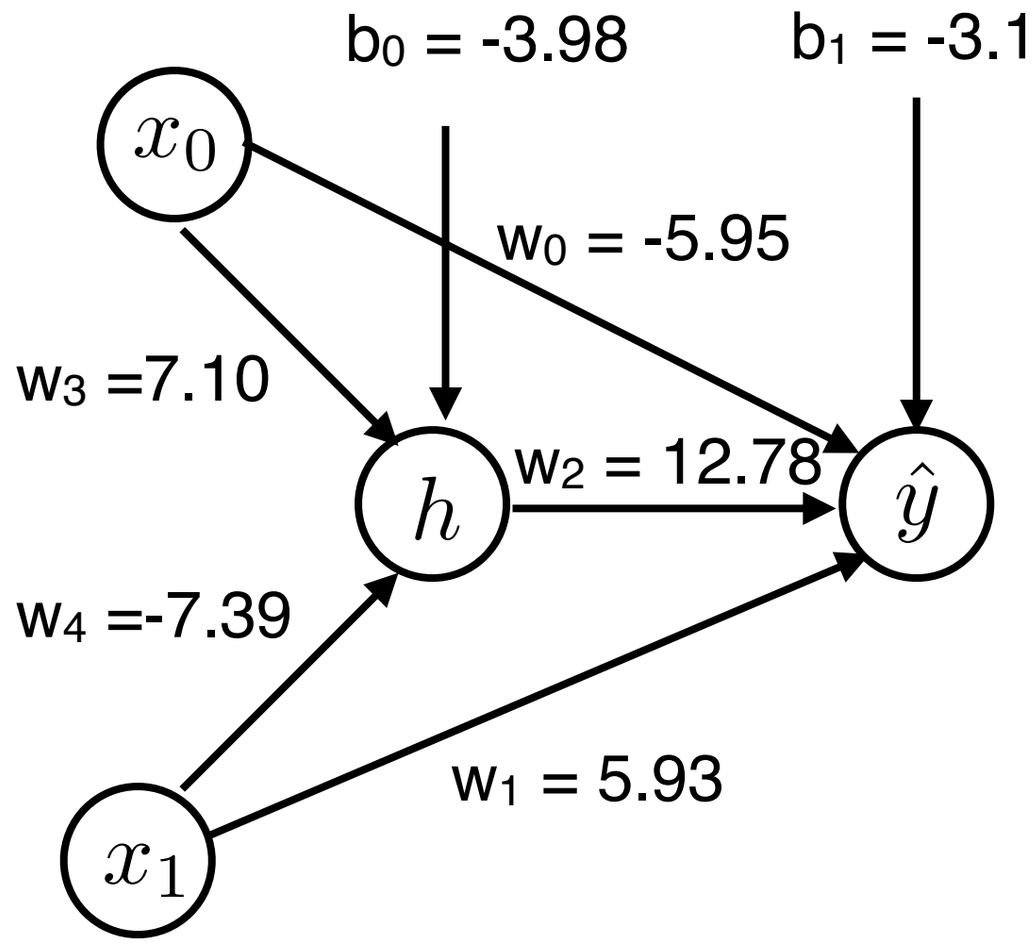
Terrible fit!

Learning XOR with a multi-layer classifier

Error over time



Final network



Learned function (logical XOR)

x_0	x_1	$\hat{y}$	y
0	0	0.05	0
0	1	0.94	1
1	0	0.96	1
1	1	0.05	0

Pretty good fit!

Backpropagation algorithm for computing gradient

Updates for these weights the same as before:

$$\frac{\partial E}{\partial w_0} \quad \frac{\partial E}{\partial w_1} \quad \frac{\partial E}{\partial w_2}$$

What about the other weights?

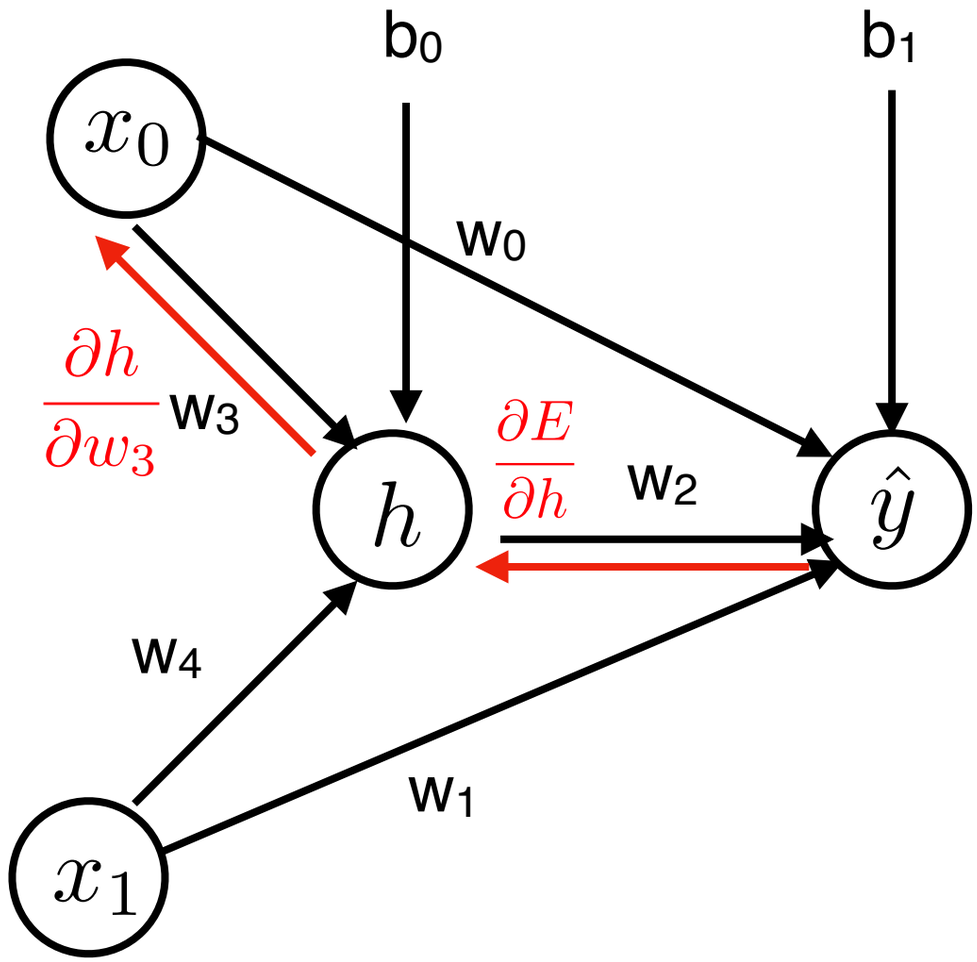
$$\frac{\partial E}{\partial w_3} \quad \frac{\partial E}{\partial w_4}$$

Multi-step strategy:

$$\frac{\partial E}{\partial w_3} = \frac{\partial E}{\partial h}\frac{\partial h}{\partial w_3}$$

Step 1) Compute how error changes as a function of hidden unit activation

Step 2) Compute how hidden unit activation changes as a function of weight



Backpropagation algorithm for computing gradient

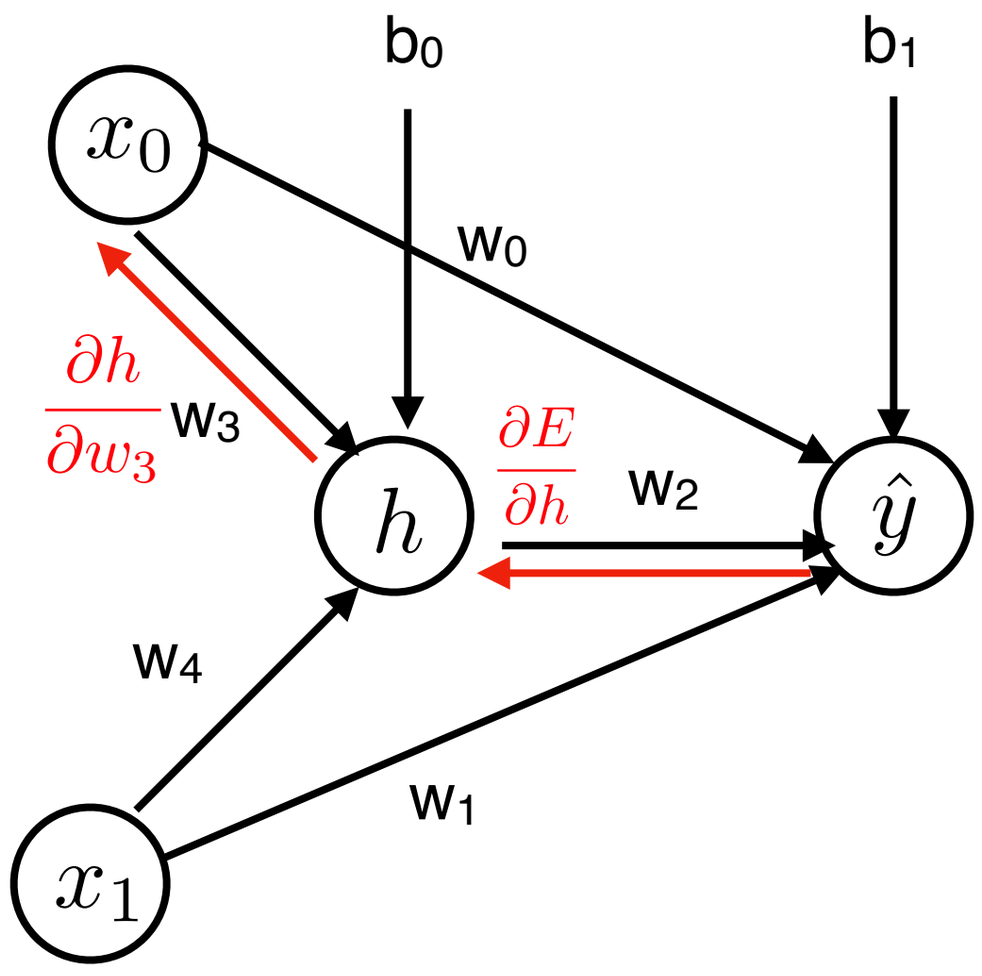
Multi-step strategy:

