So far...

So far...

- **Units**: ions, conductance, membrane potential, firing.

So far...

- **Units**: ions, conductance, membrane potential, firing.
- **Networks**: transformations, amplifications, attractors, basic building blocks of cognition.

So far...

- **Units**: ions, conductance, membrane potential, firing.
- **Networks**: transformations, amplifications, attractors, basic building blocks of cognition.

How do networks ever come to do interesting things?

- **Learning**

Learning

Learning

- Tuning detectors to achieve global results.

- *Learning rules*:
  How to adjust weights based only on *local information* (presynaptic and postsynaptic activity) to produce appropriate network behavior.

Learning

- Tuning detectors to achieve global results.

- *Learning rules*:
  How to adjust weights based only on *local information* (presynaptic and postsynaptic activity) to produce appropriate network behavior.

- Two main types:
  - *"Self organizing"*: learning internal **model** (statistics) of environment

Learning

- Tuning detectors to achieve global results.

- *Learning rules*:
  How to adjust weights based only on *local information* (presynaptic and postsynaptic activity) to produce appropriate network behavior.

- Two main types:
  - *"Self organizing"*: learning internal **model** (statistics) of environment
  - *"Error-driven":* Learning to solve a **task** (produce output from input)

Learning

- Tuning detectors to achieve global results.

- *Learning rules*:
  How to adjust weights based only on *local information* (presynaptic and postsynaptic activity) to produce appropriate network behavior.

- Two main types:
  - *"Self organizing"*: learning internal **model** (statistics) of environment
  - *"Error-driven":* Learning to solve a **task** (produce output from input)
  - Doing both at the same time

Learning

- Tuning detectors to achieve global results.

- *Learning rules*:
  How to adjust weights based only on *local information* (presynaptic and postsynaptic activity) to produce appropriate network behavior.

- Two main types:
  - *"Self organizing"*: learning internal **model** (statistics) of environment
  - *"Error-driven":* Learning to solve a **task** (produce output from input)
  - Doing both at the same time

What is self-organizing / statistical Learning?

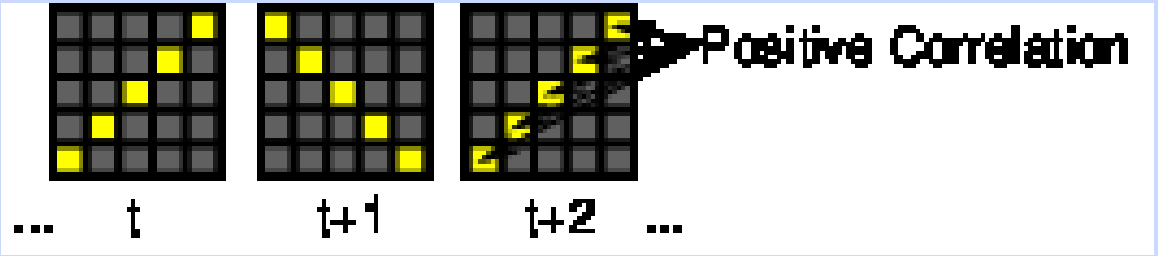
- "Things" in the world have relatively stable sets of features.
- How do detectors in our brains come to detect these things?
- The features of a particular thing tend to appear together and disappear together
- A thing is nothing more than a *correlated* cluster of features
- Learning mechanisms sensitive to correlation will lead to representation of useful things

statistical learning

Pick up on correlations in the world.

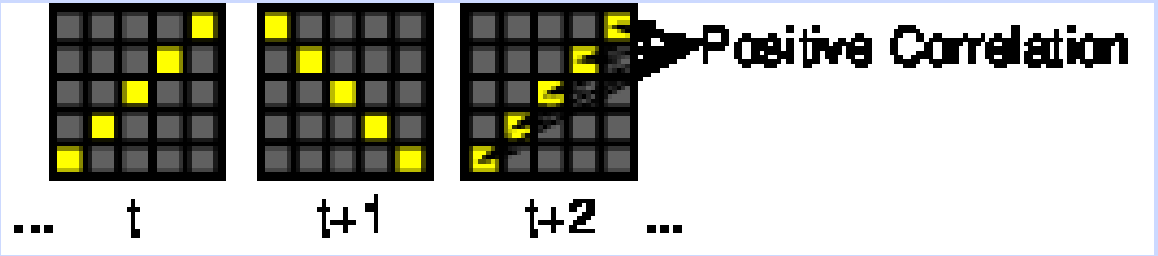
statistical learning

Pick up on correlations in the world.



statistical learning

Pick up on correlations in the world.



(whether for pixels in visual images, emotions and people, behaviors, etc.)

Biology: Associative/Hebbian LTP/D

Biology suggests associative or "Hebbian" learning:
proposed by Donald Hebb.

Biology: Associative/Hebbian LTP/D

Biology suggests associative or "Hebbian" learning:
proposed by Donald Hebb.
"Units that fire together wire together!";
"Units that fire out of sync, lose their link"

Biology: Associative/Hebbian LTP/D

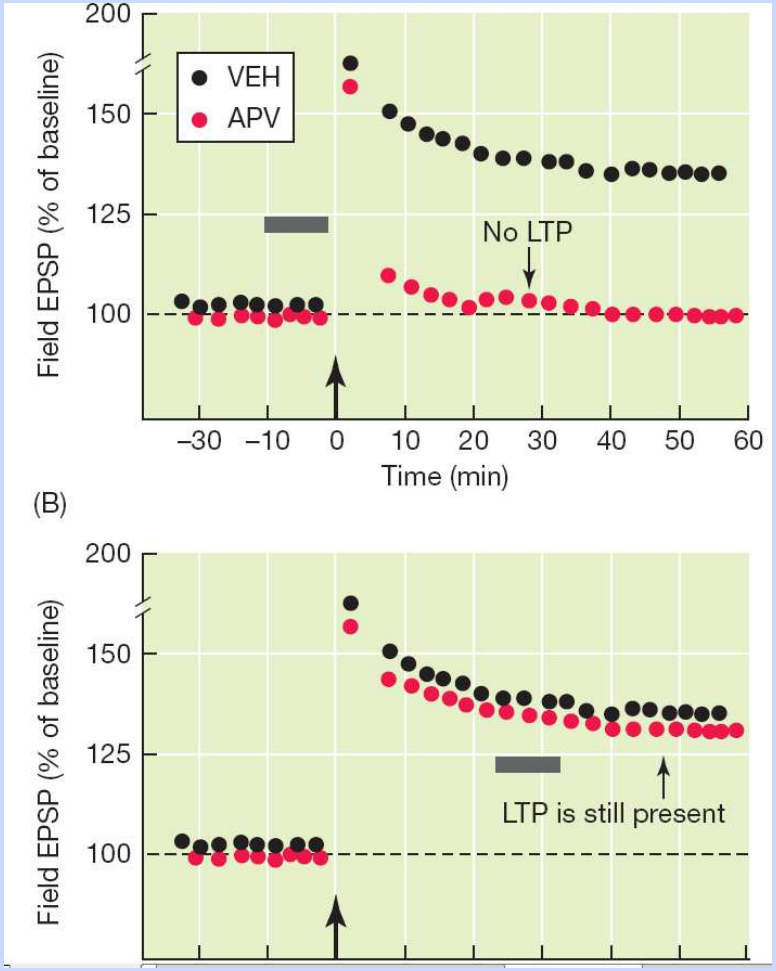
Biology suggests associative or "Hebbian" learning:
proposed by Donald Hebb.
"Units that fire together wire together!";
"Units that fire out of sync, lose their link"

Synaptic efficacy (weights) change when neurons are excited:

- Going up: Long Term Potentiation (LTP)
  (*strengthens connections between co-active units*)

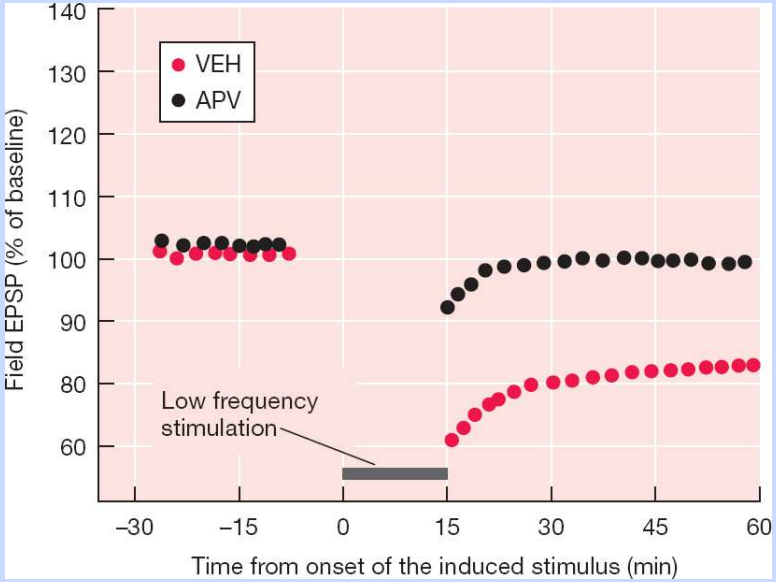
- Going down: Long Term Depression (LTD)
  (*weakens connections between units that are not co-active*)

LTP is dependent on NMDA receptor activation



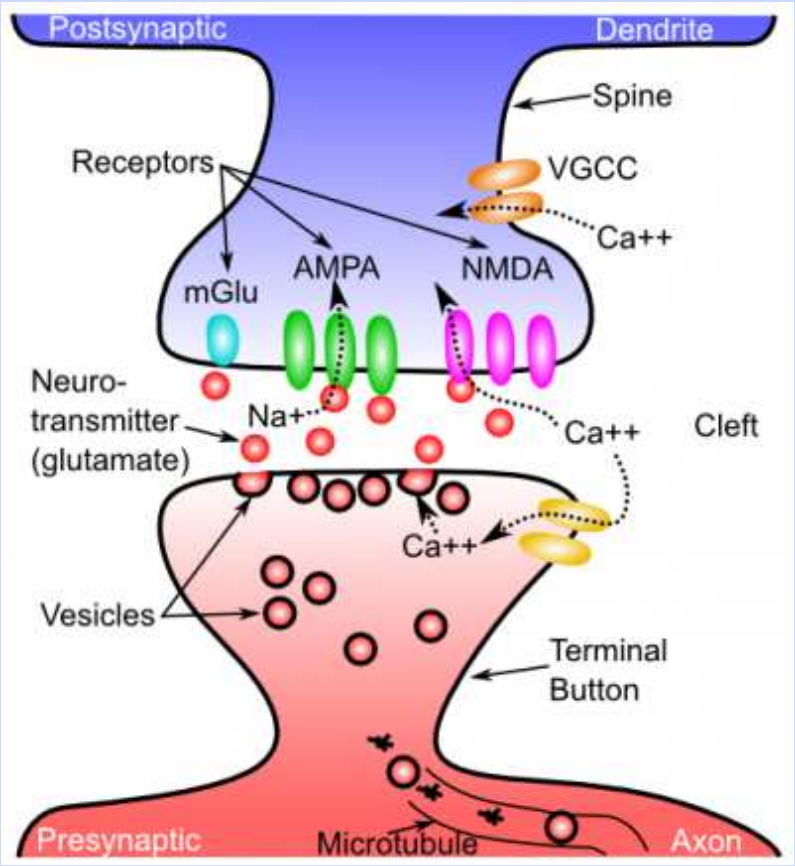
APV = NMDA antagonist

And so is LTD

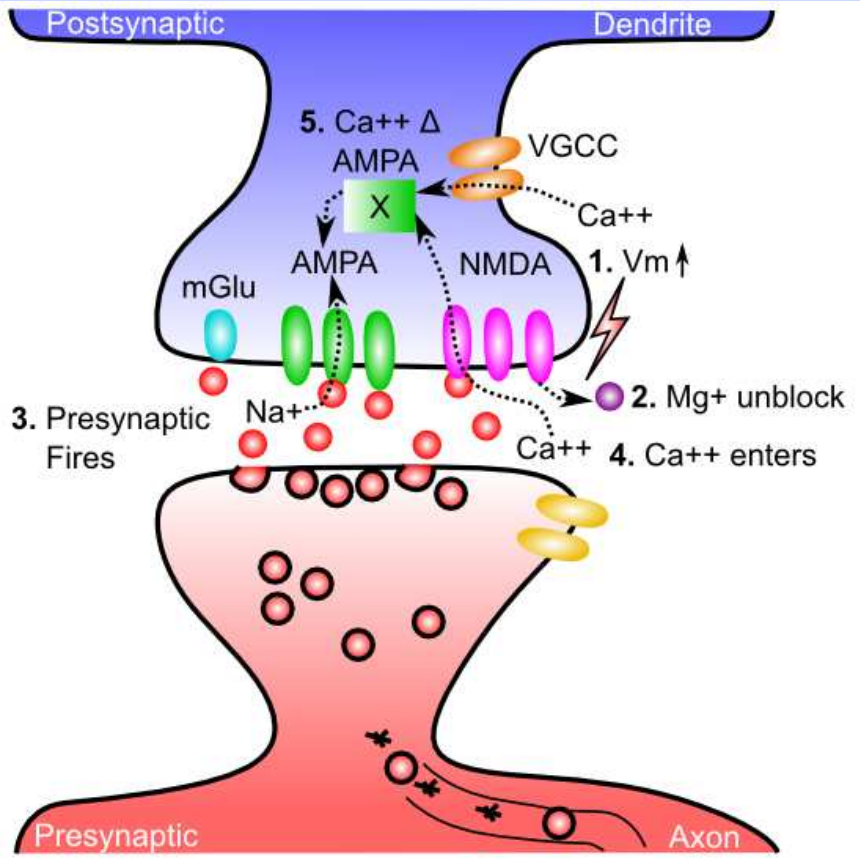


APV = NMDA antagonist

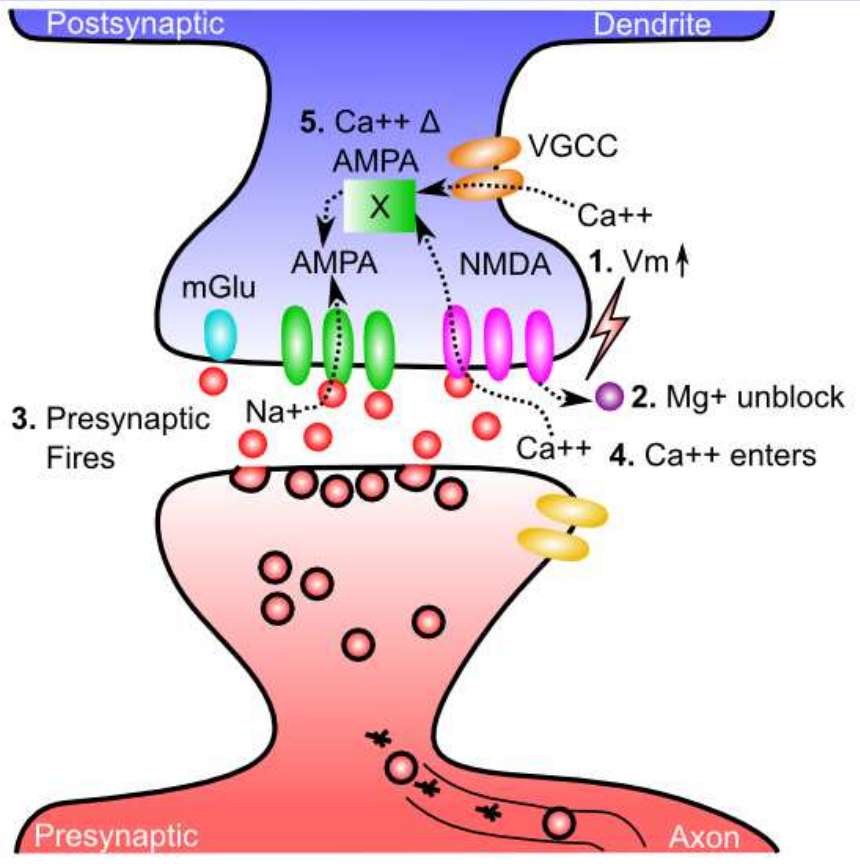
What changes in the synapse to support this learning?



NMDA = Associativity: Both Pre & Post Active

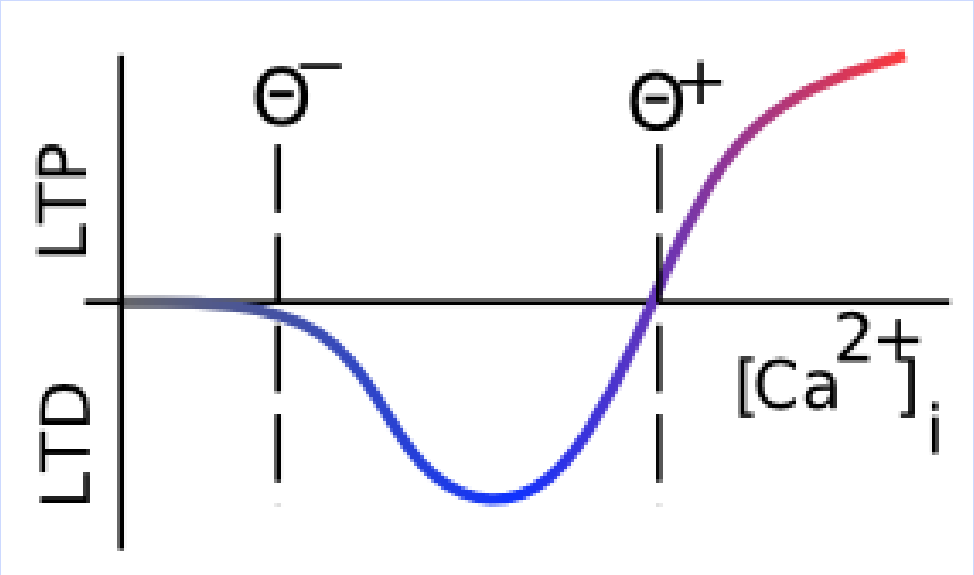


NMDA = Associativity: Both Pre & Post Active



"Gettin AMPA'd": chem processes → new AMPA receptors (or trafficking of existing ones to membrane)
AMPAR's open Na channels → increase excitability
NMDAR's: learning via Ca.

Biology: NMDA-mediated LTP/D



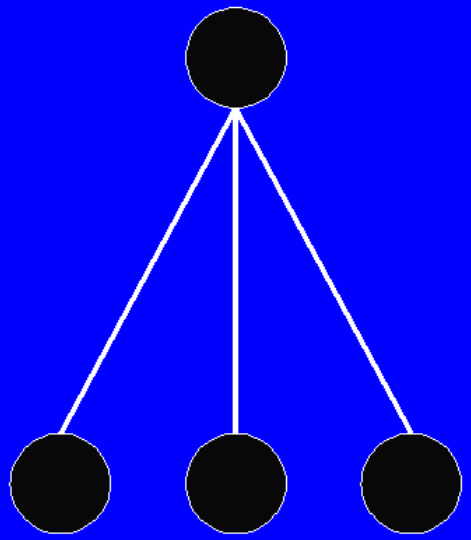
Strong activity (Ca^{++}) = LTP, weak = LTD

For details of mechanisms of LTP/LTD:

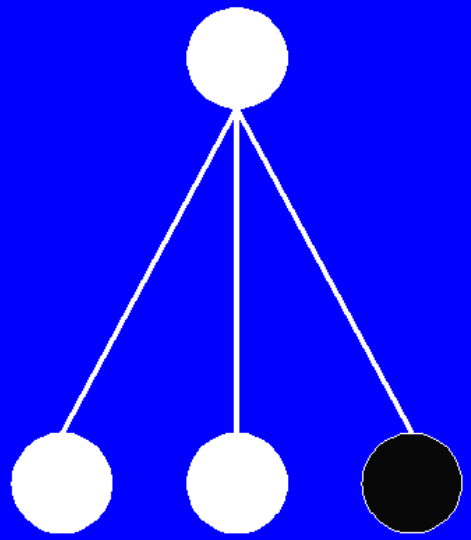
THE NEUROBIOLOGY OF LEARNING AND MEMORY
Jerry W Rudy Second Edition (2014)

Very clear and well-written!

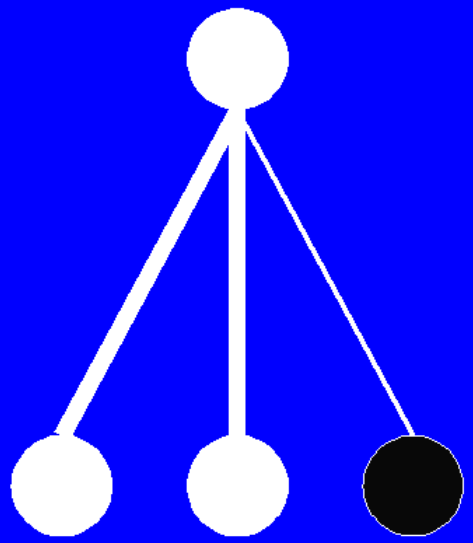
Hebbian Learning



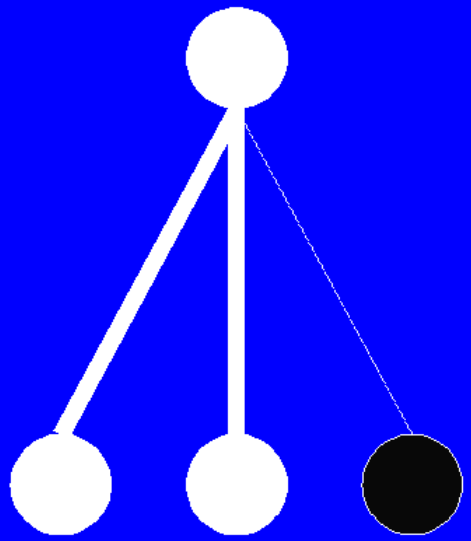
Hebbian Learning



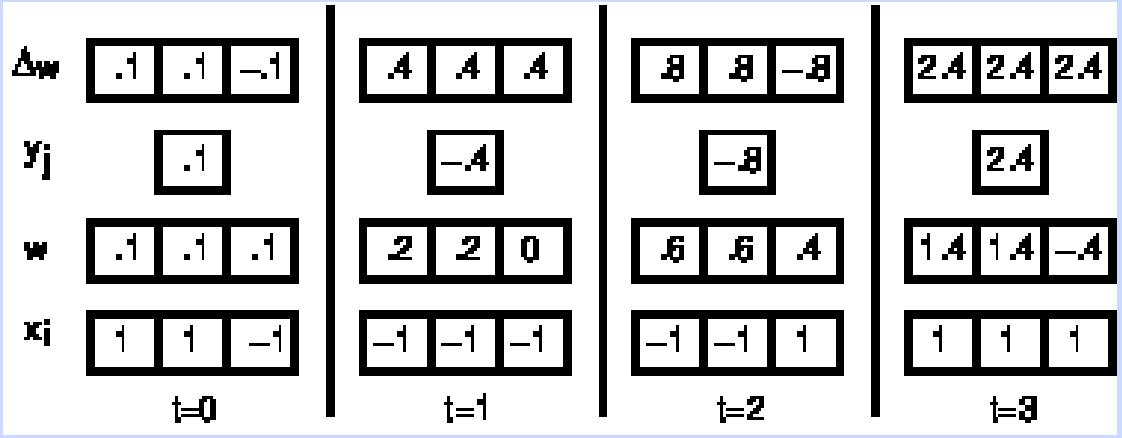
Hebbian Learning



Hebbian Learning



Hebb = PCA demo



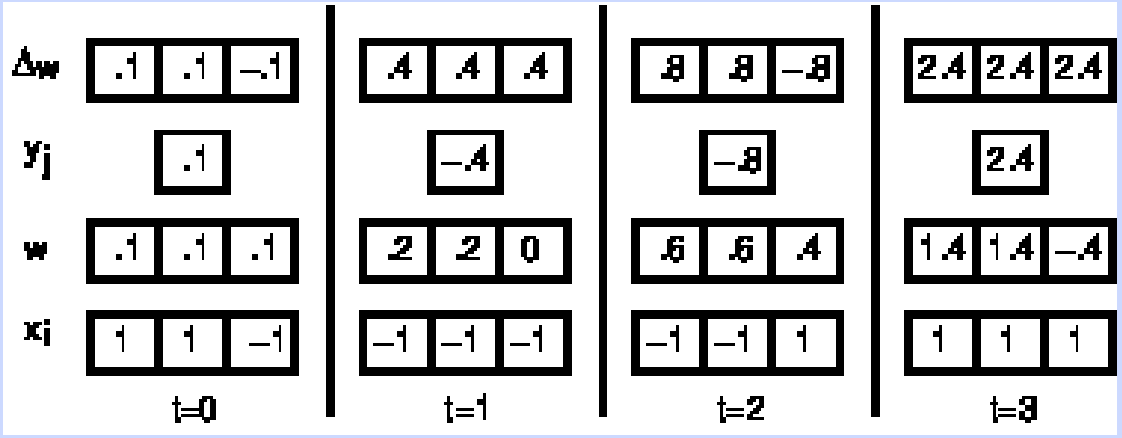
Ľinear Activation:

$$y_j = \sum_i x_i w_{ij} \tag{1}$$

Simple Hebb rule:

$$\Delta_t w_{ij} = \epsilon x_i y_j \tag{2}$$

Hebb = PCA demo



Linear Activation:

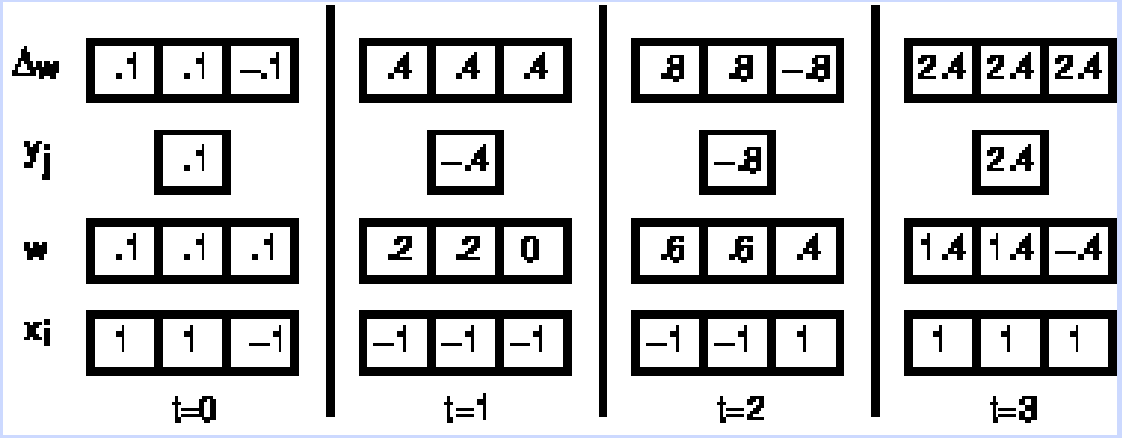
$$y_j = \sum_i x_i w_{ij} \tag{1}$$

Simple Hebb rule:

$$\Delta_t w_{ij} = \epsilon x_i y_j \tag{2}$$

Wts get stronger for two units that are correlated, but not for uncorrelated unit!

Hebb = PCA demo



Ľinear Activation:

$$y_j = \sum_i x_i w_{ij} \tag{1}$$

Simple Hebb rule:

$$\Delta_t w_{ij} = \epsilon x_i y_j \tag{2}$$

Wts get stronger for two units that are correlated, but not for uncorrelated unit!

Technically: wts evolve toward principal eigenvector of correlation matrix among inputs

Simple Hebb Rule

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

$$\Delta w_{ij} = \epsilon x_i y_j$$

ϵ = learning rate

x_i = act of sending unit i

y_j = act of receiving unit j

Simple Hebb Rule

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

$$\Delta w_{ij} = \epsilon x_i y_j$$

ϵ = learning rate

x_i = act of sending unit i

y_j = act of receiving unit j

Problem: Weights will grow infinitely large.

Simple Hebb Rule

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

$$\Delta w_{ij} = \epsilon x_i y_j$$

ϵ = learning rate

x_i = act of sending unit i

y_j = act of receiving unit j

Problem: Weights will grow infinitely large.

$\rightarrow$ Unrealistic *and* computationally bad!

Simple Hebb Rule

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

$$\Delta w_{ij} = \epsilon x_i y_j$$

ϵ = learning rate

x_i = act of sending unit i

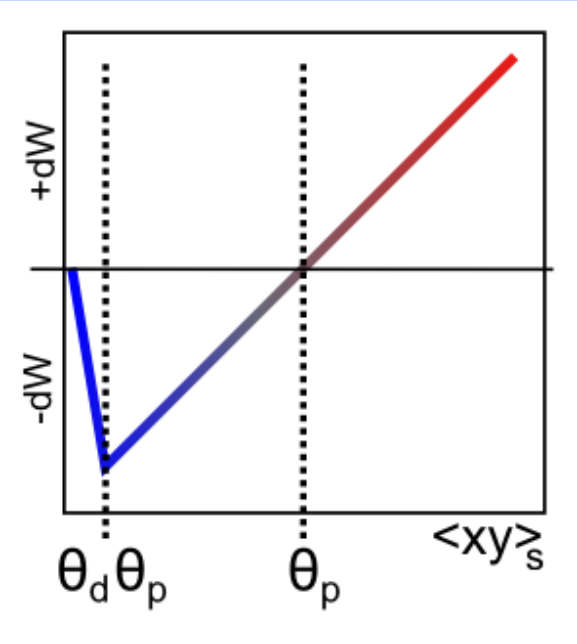
y_j = act of receiving unit j

Problem: Weights will grow infinitely large.

$\rightarrow$ Unrealistic *and* computationally bad!

$\rightarrow$ Need homeostatic mechanism to balance weight updates and allow for LTD...

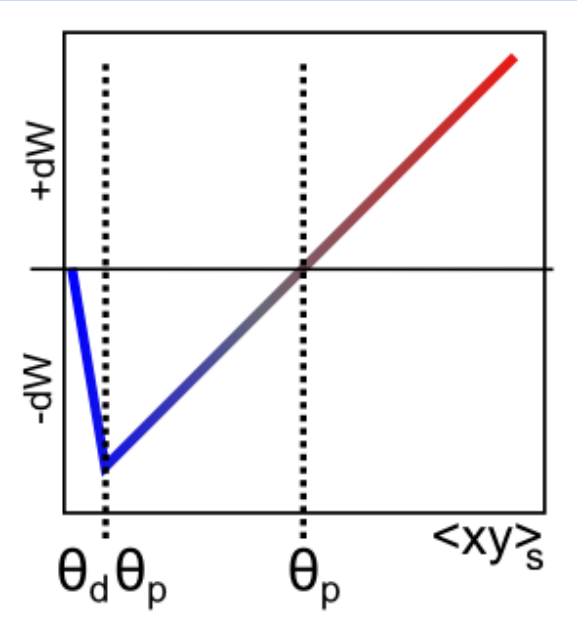
Adapted Hebb learning rule: "XCAL"



$$\Delta w = \epsilon f_{xcal}(xy, \theta_p) = \epsilon \begin{cases} (xy - \theta_p) & if xy > \theta_p \theta_d \\ -xy(1 - \theta_d)/\theta_d & otherwise \end{cases}$$

θ_d : fixed threshold determining when weight change reverses direction
θ_p : *adaptive* threshold determining when weight change reverses sign

Adapted Hebb learning rule: "XCAL"

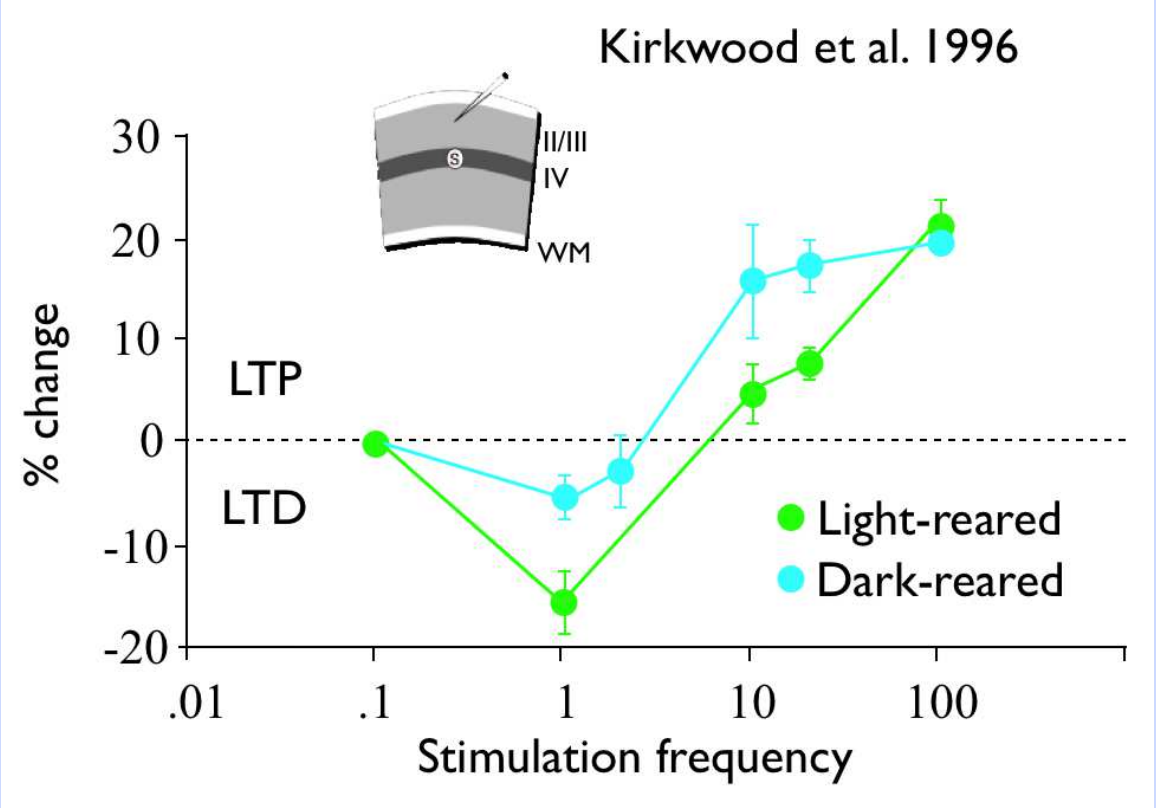


$$\Delta w = \epsilon f_{xcal}(xy, \theta_p) = \epsilon \begin{cases} (xy - \theta_p) & if xy > \theta_p \theta_d \\ -xy(1 - \theta_d)/\theta_d & otherwise \end{cases}$$