$$\frac{\partial E}{\partial w_3} = \frac{\partial E}{\partial h}\frac{\partial h}{\partial w_3} = \frac{\partial E}{\partial g(\mathbf{net}_h)}\frac{\partial g(\mathbf{net}_h)}{\partial w_3}$$

Step 1) Compute how error changes as a function of hidden unit activation *(we worked most of this step out already for single layer net)*

$$\frac{\partial E}{\partial g(\mathbf{net}_h)} = \frac{\partial E}{\partial g(\mathbf{net}_y)}\frac{\partial g(\mathbf{net}_y)}{\partial \mathbf{net}_y}\frac{\partial \mathbf{net}_y}{\partial g(\mathbf{net}_h)}$$

$$= 2(\hat{y} - y)g(\mathbf{net}_y)(1 - g(\mathbf{net}_y))w_2$$



Step 2) Compute how hidden unit activation changes as a function of weight

$$\frac{\partial g(\mathbf{net}_h)}{\partial w_3} = \frac{\partial g(\mathbf{net}_h)}{\partial \mathbf{net}_h}\frac{\partial \mathbf{net}_h}{\partial w_3}$$

$$= g(\mathbf{net}_h)(1 - g(\mathbf{net}_h))x_0$$

As before, update with gradient descent:

$$w_i \leftarrow w_i - \alpha\frac{\partial E}{\partial w_i} \quad \alpha : \text{learning rate}$$

Conceptually, still no different than wiggling w_3 and seeing how error changes!

Fortunately, modern software for training neural networks compute the gradients for you!

This is all the PyTorch code you need to update the weights in the XOR model.

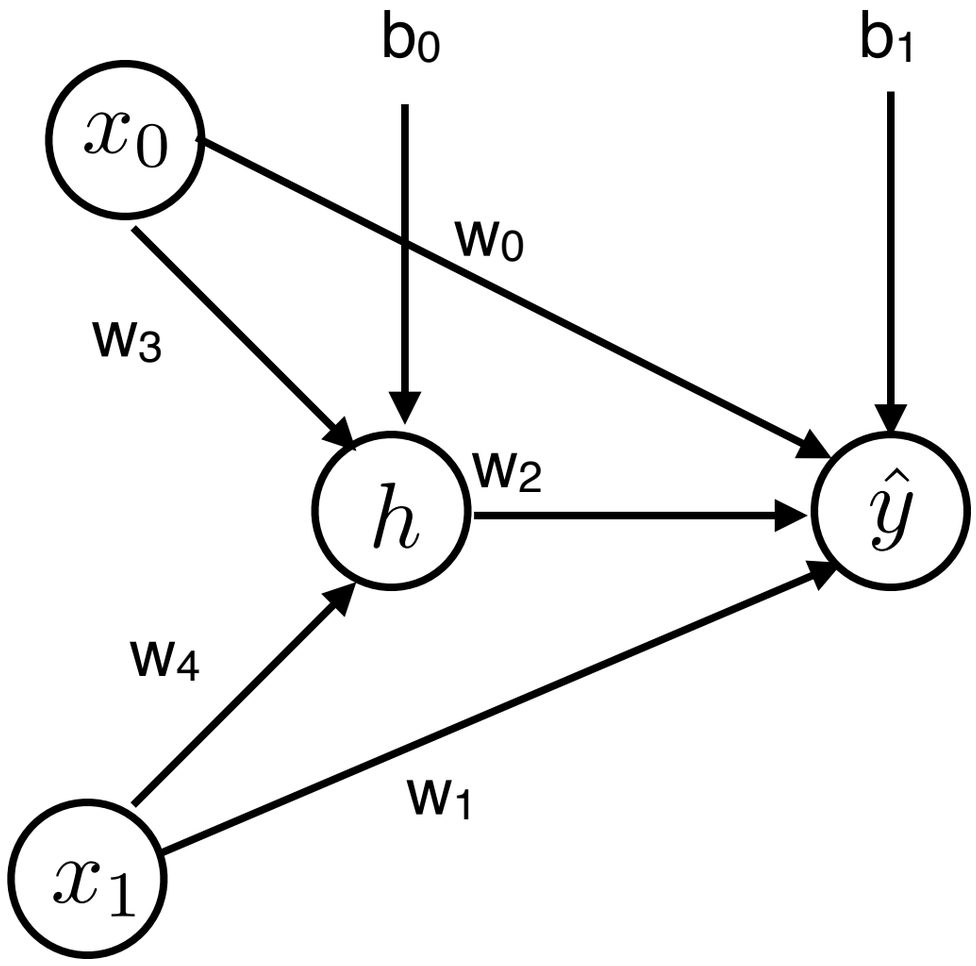
You now know what is going on under the hood.

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.i2h = nn.Linear(2, 1)
        self.all2o = nn.Linear(3, 1)

    def forward(self, x):
        netinput_h = self.i2h(x)
        h = F.sigmoid(netinput_h)
        x2 = torch.cat((x,h))
        netinput_y = self.all2o(x2)
        out = F.sigmoid(netinput_y)
        return out

def update(pat, target, net):
        net.train()
        optimizer.zero_grad()
        output = net(pat)
        loss = F.mse_loss(output, target, size_average=False)
        loss.backward()
        optimizer.step()
```

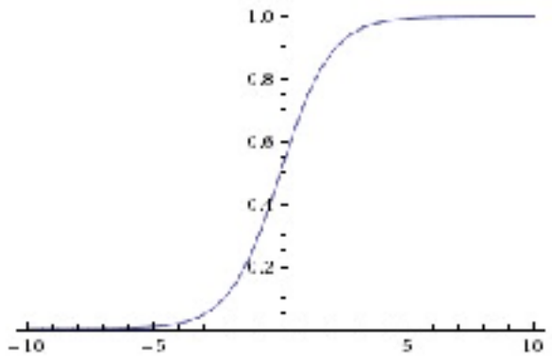


(this line computes all of the gradients for you.)

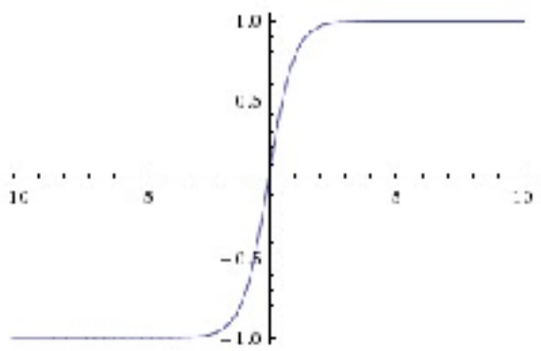
Important tricks for training neural networks

- **The learning rate is extremely important**. Your model may not learn if you set the rate too high or too low. Often, you want to start the rate high and decrease it over the course of learning. There are variants of gradient descent such as “Adam” (Kingma & Ba, 2017) that are faster and automatically adjust the learning rate.
- **Mini-batch learning**. Usually we don’t update the weights after every input pattern. Instead, we present a set of patterns in a “batch,” add their gradients together, and compute a single update to the weights for the whole mini-batch. This is more stable and usually much faster (especially if training your network on a GPU)
- **The are much better activation functions than the “sigmoid”.**

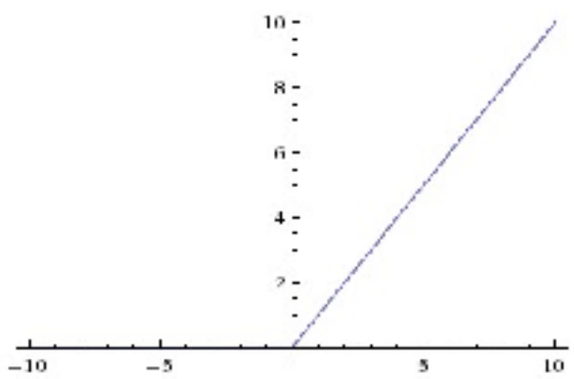
sigmoid (bad)



tanh (better)



ReLU (best; train much faster)



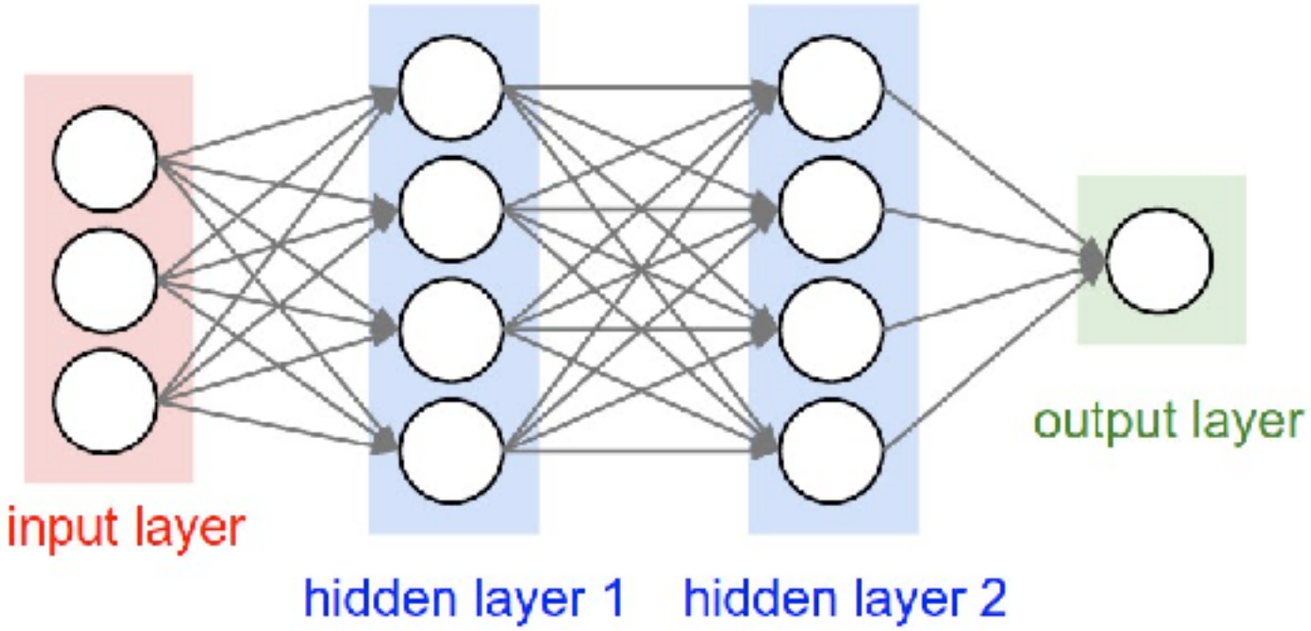
- **Classification requires a different loss and activation function**.

softmax output layer (for c possible classes) negative log-likelihood loss

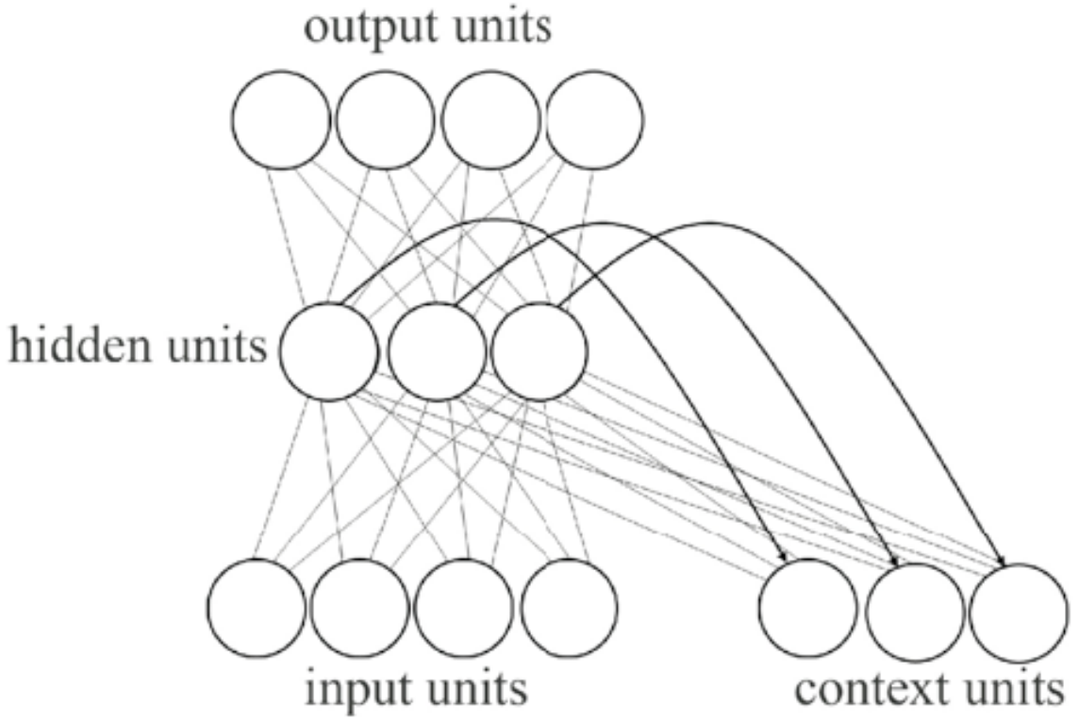
$$g(\mathbf{net}_i) = \frac{e^{\mathbf{net}_i}}{\sum_c e^{\mathbf{net}_c}}$$

Backpropagation allows us to efficiently optimize a wide range of “deep neural network” models

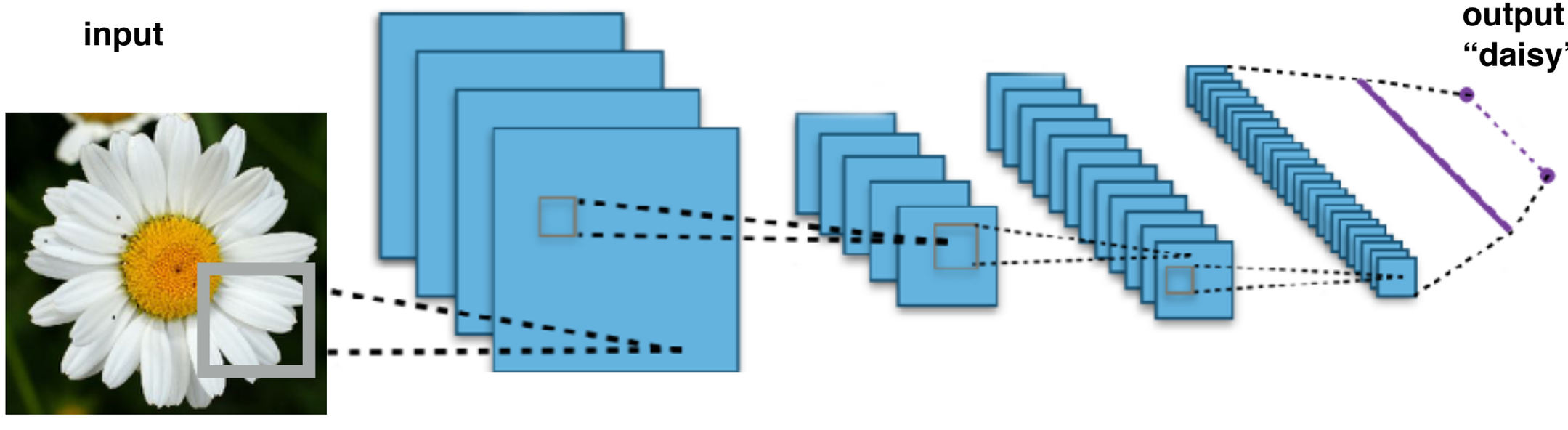
Deep fully-connected neural network



Recurrent neural network



Deep convolutional neural network



Key principles of neural network models of cognition

- *Neurally-inspired computation.* Taking inspiration from the low-level (how neurons compute) is key to understanding the high-level (intelligence and cognition)

- *Intelligence is an emergent phenomenon.* Complex behavior can emerge from a very large number of simple, interactive computations.