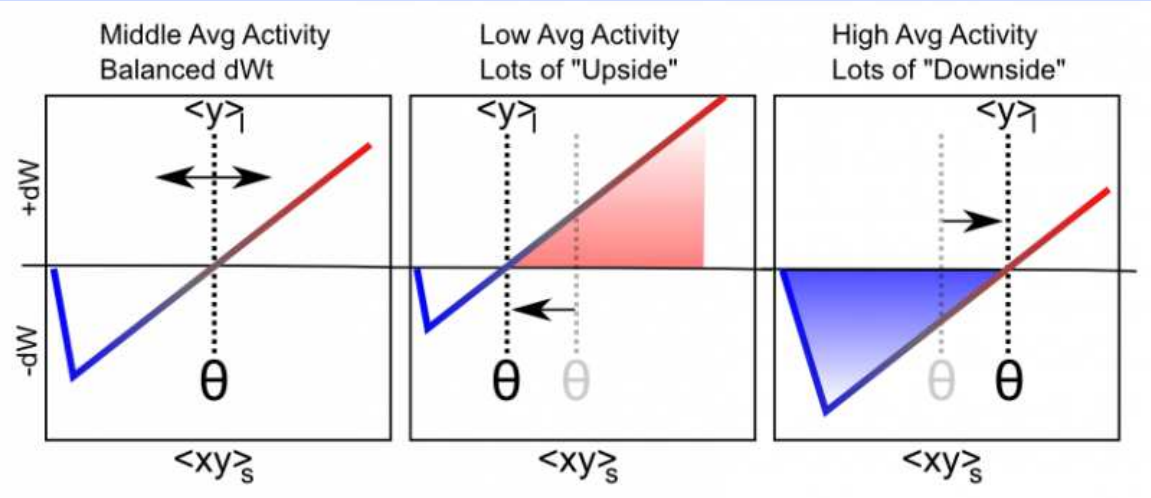
θ_d : fixed threshold determining when weight change reverses direction
θ_p : *adaptive* threshold determining when weight change reverses sign

$\theta_p \propto$ prior activity: homeostatic mechanism – "BCM rule"

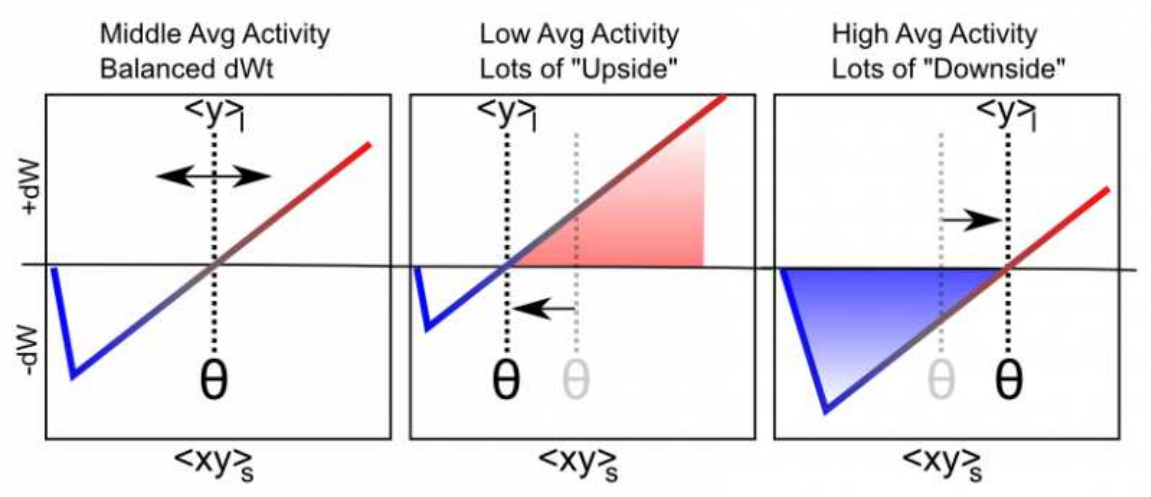
Some evidence for a 'floating threshold'



Self Organizing Learning:
Set adaptive threshold θ_p = long term average activity y_l



Self Organizing Learning:
Set adaptive threshold θ_p = long term average activity y_l



long term activity is kept track in simulator:

$$y_l = \begin{cases} y_l + \tau_+(y - y_l) & if y > y_l \\ y_l + \tau_-(y - y_l) & otherwise \end{cases}$$

$\tau_+ > \tau_-$ so that excessive activity levels are quickly penalized.

XCAL is just another Hebb learning variant

$$\Delta w_{ij} \approx \epsilon x_s (y_s - y_l)$$

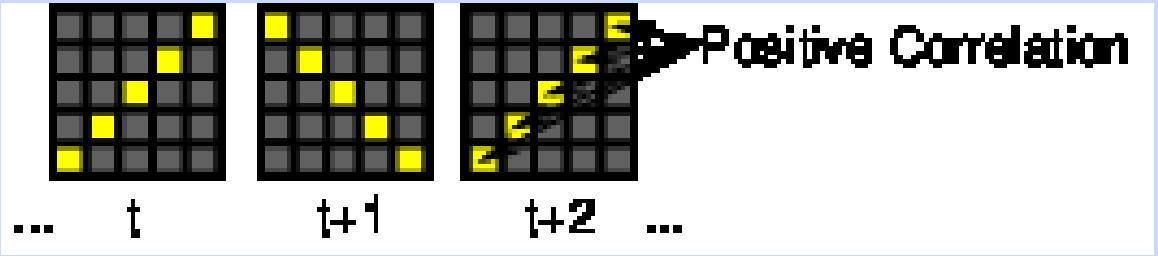
- In this variant, learning occurs in proportion to Hebbian co-product as usual, but activation of postsynaptic cell is *relative* to its long-term value y_l. This allows for LTP to reverse to LTD for low xy (low Ca^{2+}).

- x_s and y_s are the activations of x and y integrated over short time scales (i.e., not just some instantaneous rate, which is implausible to drive learning).

- We'll see later that these values can be compared against more medium term time scales to produce other learning dynamics

Model learning

Pick up on correlations in the world.



(whether for pixels in visual images, emotions and people, behaviors, etc.)

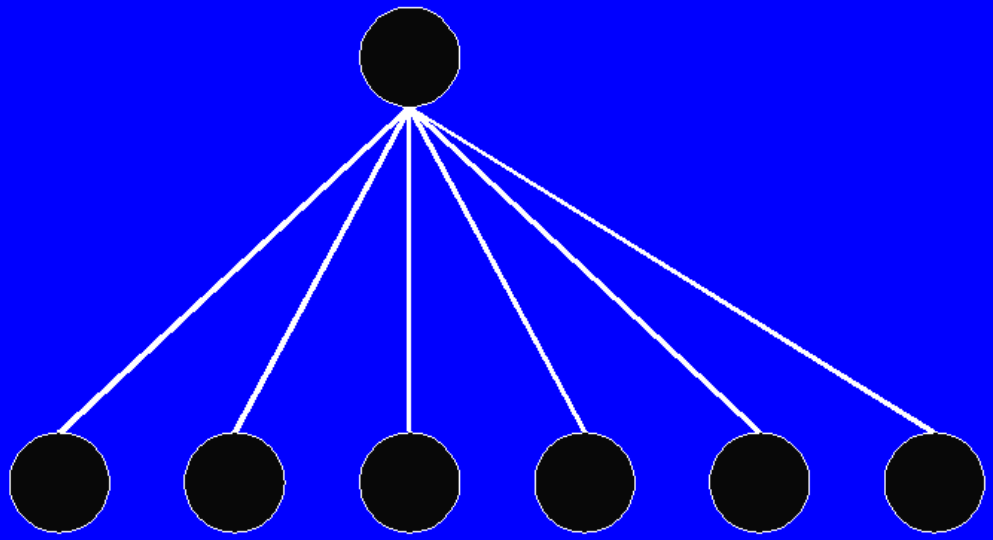
Based on Hebbian (LTP/LTD) mechanisms.

[hebb_correl.proj]

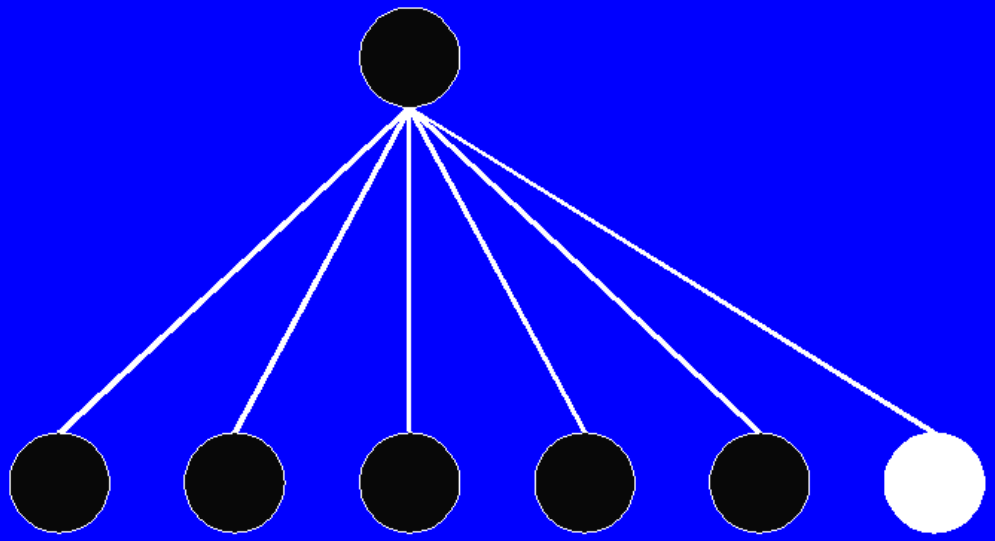
What does Hebbian Learning Do?

- Hebbian learning *tunes* units to represent correlated sets of input features
- If a unit has 1000 inputs, turning on and off a single input feature won't have a big effect on the unit's activity
- In contrast, turning on and off a large *cluster* of 900 input features will have a big effect on that unit's activity
- Learning increases with activity, so unit will be tuned to respond to correlated features

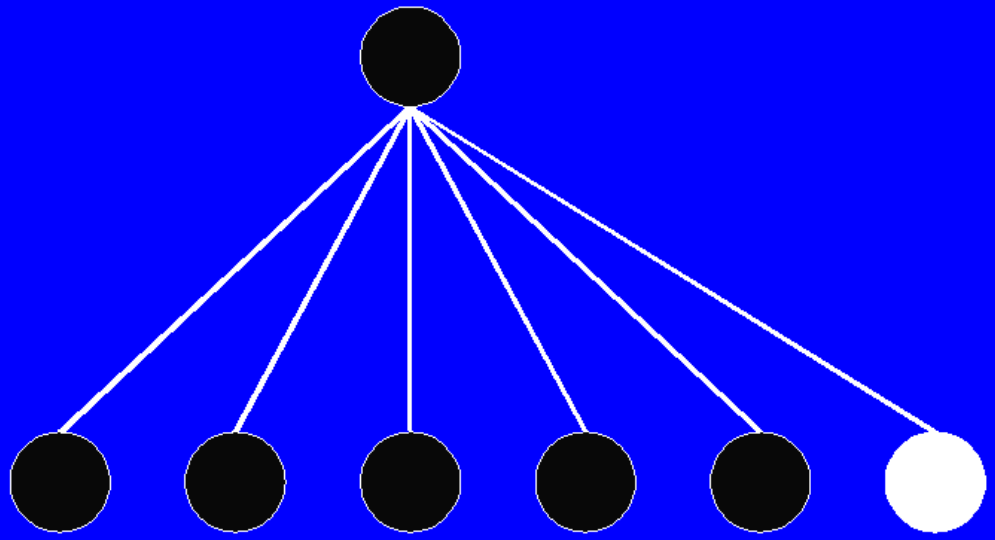
Hebbian Learning



Hebbian Learning

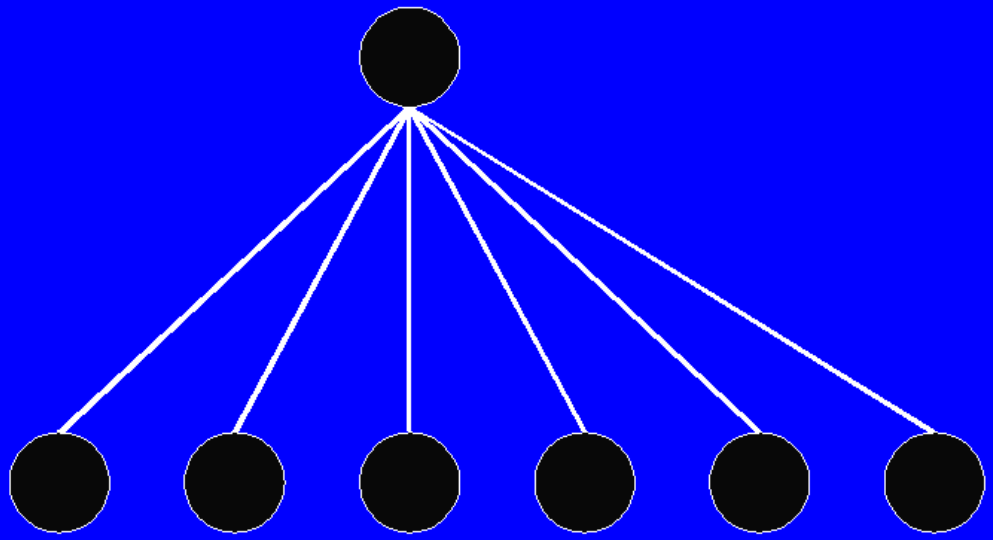


Hebbian Learning

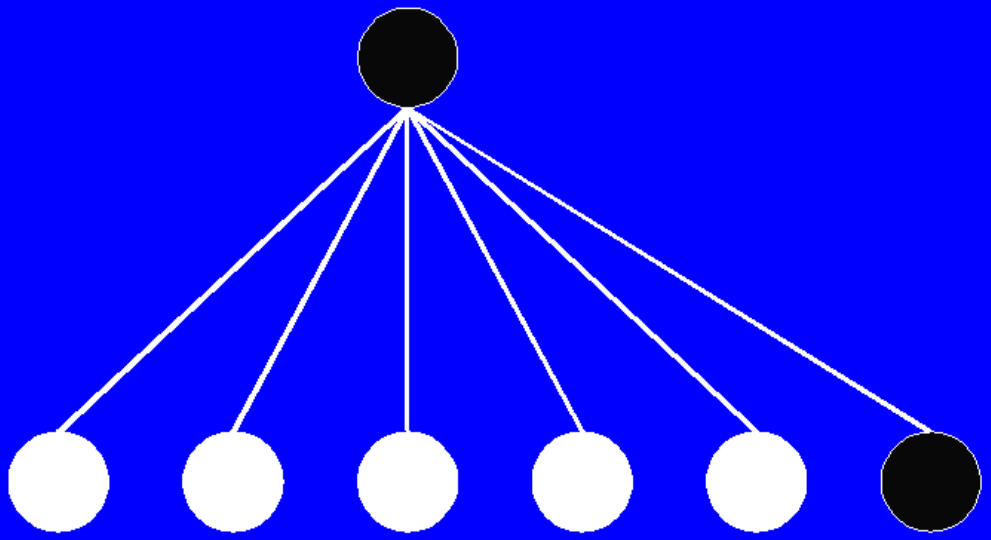


- Because small clusters of inputs do not reliably activate the receiving unit, the receiving unit does not **learn much** about these inputs

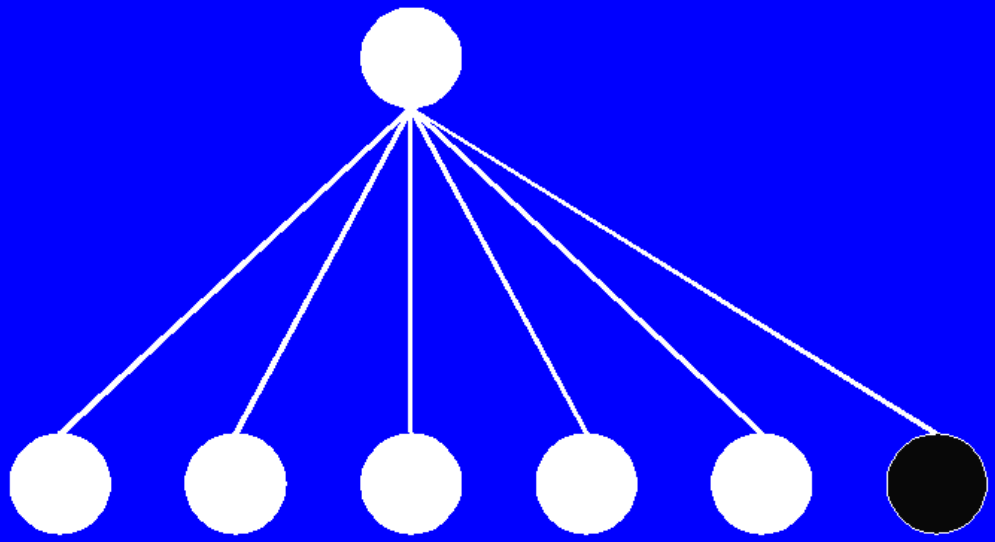
Hebbian Learning



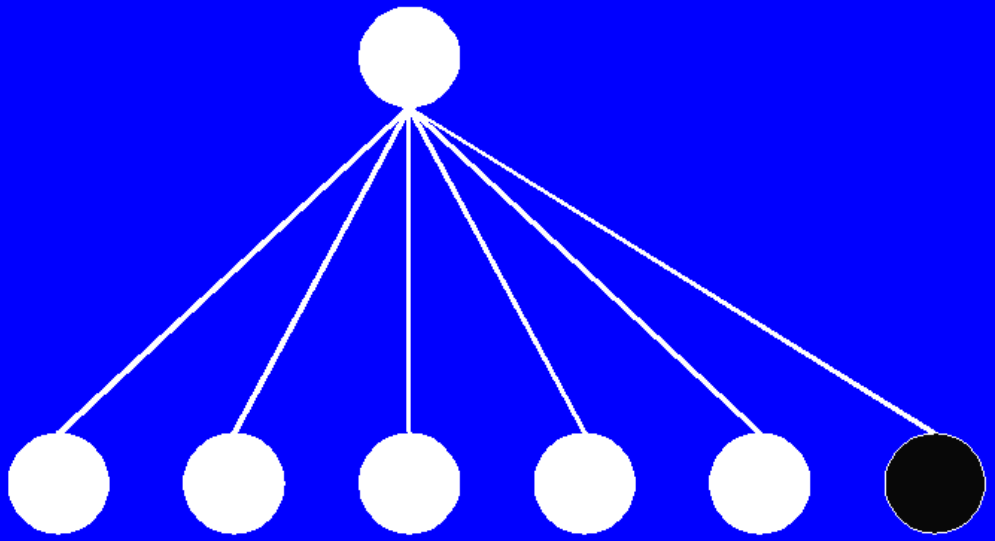
Hebbian Learning



Hebbian Learning

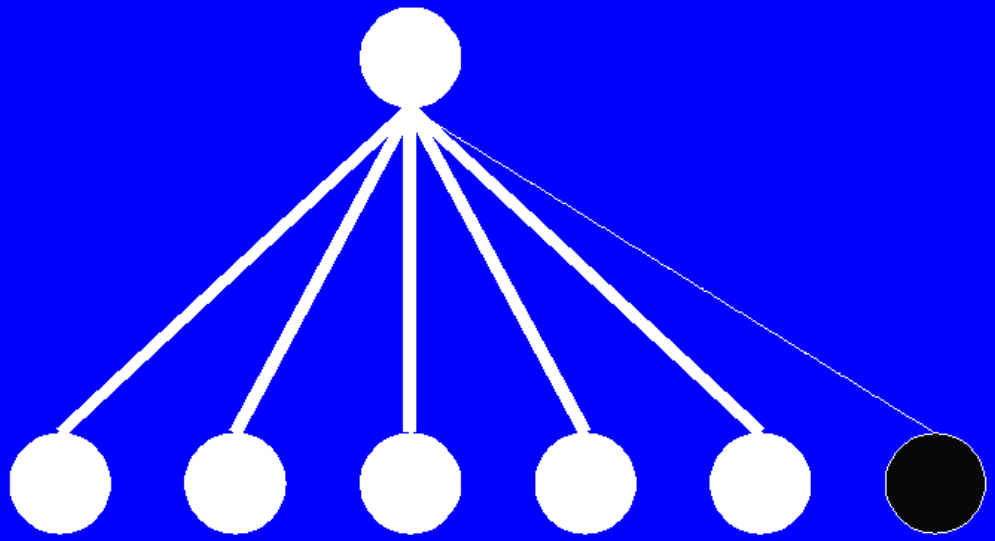


Hebbian Learning



Big clusters of inputs reliably activate the receiving unit, so the network **learns more** about big (vs. small) clusters (the “gang effect”).

Hebbian Learning



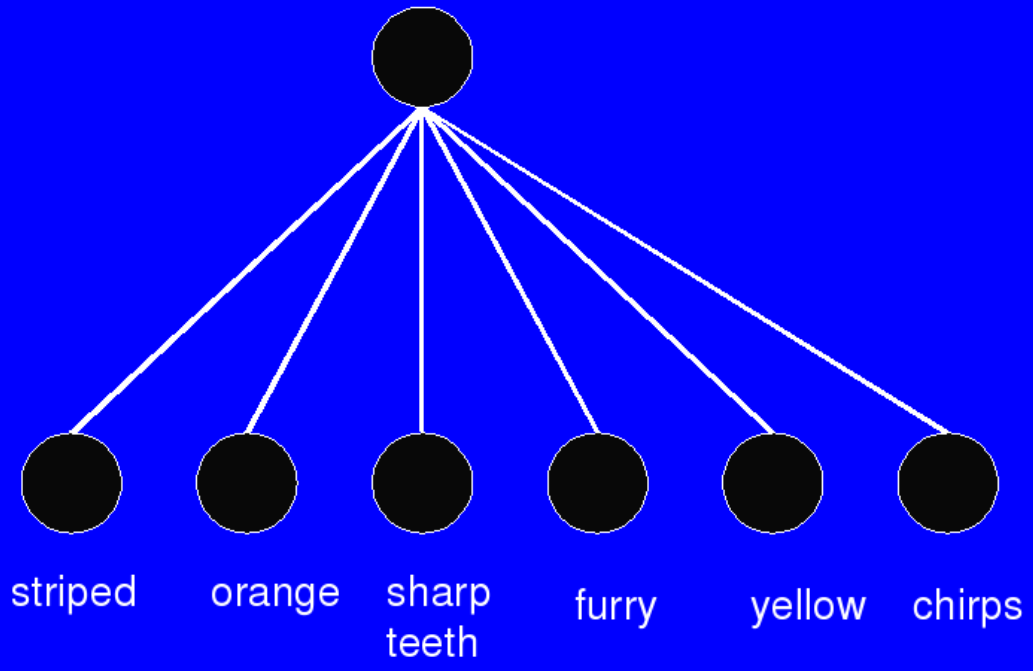
Big clusters of inputs reliably activate the receiving unit, so the network **learns more** about big (vs. small) clusters (the “gang effect”).

[newgang.proj]

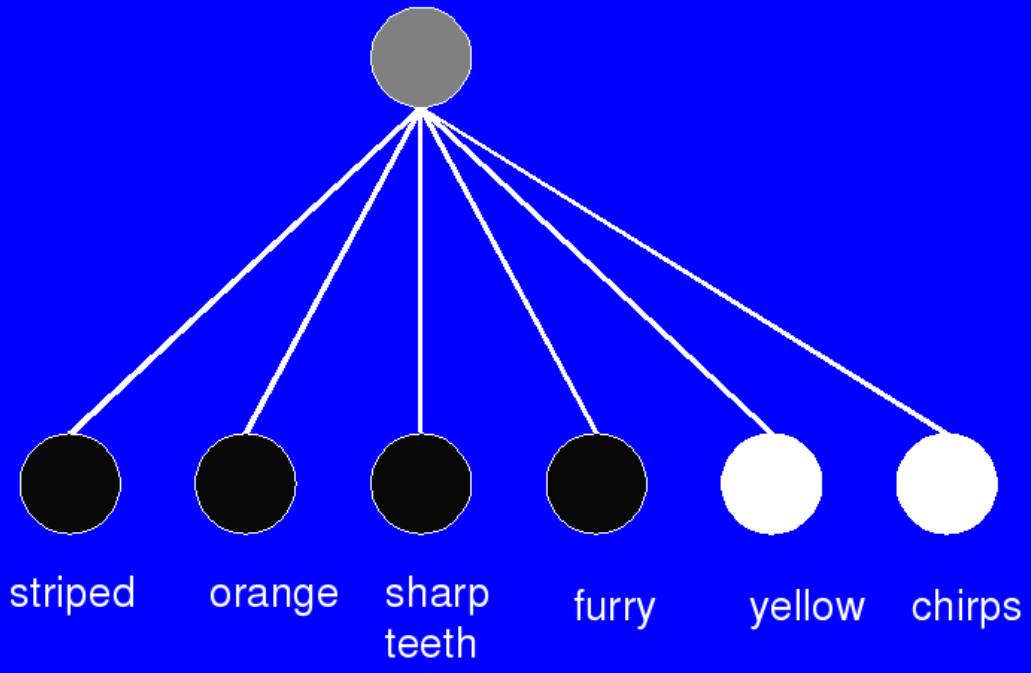
What does Hebbian Learning Do?

- Hebbian learning find the *thing in the world* that most reliably activates the unit, and *tunes* the unit to like that thing even more!