- *Simulation is central.* It's hard to predict how complexity will emerge. Computational modeling and simulation are essential for understanding intelligence.

Key principles of deep learning for data science and machine learning

- *Neurally-inspired computation.* Take (**very**) loose inspiration from the brain when designing learning algorithms (mostly in two ways: multiple layers of computation, and simple units for collective computation)

- *Generic architectures.* Model architecture should be as generic as possible. When in doubt, let the data decide rather than building in assumptions.

- *Learning is central.* Models should have a very large number of parameters, and you should feed your model as much data as possible.

- *Learning from raw data.* Learn from the raw data if possible. It is better to learn a representation of your data than to hand-craft a representation.

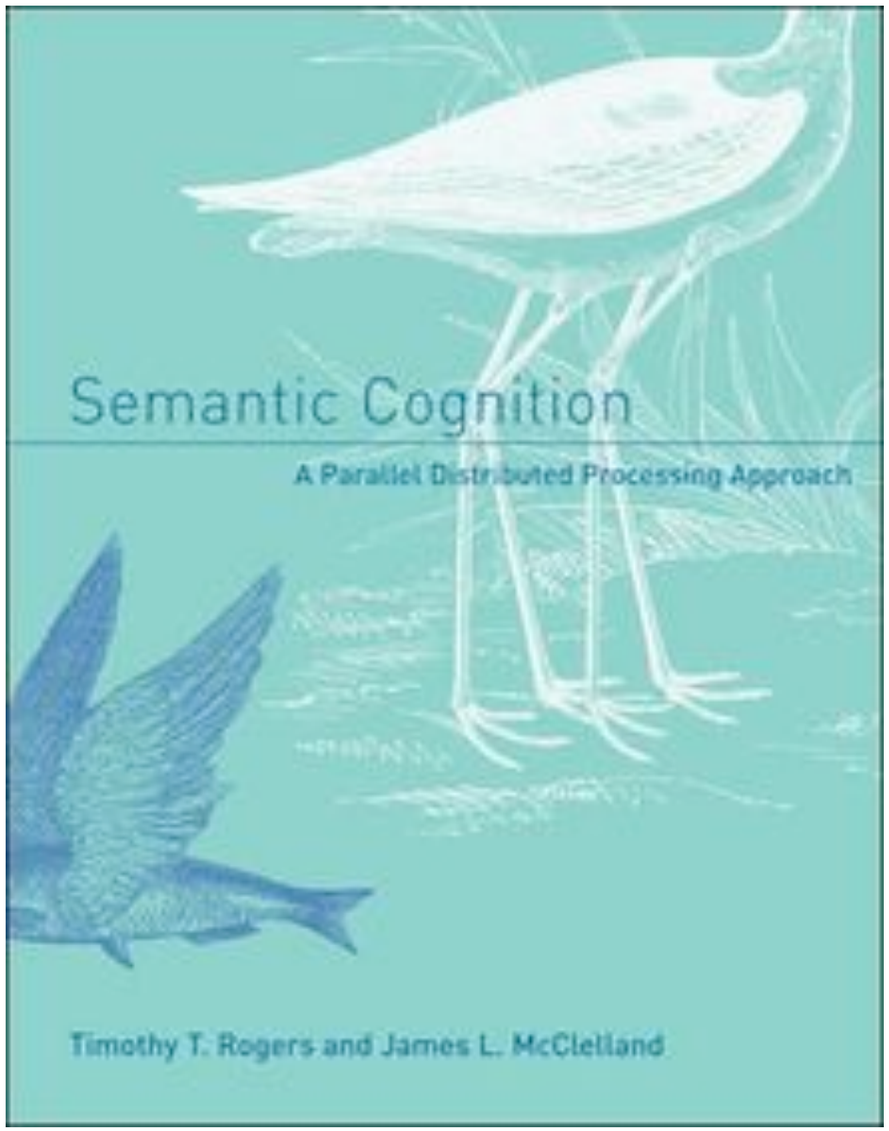
- Gradient-based learning. Gradient descent is very effective in high-dimensional parameter spaces (e.g., millions of weights).

Modeling semantic cognition with a multi-layer neural net trained with backpropagation

THE PARALLEL DISTRIBUTED PROCESSING APPROACH TO SEMANTIC COGNITION

James L. McClelland and Timothy T. Rogers‡*

How do we know what properties something has, and which of its properties should be generalized to other objects? How is the knowledge underlying these abilities acquired, and how is it affected by brain disorders? Our approach to these issues is based on the idea that cognitive processes arise from the interactions of neurons through synaptic connections. The knowledge in such interactive and distributed processing systems is stored in the strengths of the connections and is acquired gradually through experience. Degradation of semantic knowledge occurs through degradation of the patterns of neural activity that probe the knowledge stored in the connections. Simulation models based on these ideas capture semantic cognitive processes and their development and disintegration, encompassing domain-specific patterns of generalization in young children, and the restructuring of conceptual knowledge as a function of experience.