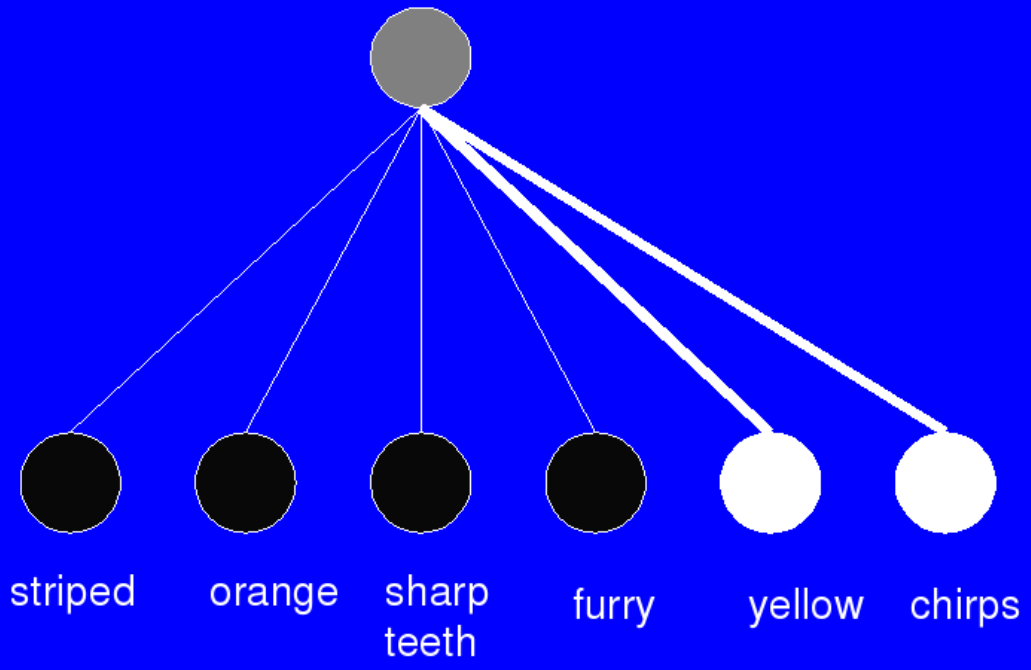
Hebbian Learning



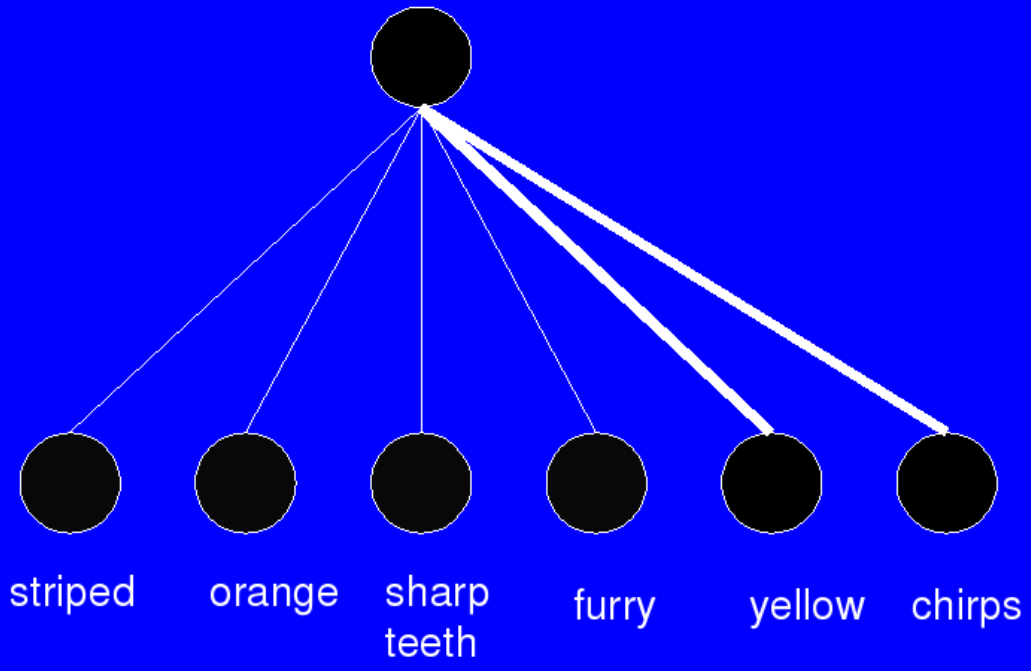
Hebbian Learning



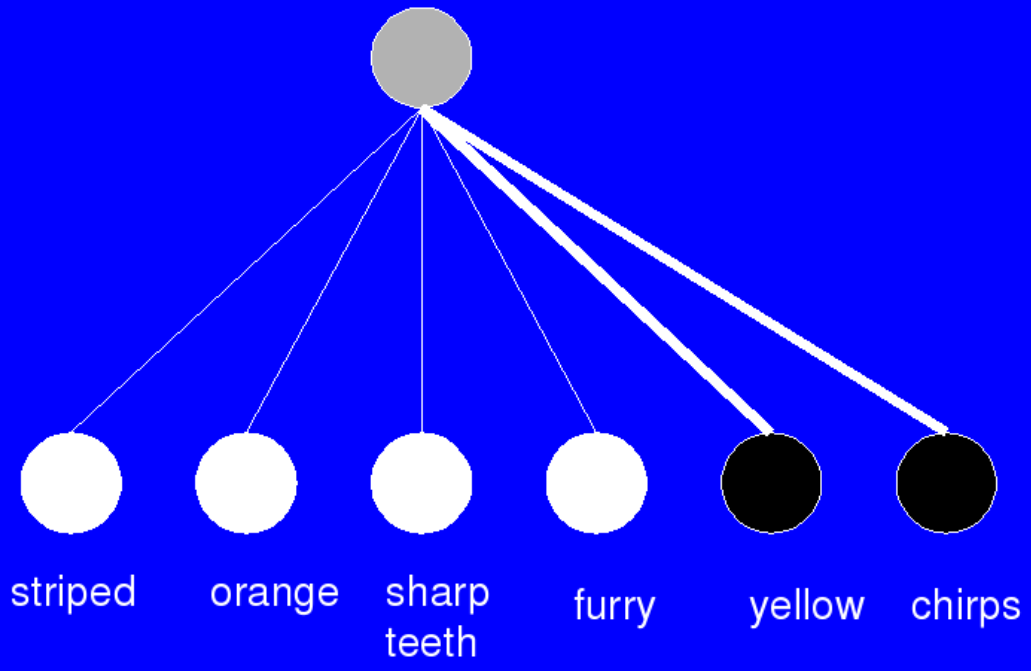
Hebbian Learning



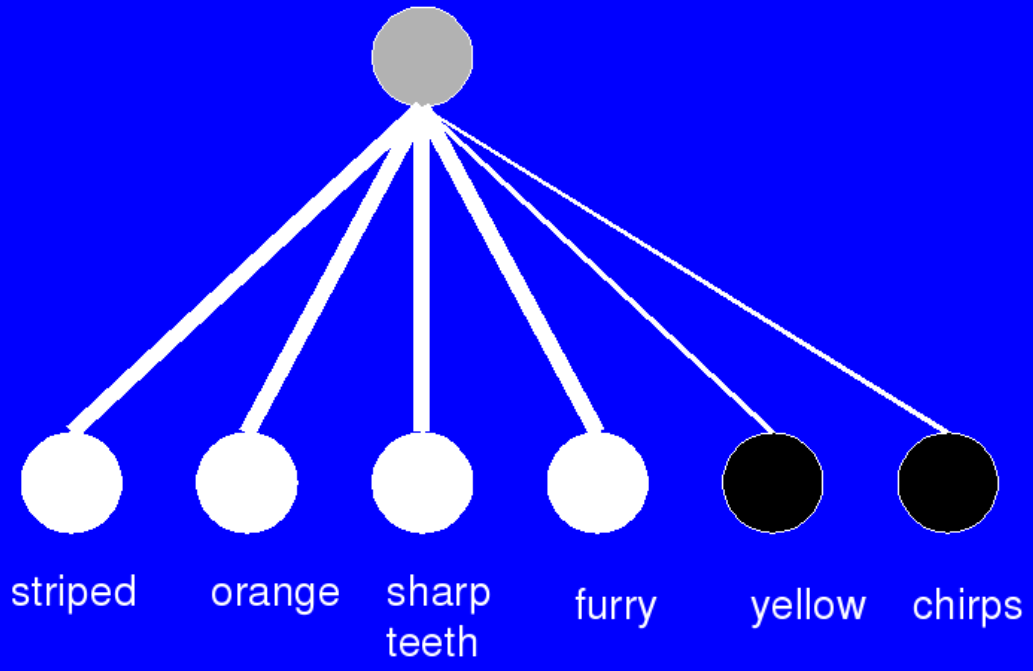
Hebbian Learning



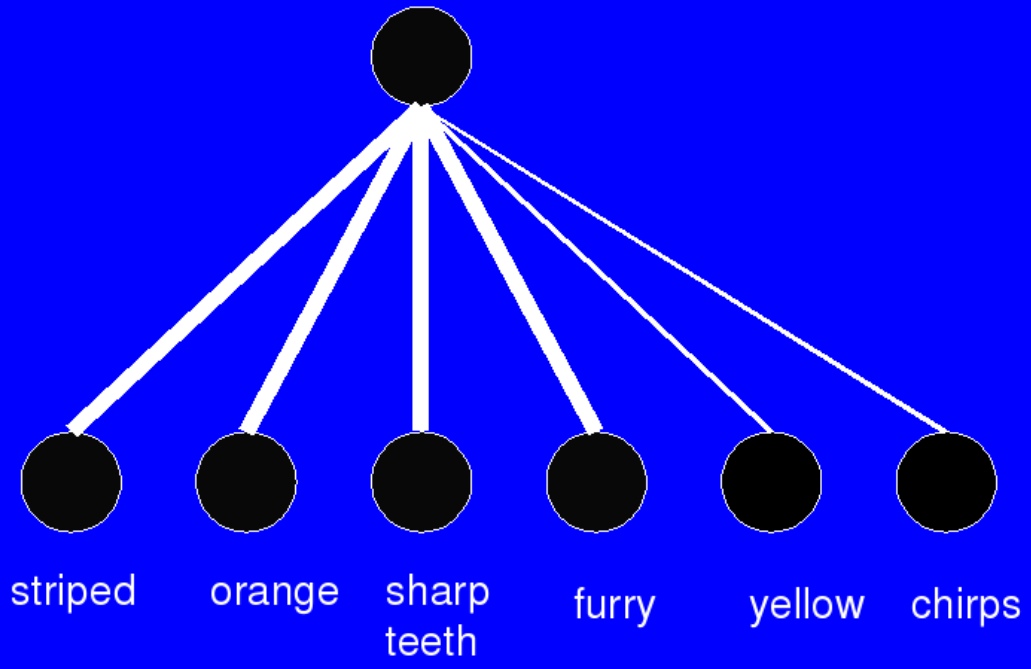
Hebbian Learning



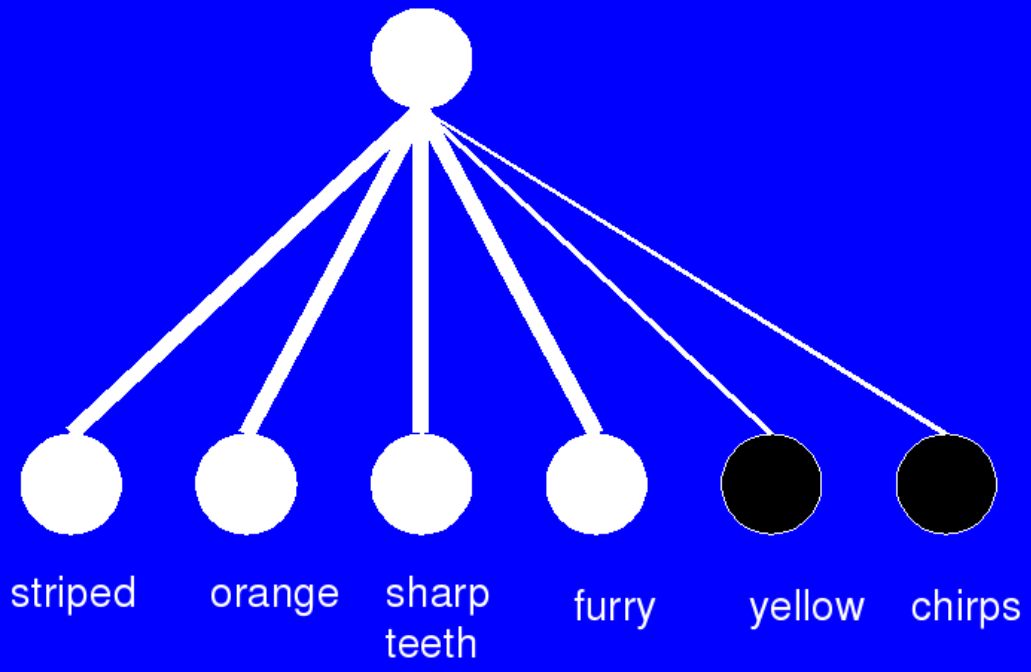
Hebbian Learning



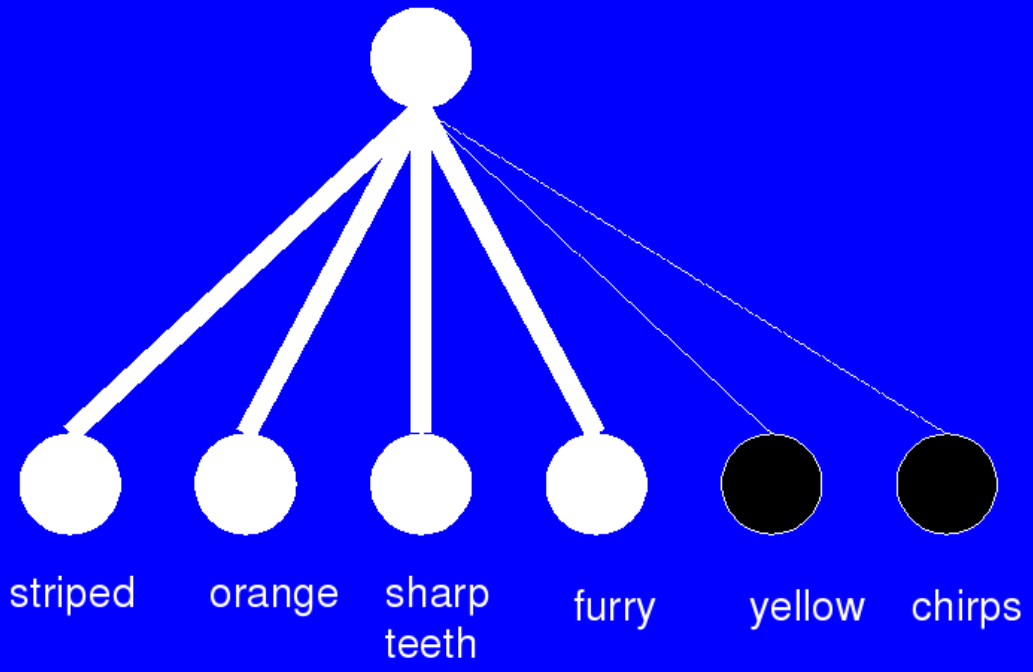
Hebbian Learning



Hebbian Learning



Hebbian Learning



What does Hebbian Learning Do?

- Hebbian learning find the *thing in the world* that most reliably activates the unit, and *tunes* the unit to like that thing even more!
- "thing in the world that most reliably activates the unit" = *principal component*
- function of *how well* an input activates the unit

Integrating over Experiences

- What a unit learns to represent is a function of how *excitable* the unit is
- Units that are activated (initially) by a wide range of stimuli end up representing an *average* of all of these stimuli
  - Imagine a unit that is activated by every stimulus; in this case, it learns a little about every one of these stimuli
  - If all of the input patterns have something in common, then the unit will learn what they have in common
  - There are very few meaningful things that are present in *all* inputs, if you try to average over too large a set of stimuli then you get mush

Integrating over Experiences

- Units that are activated (initially) by a wide range of stimuli end up representing an *average* of all of these stimuli
- Units that are activated (initially) by a narrow range of stimuli end up representing very *specific* features
  - If a unit is only activated by a single stimulus, it will only represent that stimulus
- Thus the learning rule allows you to represent concepts at varying levels of abstraction, depending on how excitable the unit is

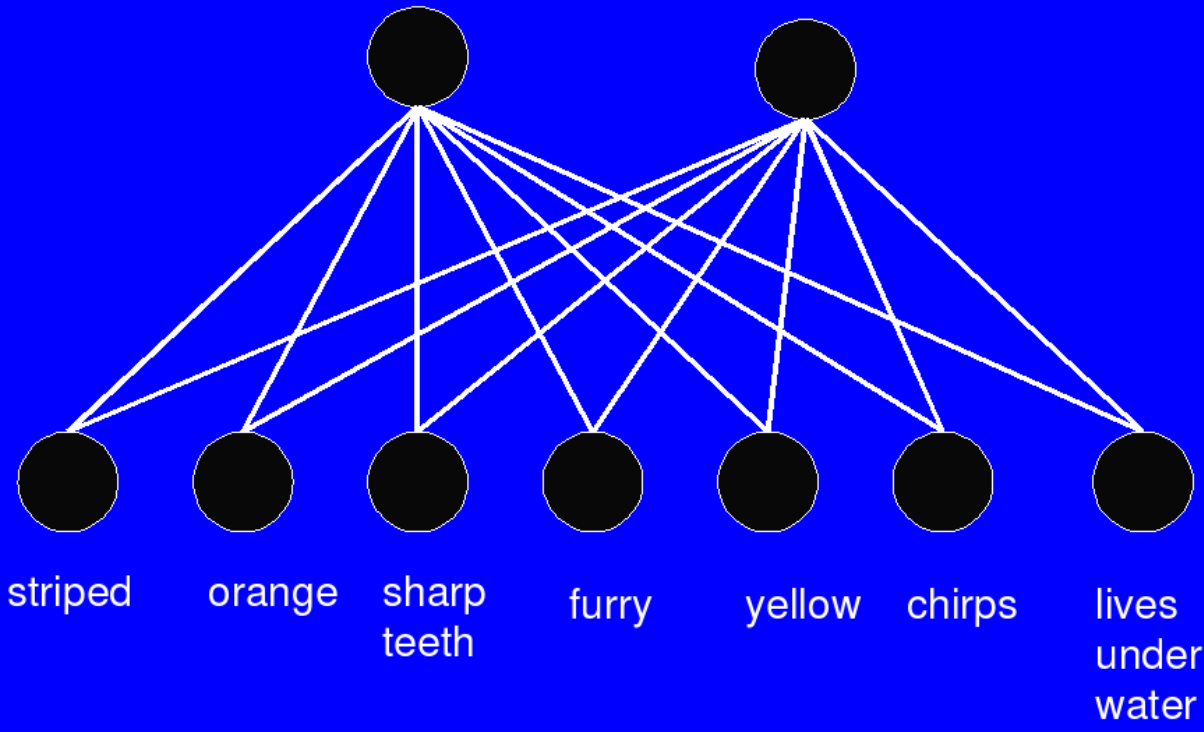
Multiple Units

- One detector can only represent one "thing" (pattern of correlated features)
- *Goal:* We want to have different units in the network learn to *"specialize"* for different things, such that each thing is represented by at least one unit
- Random initial weights and inhibitory competition are important for achieving this goal

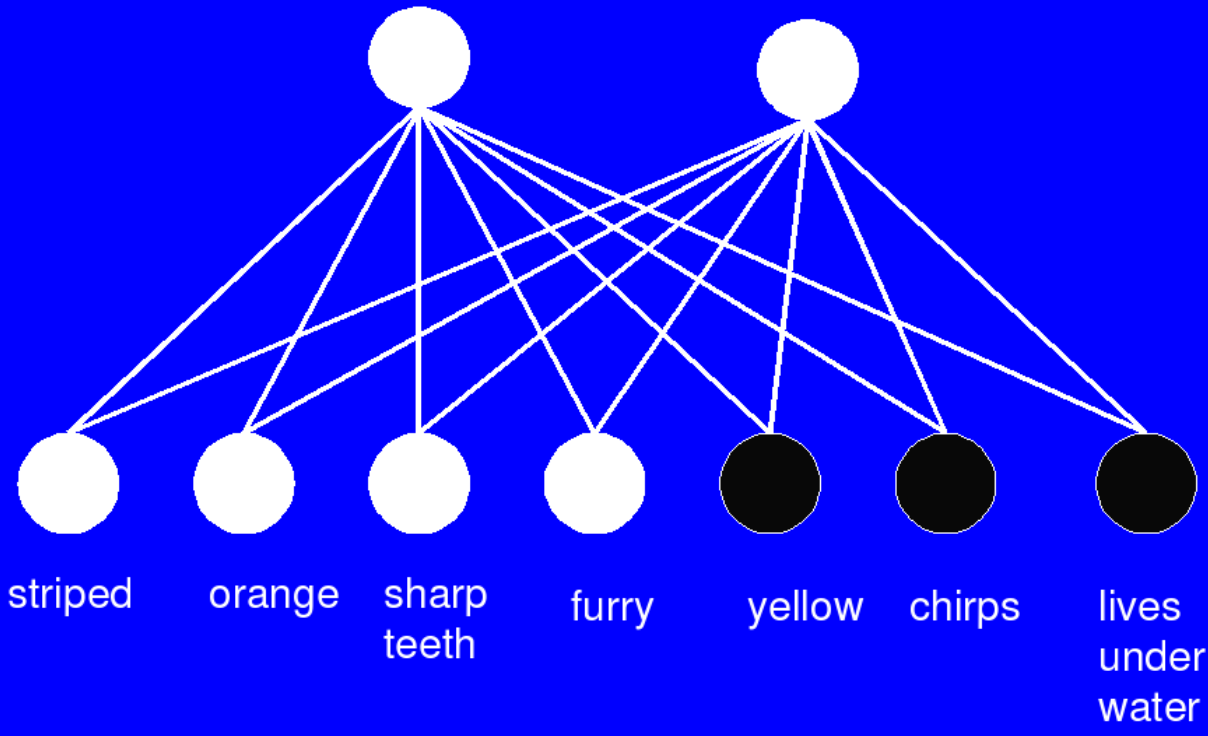
Multiple Units

- One detector can only represent one "thing" (pattern of correlated features)
- *Goal:* We want to have different units in the network learn to *"specialize"* for different things, such that each thing is represented by at least one unit
- Random initial weights and inhibitory competition are important for achieving this goal
- What happens when different units have the same initial weights and no competition...

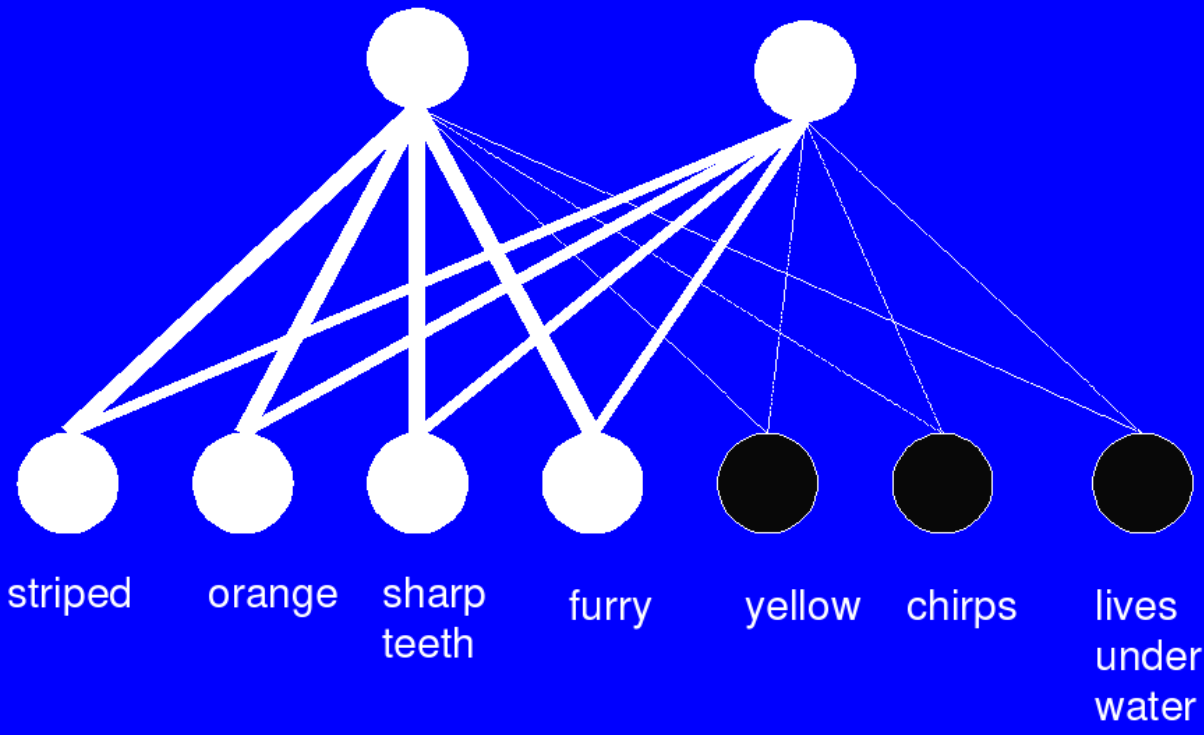
No Competition



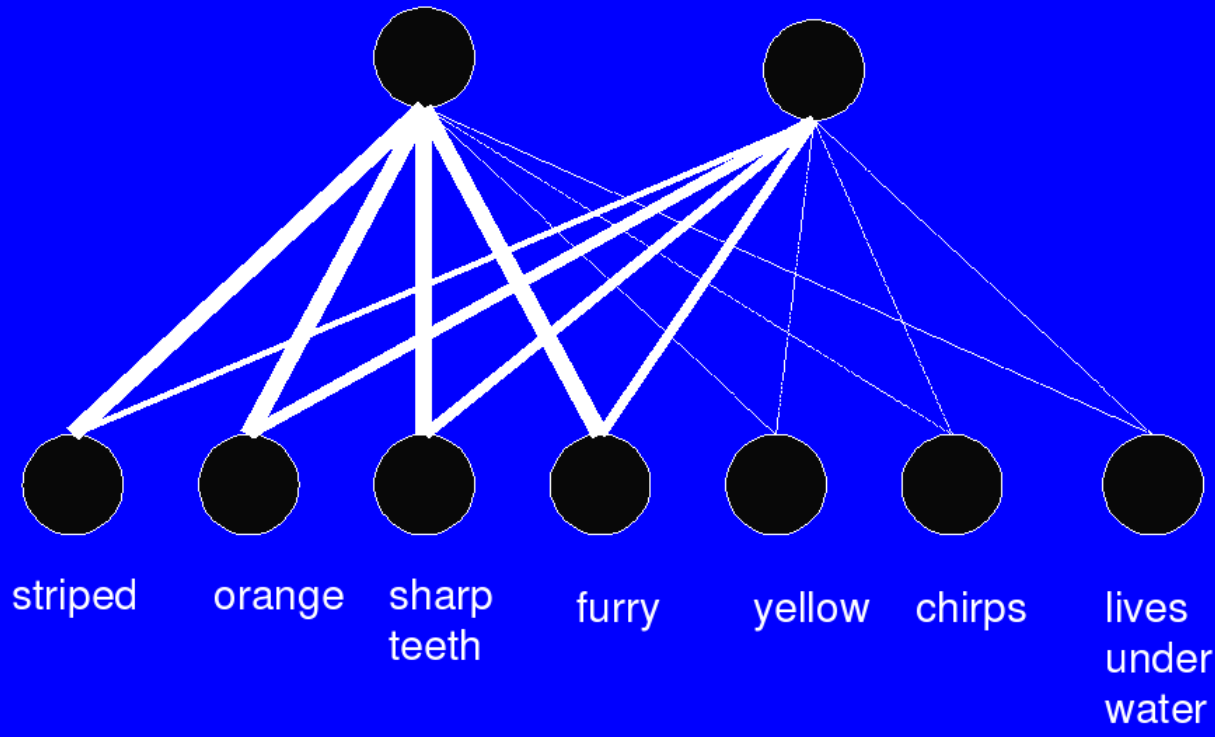
No Competition



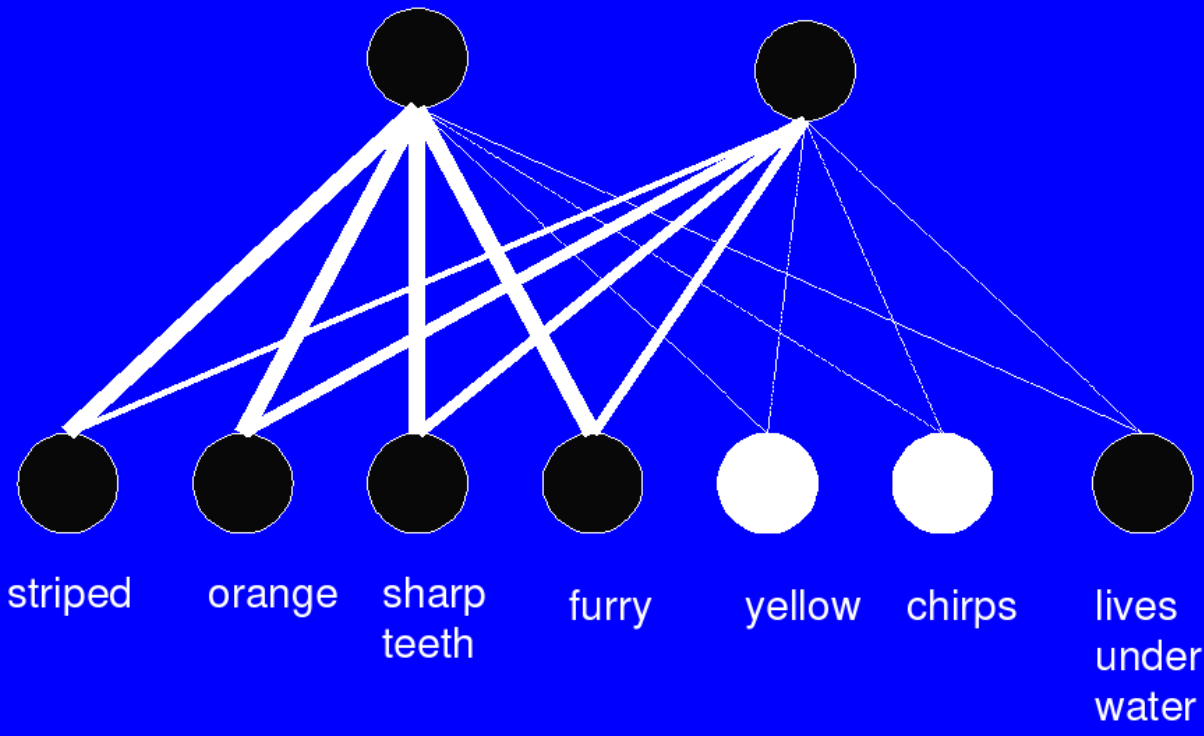
No Competition



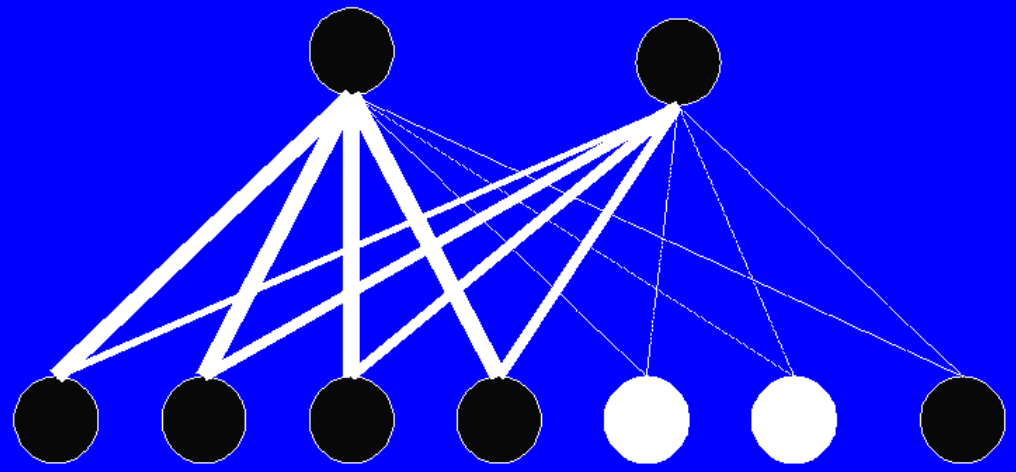
No Competition



No Competition



No Competition



Without competition, all units end up representing the same “principal component”; other, smaller correlations get ignored

Solution: Self-Organized Learning

1. Random initial weights

2. FFFB inhibition = only strongest units active.

Solution: Self-Organized Learning

1. Random initial weights

2. FFFB inhibition = only strongest units active.

3. Hebbian learning = winners get stronger (rich get richer)
   (losers don't do anything; can win on something else).

Solution: Self-Organized Learning

1. Random initial weights

2. FFFB inhibition = only strongest units active.

3. Hebbian learning = winners get stronger (rich get richer)
   (losers don't do anything; can win on something else).

4. Homeostasis: keeping things more evenly distributed (higher taxes for the rich!)

5. Goto 1

Solution: Self-Organized Learning

1. Random initial weights

2. FFFB inhibition = only strongest units active.

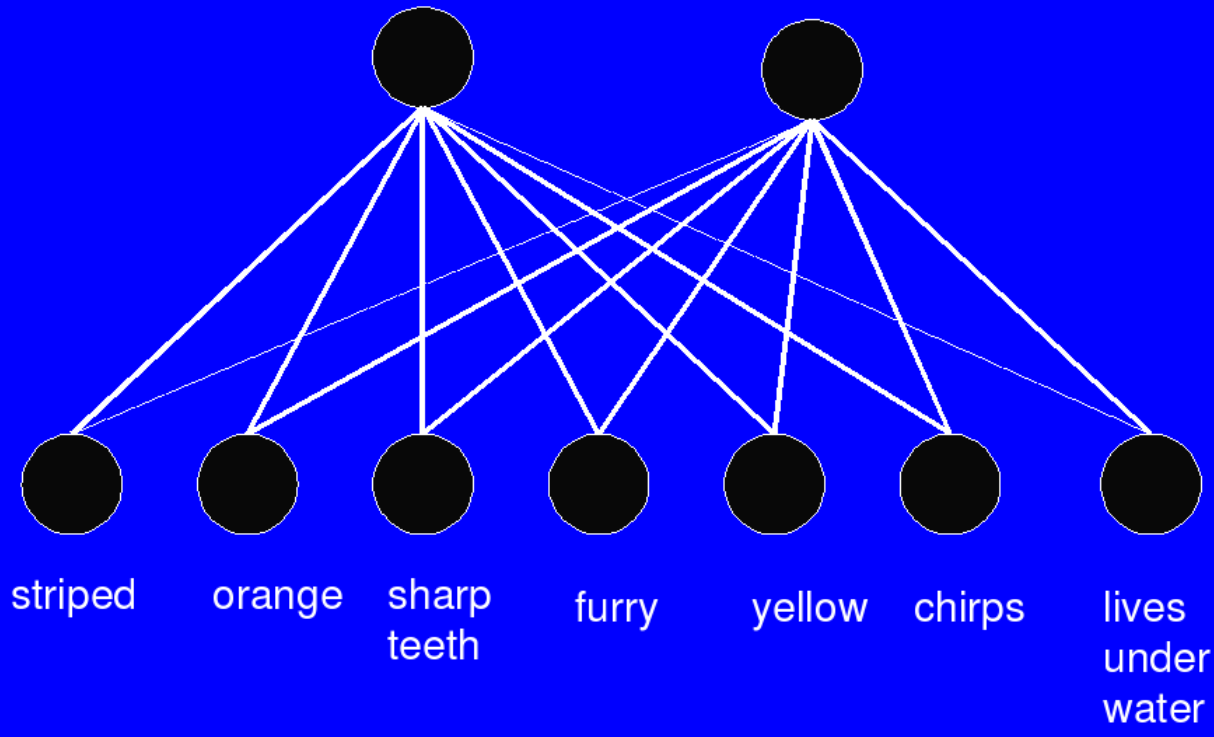
3. Hebbian learning = winners get stronger (rich get richer)
   (losers don't do anything; can win on something else).

4. Homeostasis: keeping things more evenly distributed (higher taxes for the rich!)

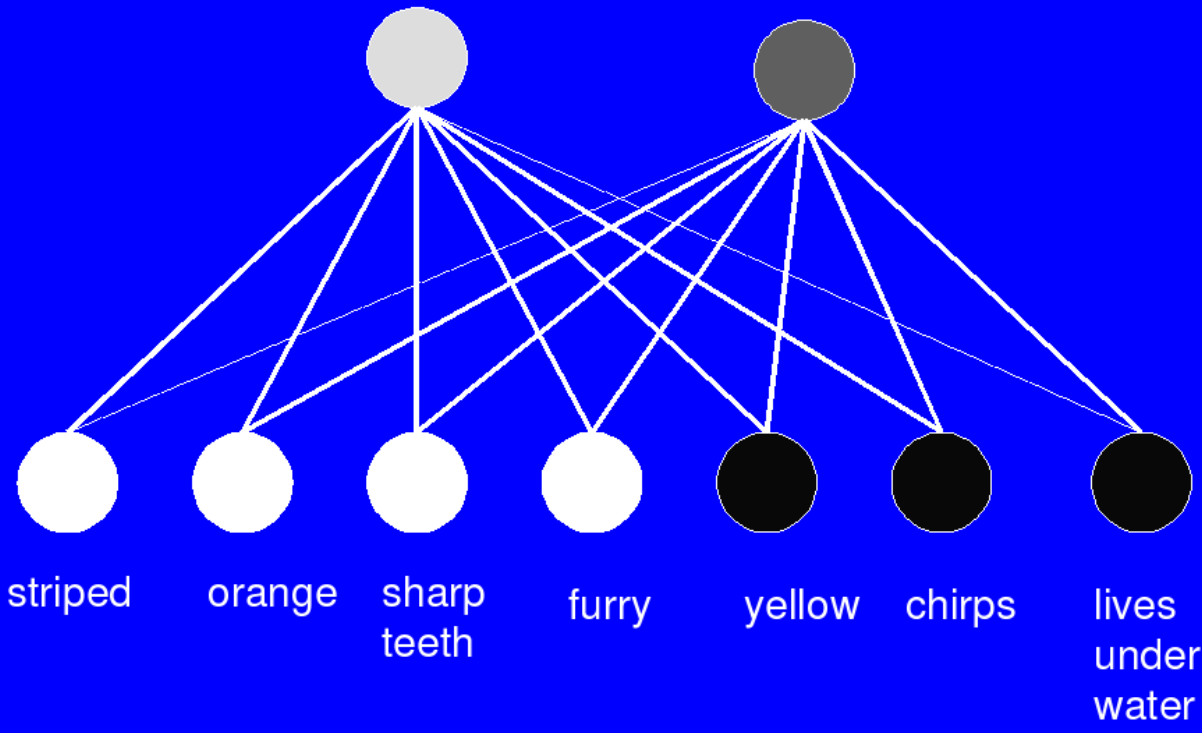
5. Goto 1

Result: different units tuned for different input features.
$\rightarrow$ *"Self-Organized Learning"*