How do we know that Socrates is mortal? Aristotle suggested that we reason from two propositions, in this case: Socrates is a man; and all men are mortal. This classical SYLLOGISM forms the basis of many theories of how we attribute properties to individuals. First we categorize them, then we consult properties known to apply to members of the category. Another answer — the one that we and a growing community of researchers would give — is that the knowledge that Socrates is mortal is latent in the connections among the neurons in the brain that process semantic information. In this article, we contrast this approach with other proposals, including a hierarchical propositional approach that grows out of the classical tradition. We show how it can address several findings on the acquisition of SEMANTIC KNOWLEDGE in development and its disintegration in dementia. It can also capture a set of phenomena that have motivated the idea that semantic cognition rests on innately specified, intuitive, domain-specific theories. Although challenges remain to be addressed, this approach provides an integrated account of a wide range of phenomena, and provides a promising basis for addressing the remaining issues.

SYLLOGISM
A formal structure for deduction in argument, consisting of a major and a minor premise from which a conclusion logically follows.

**Center for the Neural Basis of Cognition and Department of Psychology, Carnegie Mellon University, 4400 Fifth Avenue, Pittsburgh, Pennsylvania 15213-2683, USA.*
‡Medical Research Council Cognition and Brain Sciences Unit, 15 Chaucer Road, Cambridge CB2 2EF, UK.
e-mails: jlm@cnbc.cmu.edu; tim.rogers@mrc-cbu.cam.ac.uk
doi:10.1038/nrn1076

The hierarchical propositional approach