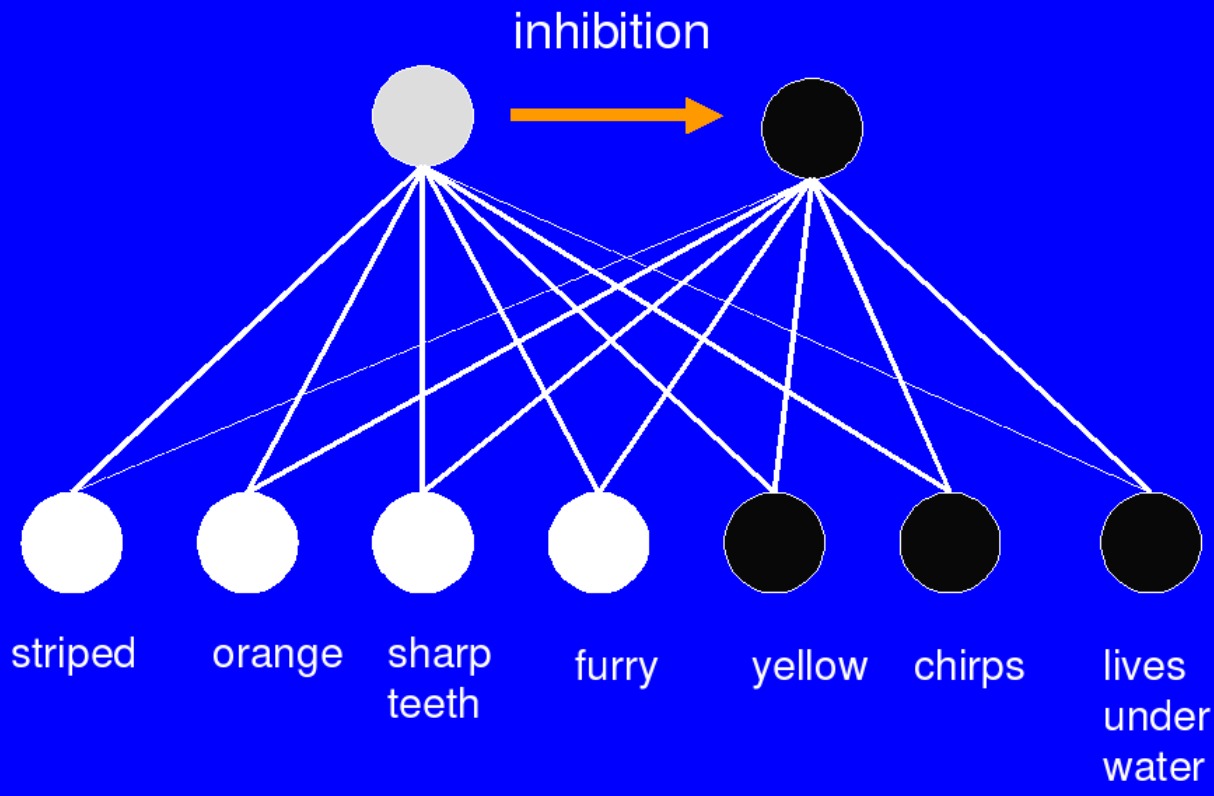
Competition is important



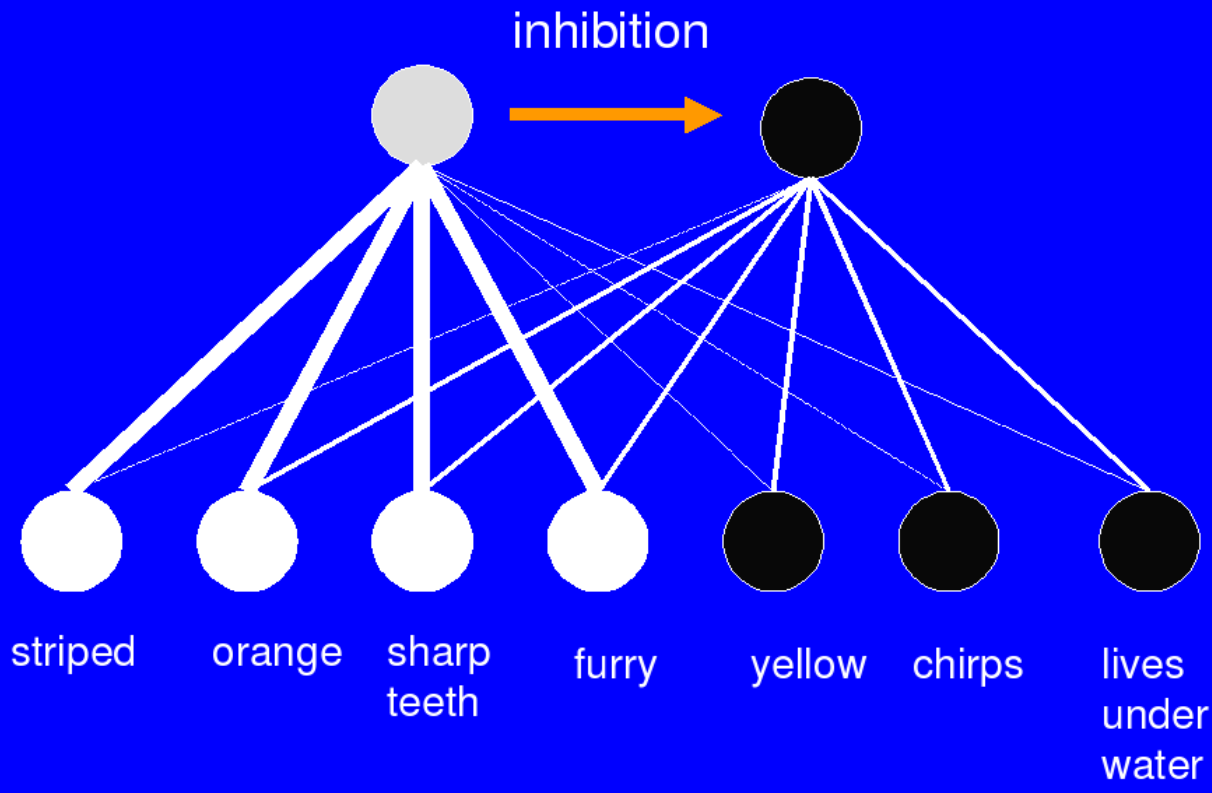
Competition is important



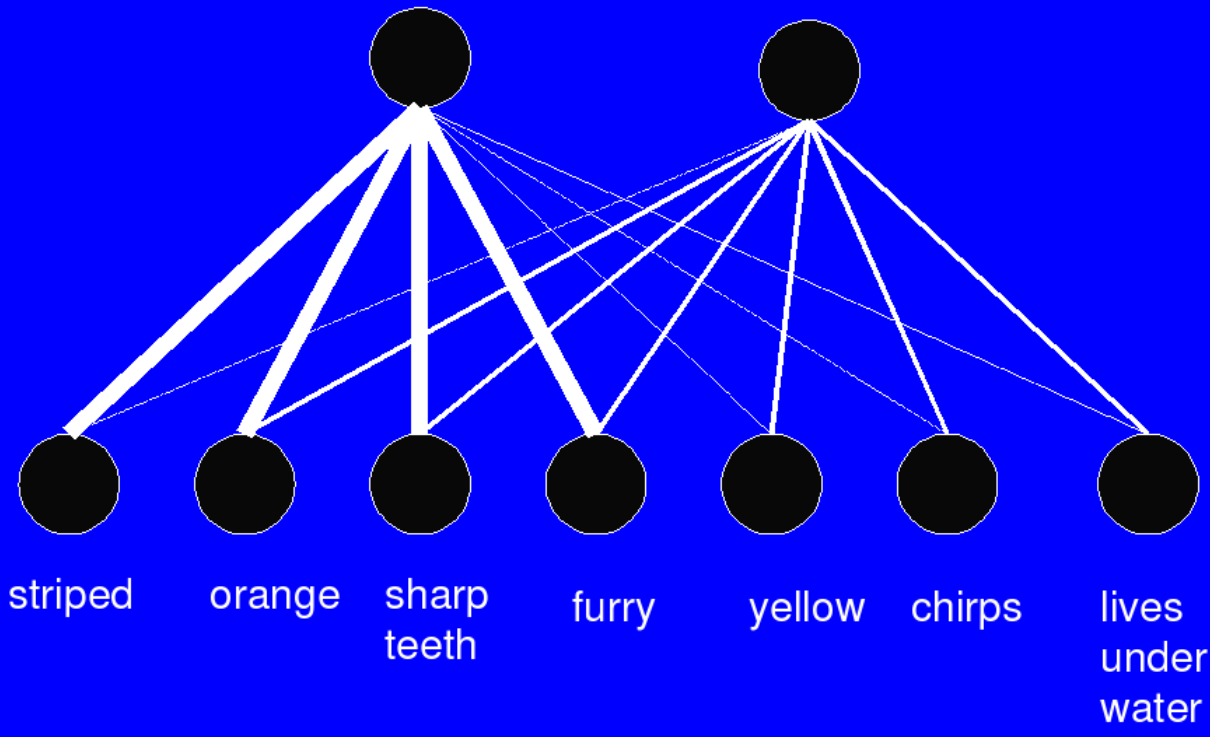
Competition is important



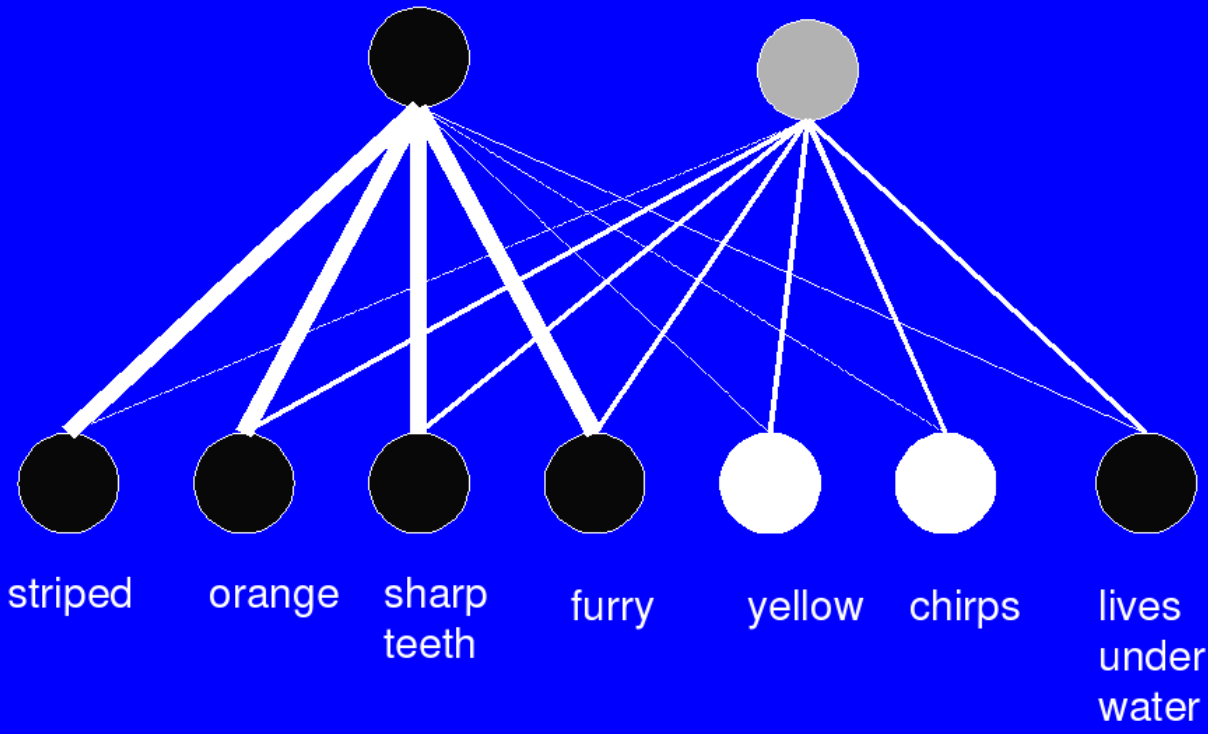
Competition is important



Competition is important



Competition is important

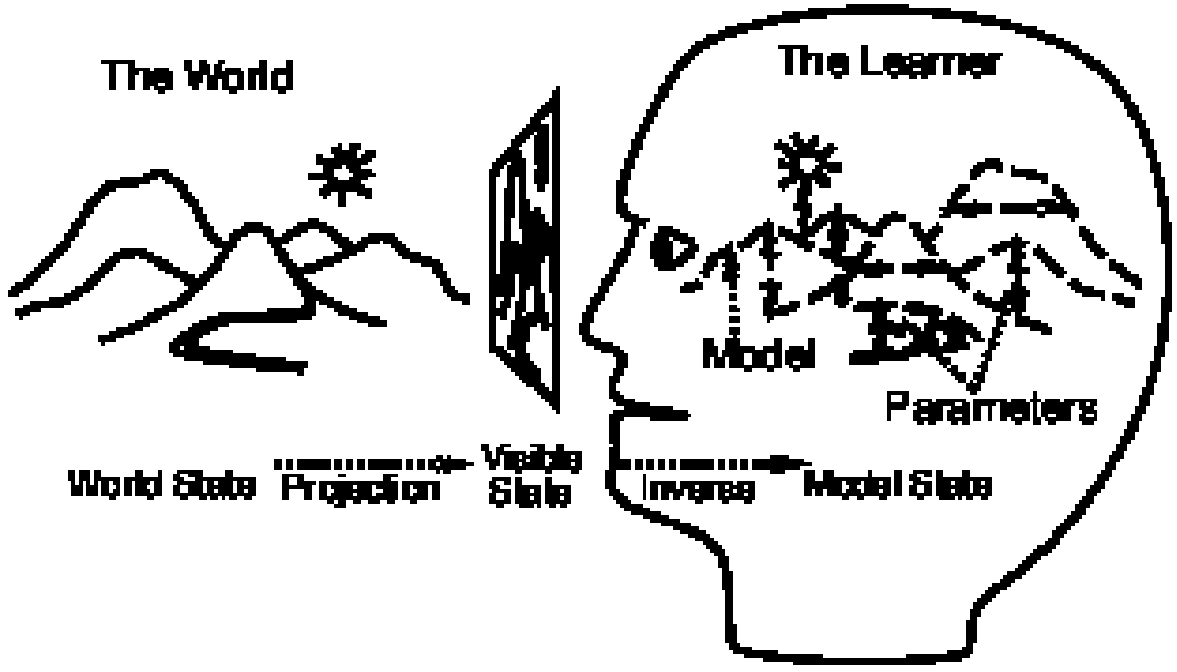


[self_org.proj]

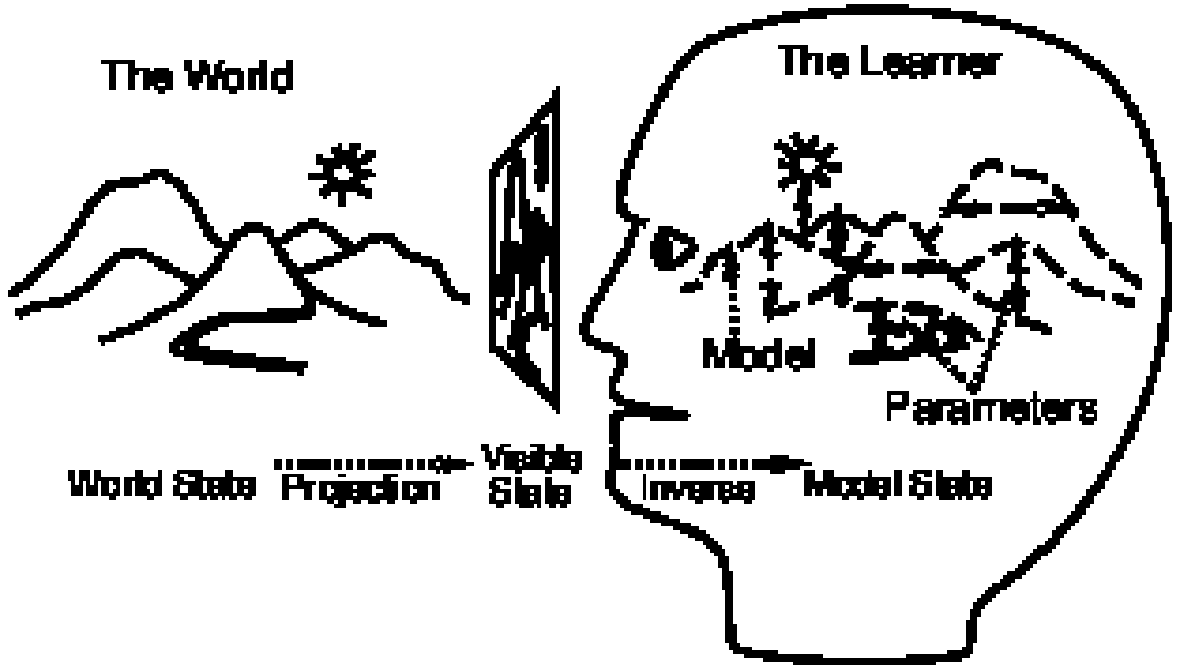
[self_org_inhib.proj]

What if don't enforce competition 'artificially' with FFFB?

Summary: statistical Learning

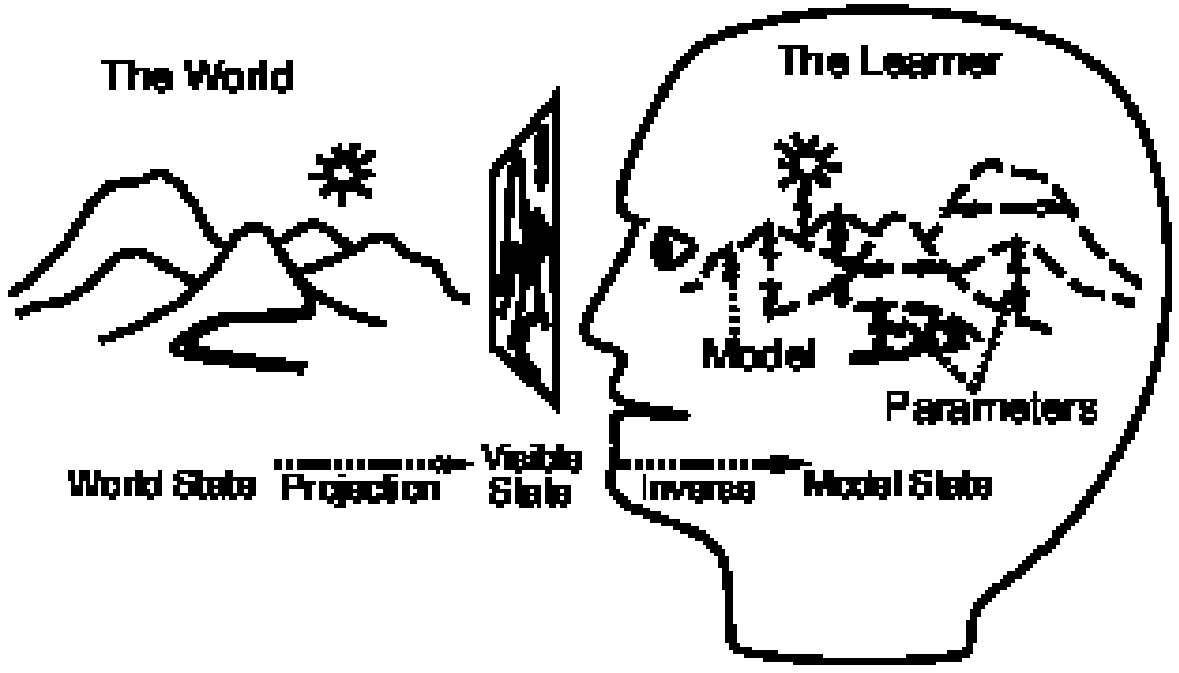


Summary: statistical Learning



Get a lot of poor quality information.