In the early days of computer simulation models, researchers assumed that human semantic cognition was based on the use of categories and propositions. Quillian[1] proposed that if the concepts were organized into a hierarchy progressing from specific to general categories, then propositions true of all members of a superordinate category could be stored only once, at the level of the superordinate category. For example, propositions true of all living things could be stored at the top of the tree (FIG. 1). Other propositions, true of all animals but not of plants, could be stored at the next level down, and so on, with specific facts about an individual concept stored directly with it. To determine whether a proposition were true of a particular concept, one could access the concept, and see whether the proposition was stored there. If not, one could search at successively higher levels until the property was found, or until the top of the hierarchy was reached.

Quillian's proposal was appealing in part for its economy of storage: propositions true of many items could often be specified just once. The proposal also allowed for immediate generalization of what is known

Key questions in semantic cognition

- **Semantic cognition**: our intuitive understanding of objects and their properties

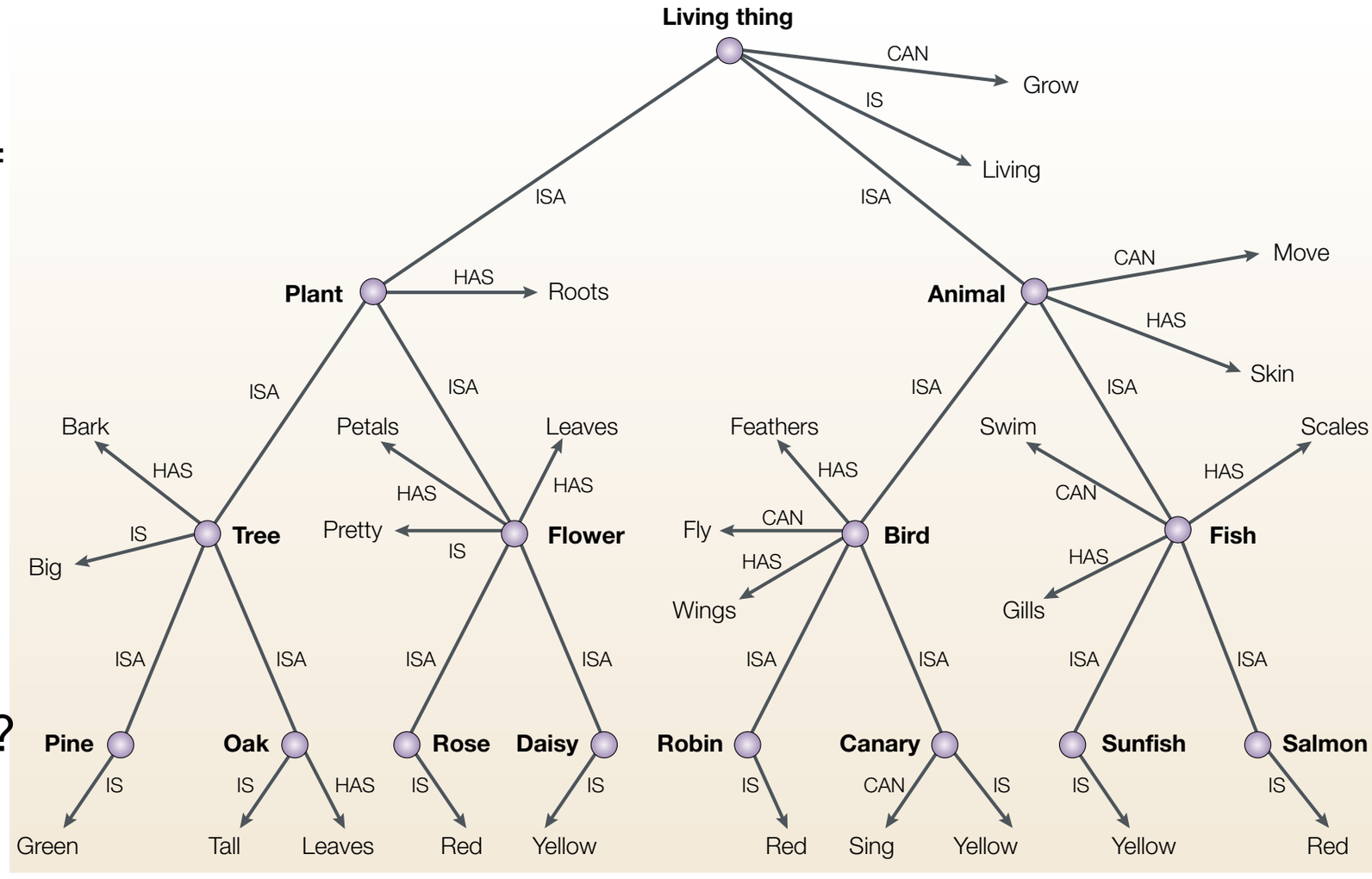
- How do we know what properties something has, and which of its properties should be generalized to other objects?

- How is the knowledge underlying these abilities acquired, and how is it affected by brain disorders?

- Can we understand semantic cognition as gradual optimization in a multi-layer neural network?

Quillian's hierarchical propositional model

- Quillian proposed that concepts are represented in a hierarchy organized from specific to general.
- Propositions true of all members of specific categories are stored only once, at the higher-level
- **Strengths of the model**
  - economy of storage
  - powerful inferences when adding a new concept
- **Problems for the model**
  - How do you handle exceptions? (e.g., a penguin that can't fly)
  - How do you decide which level to store a property?



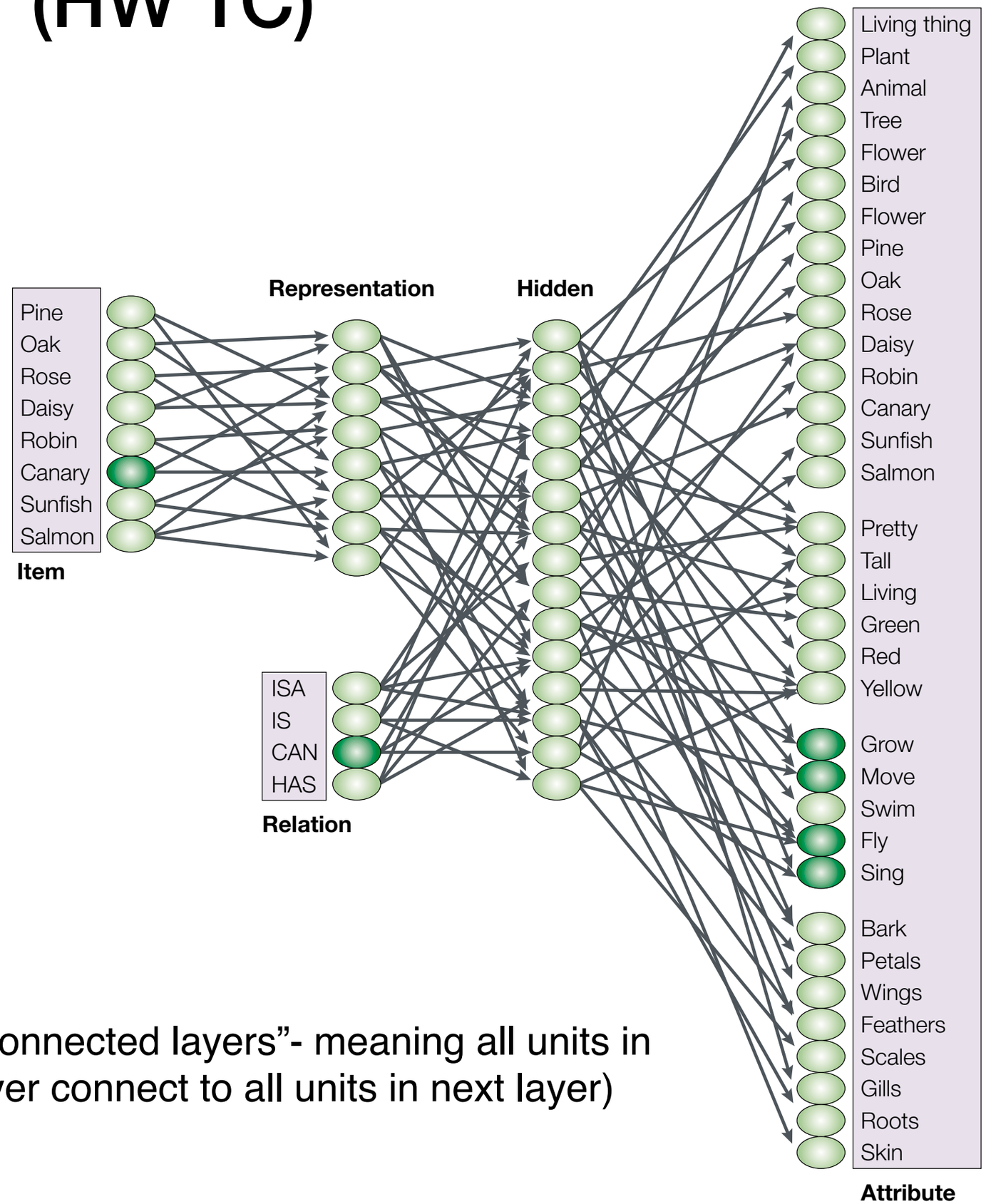
(Quillian, 1968)

A neural network model of semantic cognition (HW 1C)

- Network is trained to answer queries involving an **item** (e.g., “Canary”) and a **relation** (e.g., “CAN”), outputting all **attributes** that are true of the item/relation pair (e.g., “grow, move, fly, sing”)

- Unlike Quillian’s model, knowledge is not stored explicitly in a hierarchy. It is stored implicitly in the web of connection weights.

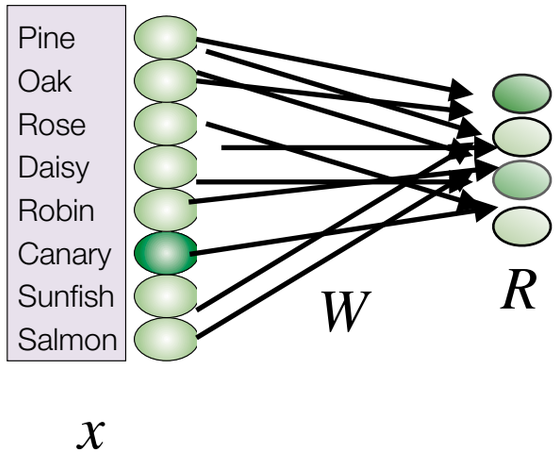
- Starting with random weights, the network is trained on all facts stored in the Quillian hierarchy on the previous slide.



(has “fully-connected layers”- meaning all units in previous layer connect to all units in next layer)

Fully-connected layers as matrix multiplication

Item/input **Representation**



two fully connected layers (all input units connect to all representation units). Some arrows not shown.

Computing activation of representation unit j

$$R_j = g(b_j + \sum_i W_{ji} x_i)$$

or equivalently in vector/matrix form

$$R = g(b + Wx)$$

Visual interpretation of

$$R = g(Wx)$$

Let's assume the bias ***b=0***

$$g(\mathbf{net}) = \frac{1}{1 + e^{-\mathbf{net}}}$$

x

0	0	0	0	0	1	0	0

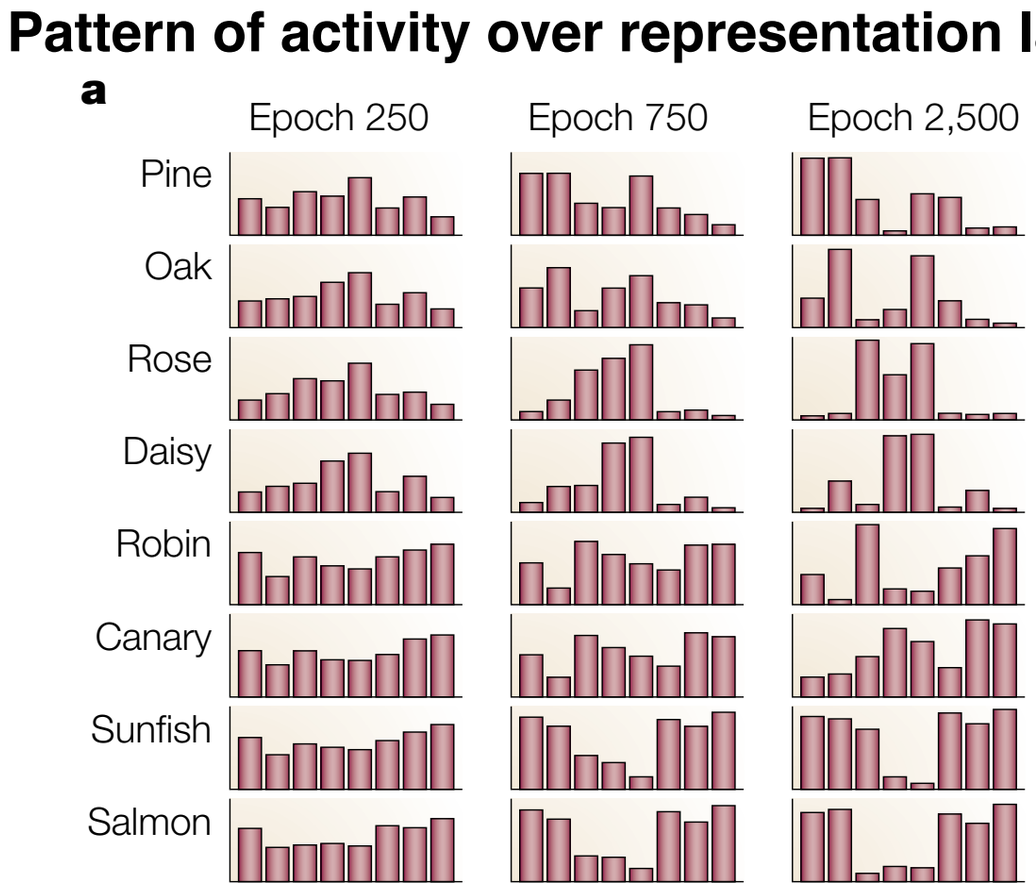
$R = g(Wx)$ Wx

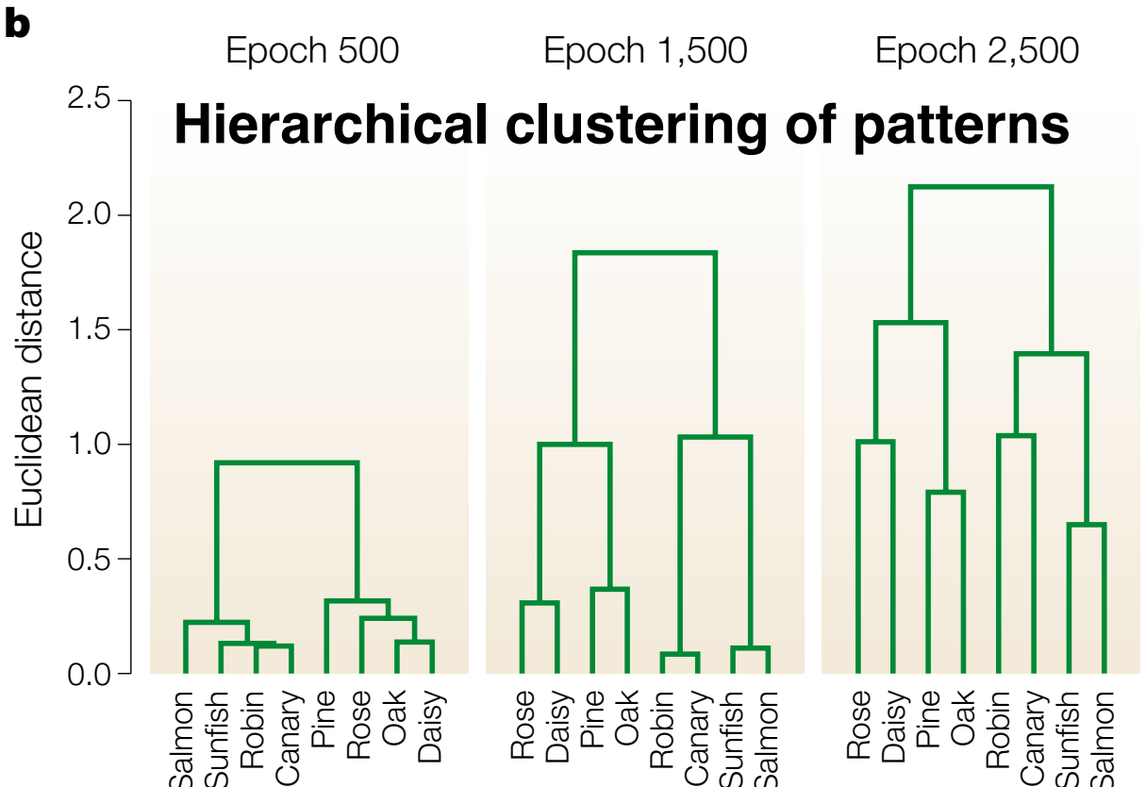
0.8	1.4
0.1	-2.0
0.5	-0.2
0.2	-1.2

W

-0.1	0.8	-0.3	1.7	1.4	1.4	-2.9	-0.5
-1.9	-0.9	0.6	0.2	-1.1	-2.0	0.8	-0.3
-0.4	0.1	-0.5	2.1	1.0	-0.2	-1.4	1.1
-1.8	-0.5	0.7	0.8	0.1	-1.2	-1.1	-0.3

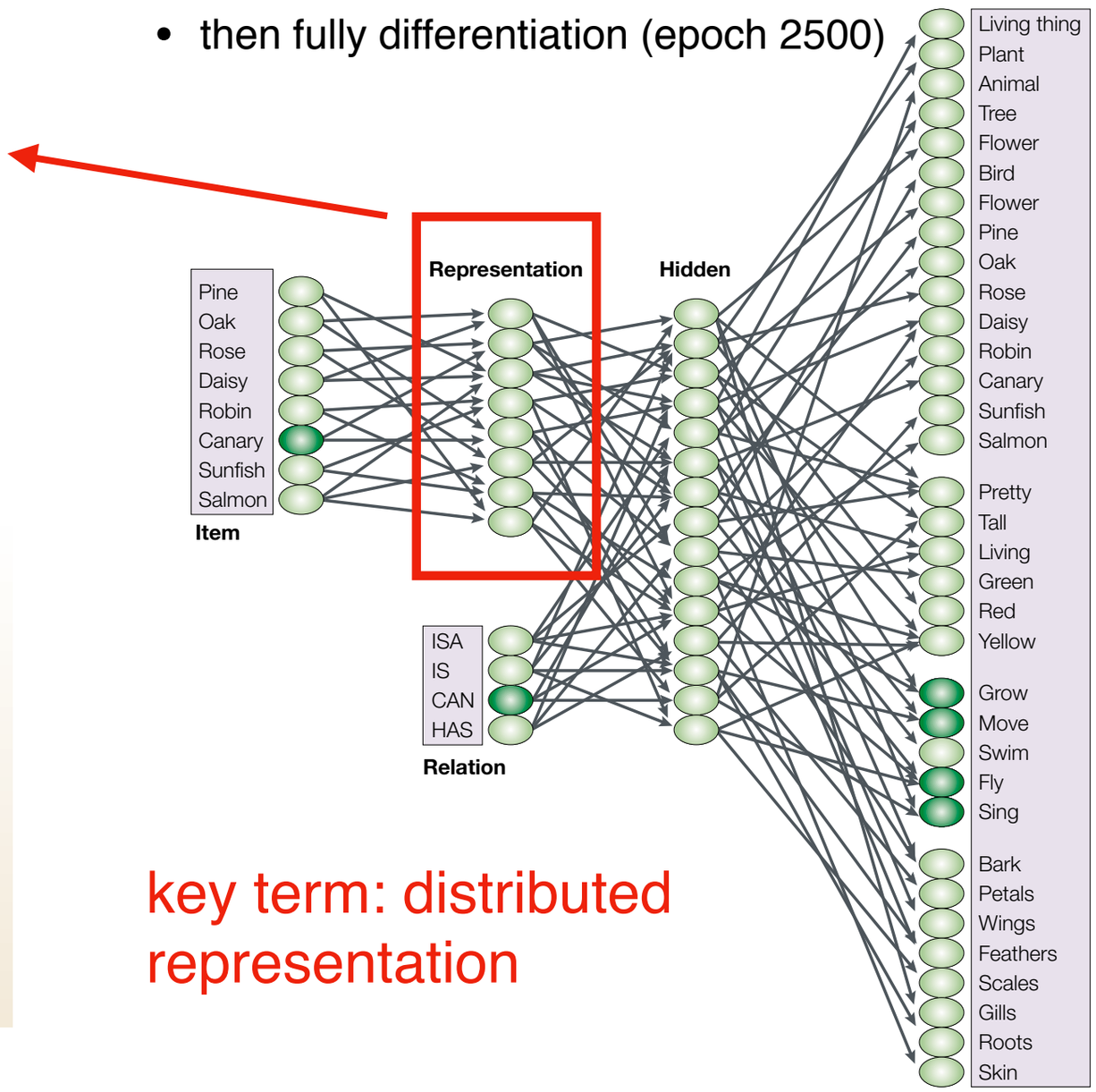
Modeling cognitive development of semantic representation through training with backpropagation





During training, model goes through stages that resemble broad-to-specific differentiation in children's cognitive development

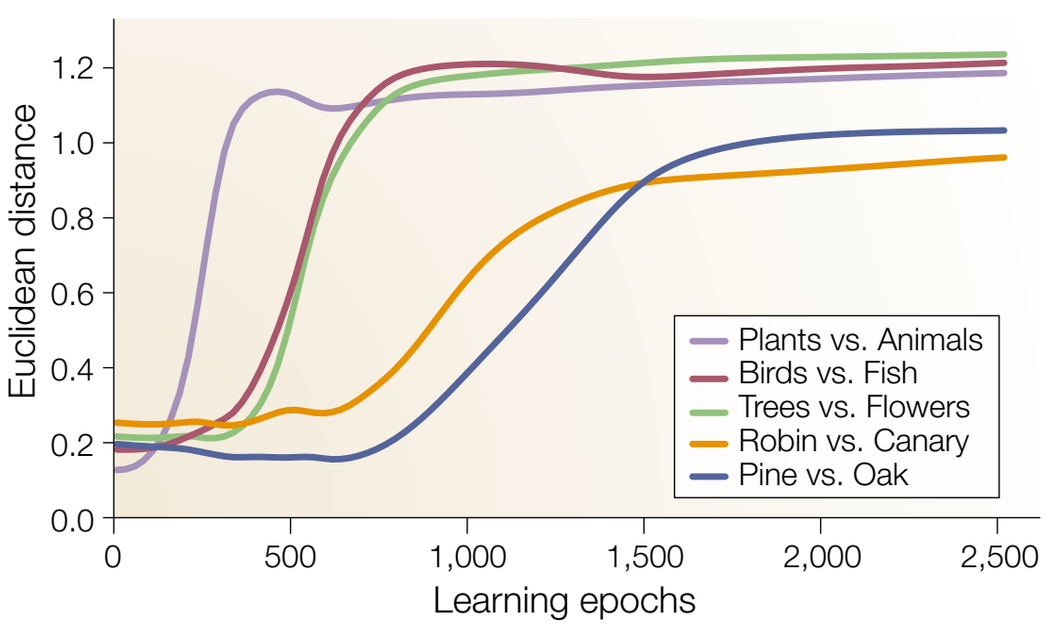
- first differentiates plants vs. animals (epoch 250)
- then birds vs. fish and trees vs. flowers (epoch 750)
- then fully differentiation (epoch 2500)



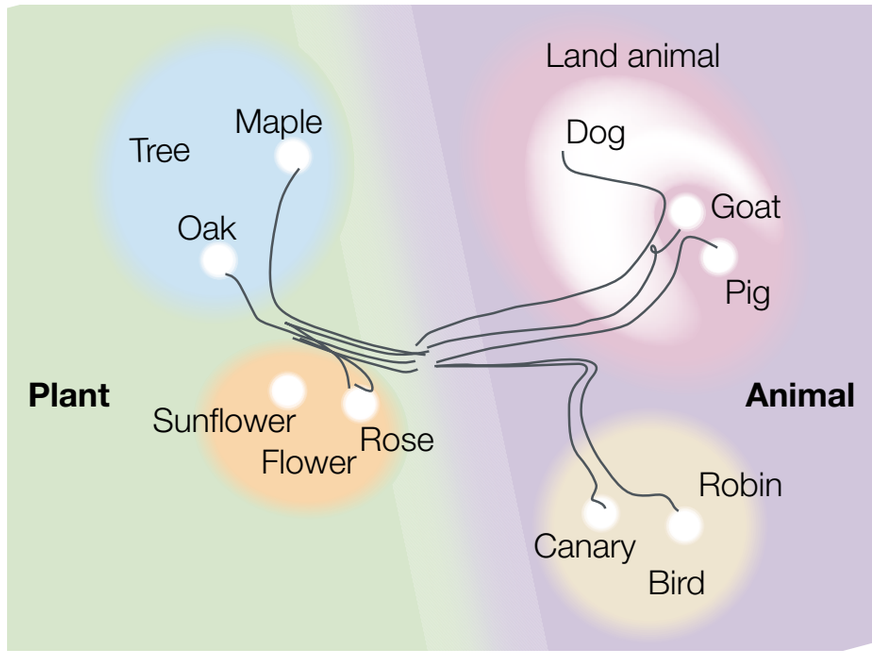
key term: distributed representation

Modeling cognitive development of semantic representation through training with backpropagation

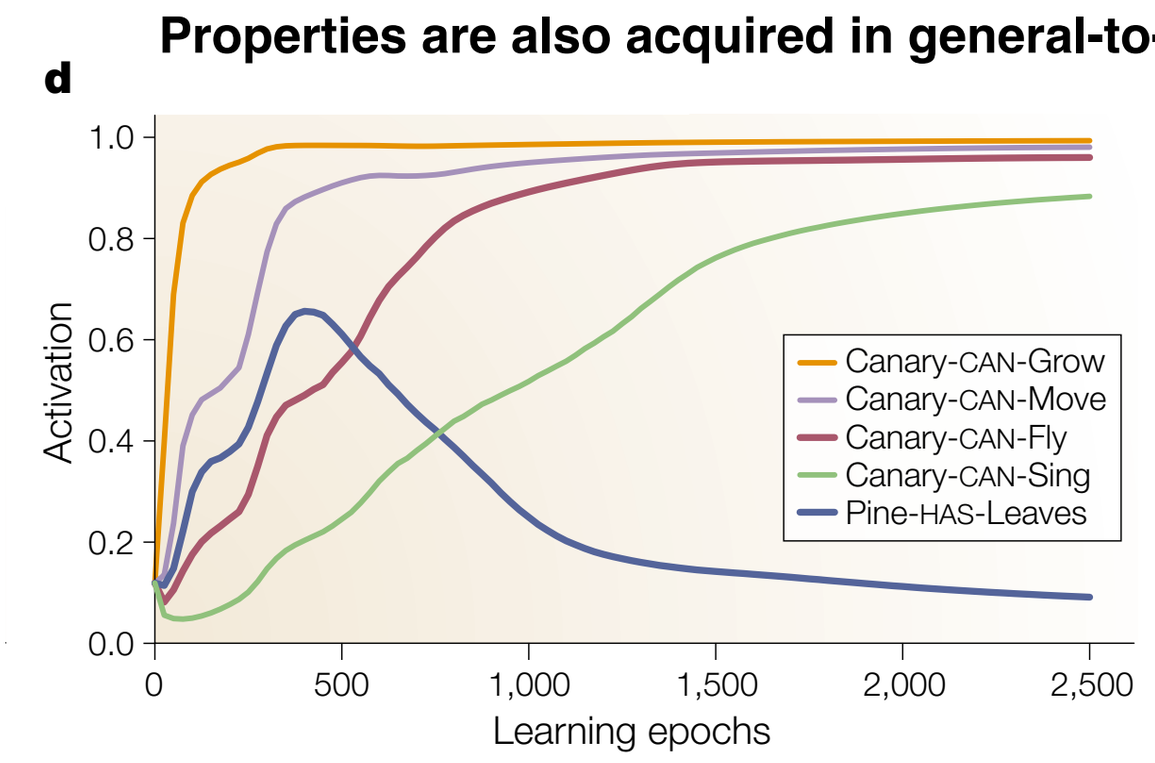
Broad-to-specific stages of conceptual development



PCA embedding over time of representation layer



Properties are also acquired in general-to-specific manner



Modeling semantic dementia by adding noise to the neural network

- For human patients with semantic dementia (a form of progressive brain damage), specific categories and properties are lost first, while more general information is preserved.

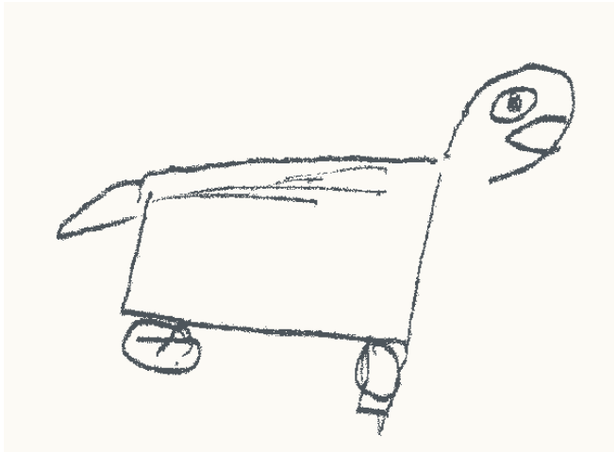
Patient naming pictures of birds

Picture naming responses for JL

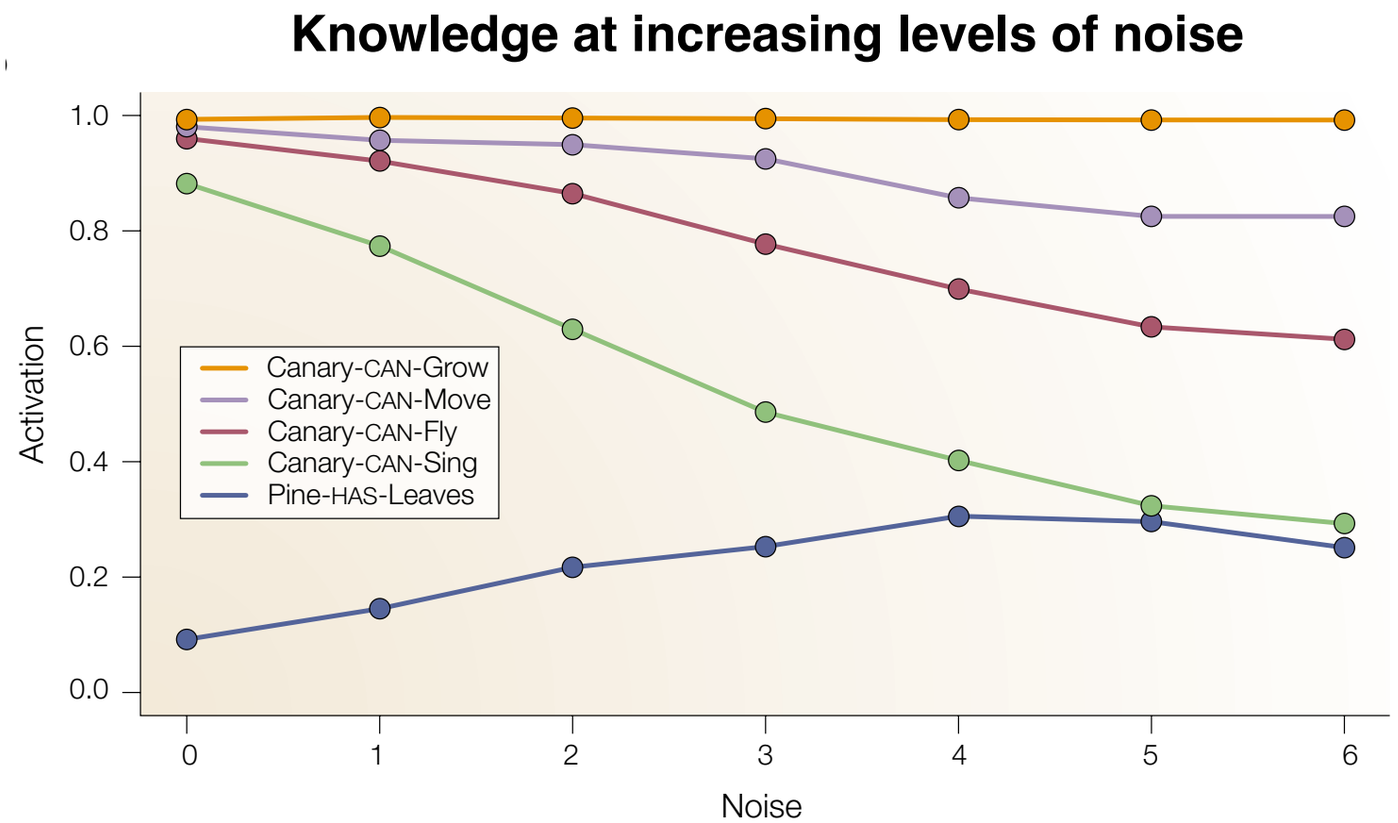
Item	Sept. 91	March 92	March 93
Bird	+	+	Animal
Chicken	+	+	Animal
Duck	+	Bird	Dog
Swan	+	Bird	Animal
Eagle	Duck	Bird	Horse
Ostrich	Swan	Bird	Animal
Peacock	Duck	Bird	Vehicle
Penguin	Duck	Bird	Part of animal
Rooster	Chicken	Chicken	Dog

Drawing a camel shows loss of specifics

c IF's delayed copy of a camel



- If random noise is added to the representation layer of the neural network, a similar pattern of losing the specifics but keeping the general information is observed.



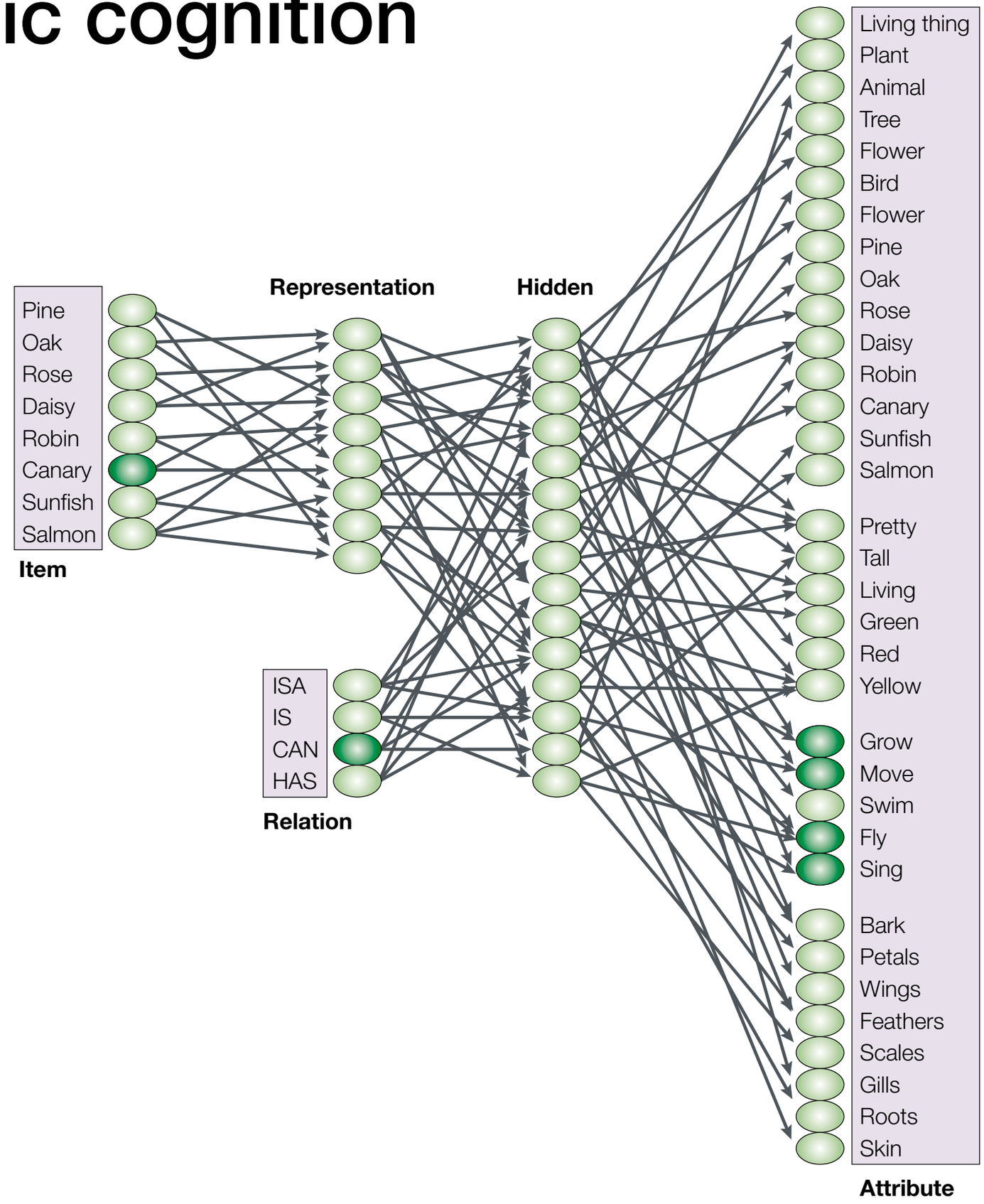
Summary: A neural network model of semantic cognition

- Network is trained to answer queries involving an **item** (e.g., “Canary”) and a **relation** (e.g., “CAN”), outputting all **attributes** that are true of the item/ relation pair (e.g., “grow, move, fly, sing”)

- Trained with stochastic gradient descent, as we learned about in this lecture

- The model helps us to understand the broad-to-specific pattern of differentiation in children’s cognitive development

- It also helps us to understand the specific-to-general deterioration in semantic dementia



Key principles of neural network models of cognition

- *Neurally-inspired computation.* Taking inspiration from the low-level (how neurons compute) is key to understanding the high-level (intelligence and cognition)

- *Intelligence is an emergent phenomenon.* Complex behavior can emerge from a very large number of simple, interactive computations.

- *Simulation is central.* It's hard to predict how complexity will emerge. Computational modeling and simulation are essential for understanding intelligence.