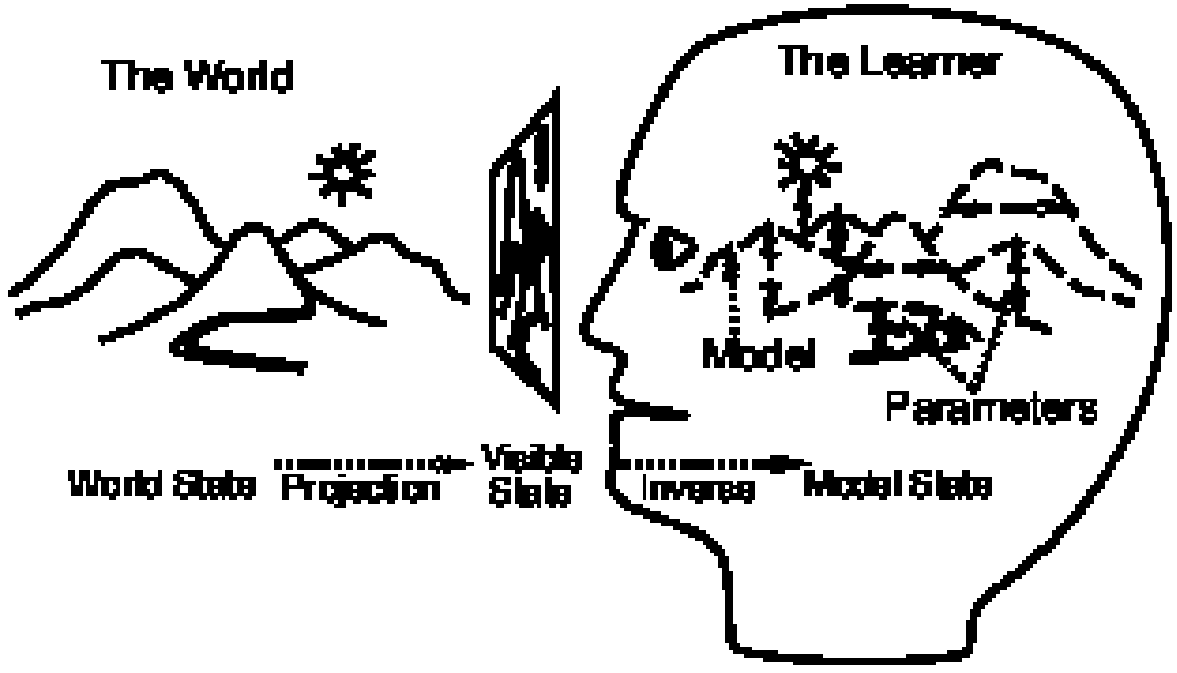
Summary: statistical Learning



Get a lot of poor quality information.

Need biases to augment, structure this info (e.g., parsimony).

Summary: statistical Learning

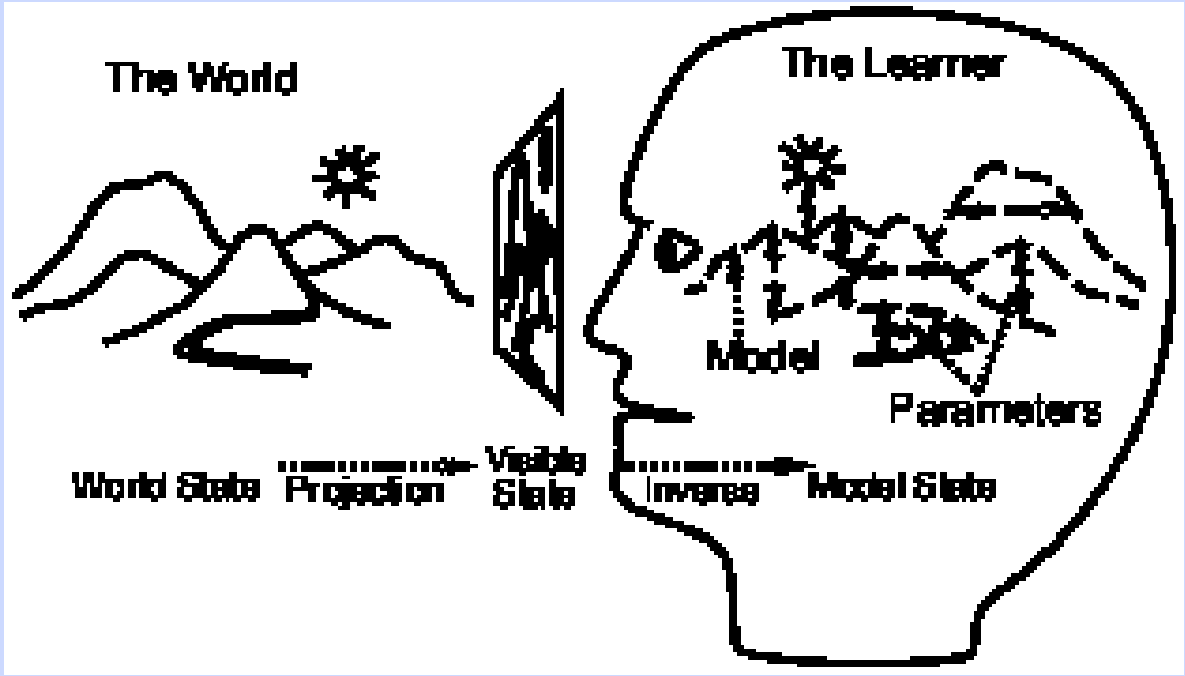


Get a lot of poor quality information.

Need biases to augment, structure this info (e.g., parsimony).

One good bias is focus on correlations

Summary: statistical Learning



Get a lot of poor quality information.

Need biases to augment, structure this info (e.g., parsimony).

One good bias is focus on correlations

Built in biases? Biological/genetic: associative LTP, different cortical layers, connectivity...

but not necessarily specific representations or knowledge!

Built in biases? Biological/genetic: associative LTP, different cortical layers, connectivity...
but not necessarily specific representations or knowledge!

$\rightarrow$ Bias to learn! (but no bias to know language, etc)

EXTRA: Fine-Tuning Hebbian Learning

The remaining stuff in chapter 4 describes various methods of fine-tuning the Hebbian learning rule to do better under different conditions. This is not critical for understanding the main principles, which remain essentially unaltered.

Corrections

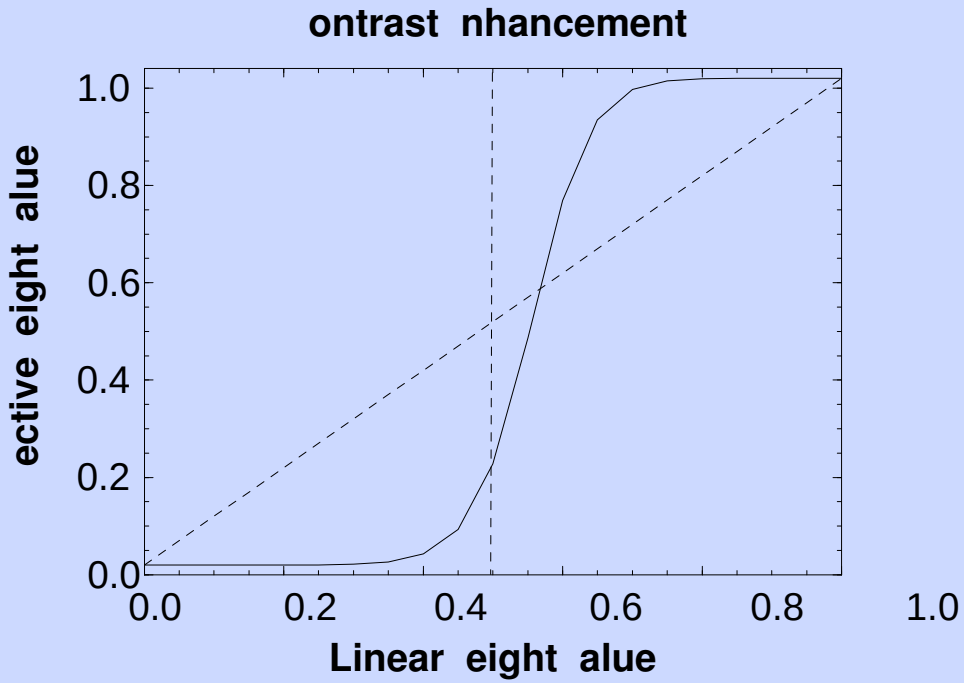
- weights don't have much *dynamic range* or *selectivity*.
- Solution: *contrast enhancement*.
- Quantitative adjustments – retain qualitative features motivated by biology.

Cleaning up Messy Receptive Fields

- The "gang" project ends up with a big weight to the first two (correlated) features, and a moderate weight to the third feature
- SO, there is a sense in which the unit is selective for the first two input features, but it is only *weakly* selective
- How can we enhance selectivity?

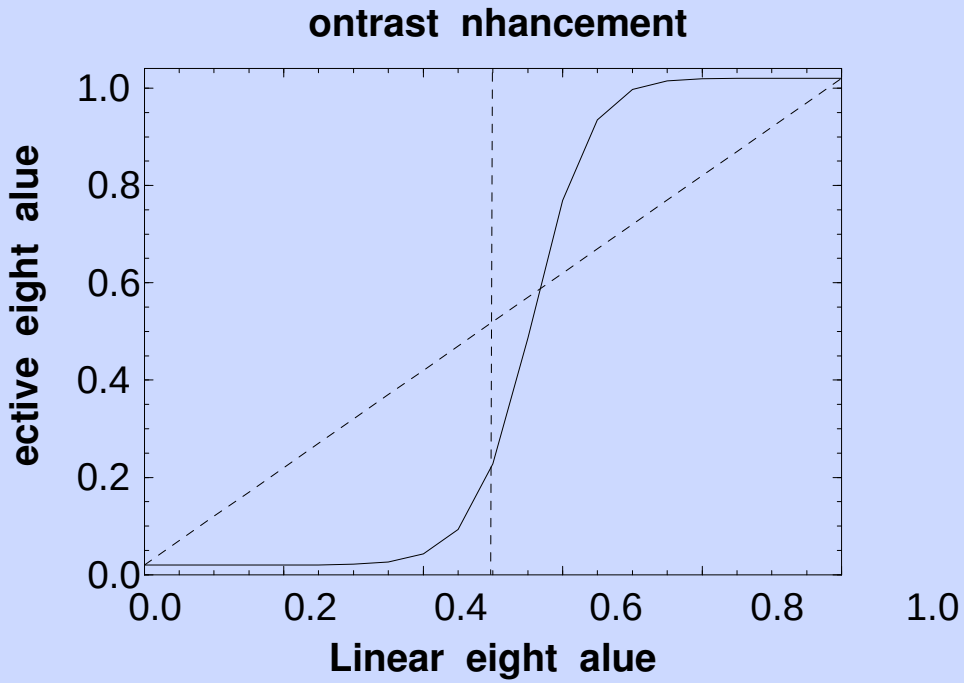
Contrast Enhancement

Between strongest and weaker correlations, via sigmoidal function:



Contrast Enhancement

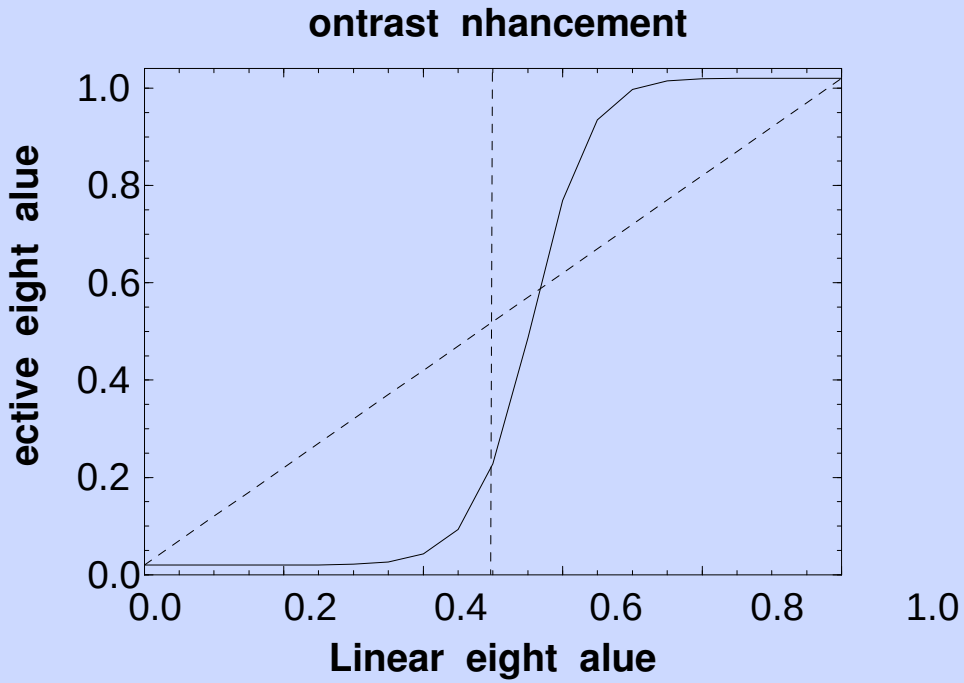
Between strongest and weaker correlations, via sigmoidal function:



Slope (sharpness of contrast) via *gain* γ (`wt_sig.gain`)

Contrast Enhancement

Between strongest and weaker correlations, via sigmoidal function:



Slope (sharpness of contrast) via *gain* γ (`wt_sig.gain`)
Offset (where midpoint is) via θ (`wt_sig.off`)

[newgang.proj]

effects of wt contrast enhancement

[self_org.proj]

effects of wt contrast enhancement

Sequential (Standard) PCA (SPCA)

First compute correlations across all inputs for first unit

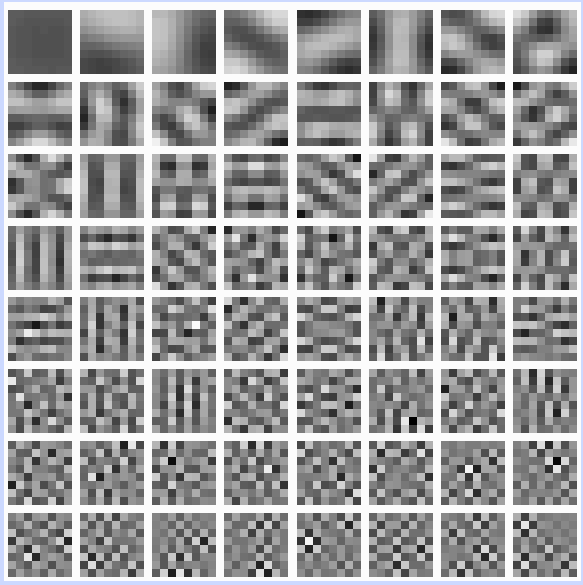
Sequential (Standard) PCA (SPCA)

First compute correlations across all inputs for first unit
Then do same for second unit, but keep it orthogonal to 1st, etc.

Sequential (Standard) PCA (SPCA)

First compute correlations across all inputs for first unit
Then do same for second unit, but keep it orthogonal to 1st, etc.

As applied to natural visual scenes:



1st is blob, 2nd is 1/2 blob, etc: Average over all inputs = blob! Problem: assumes world is *hierarchy*, but it isn't!

Variations of Hebb: CPCA learning Rule

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

$$\Delta w_{ij} = \epsilon y_j (x_i - w_{ij})$$

Variations of Hebb: CPCA learning Rule

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

$$\Delta w_{ij} = \epsilon y_j (x_i - w_{ij})$$

$\rightarrow$ Weight converges on the **probability that the sending unit is active, *given* that the receiving unit is active**

Variations of Hebb: CPCA learning Rule

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

$$\Delta w_{ij} = \epsilon y_j (x_i - w_{ij})$$

$\rightarrow$ Weight converges on the **probability that the sending unit is active, *given* that the receiving unit is active**

$\rightarrow$ If y_j is active, learned weight reflects the likelihood that x_i caused that activation

Variations of Hebb: CPCA learning Rule

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

$$\Delta w_{ij} = \epsilon y_j (x_i - w_{ij})$$

$\rightarrow$ Weight converges on the **probability that the sending unit is active, *given* that the receiving unit is active**

$\rightarrow$ If y_j is active, learned weight reflects the likelihood that x_i caused that activation

$\rightarrow$ Allows units to learn about correlations among input patterns that activate it, but not among those that don't activate it.

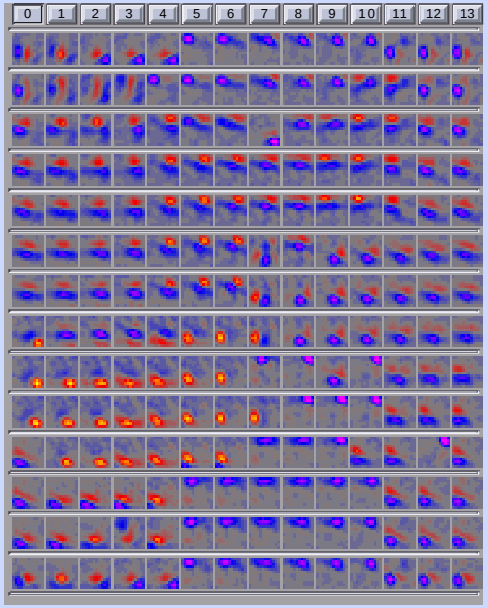
Conditional PCA (CPCA)

Compute correlations conditional on only *subset* of inputs (i.e., where particular features are present).

Conditional PCA (CPCA)

Compute correlations conditional on only *subset* of inputs (i.e., where particular features are present).

CPCA of natural visual scenes:



World is a *heterarchy* – no uber-components, just lots of features!

Comparison

SPCA operates over *all* inputs, ensures different units encode different things by making them *orthogonal*.

Comparison

SPCA operates over *all* inputs, ensures different units encode different things by making them *orthogonal*.

CPCA operates over *subsets* of inputs, ensures different units encode different things by giving them *different subsets* of inputs.

Comparison

SPCA operates over *all* inputs, ensures different units encode different things by making them *orthogonal*.

CPCA operates over *subsets* of inputs, ensures different units encode different things by giving them *different subsets* of inputs.

kWTA competition ensures that different units are active for different inputs, leads to more *meaningful* correlations

CPCA Equations

$$\Delta w_{ij} = \epsilon y_j (x_i - w_{ij}) \quad (3)$$

Weight moves towards x_i, conditional on y_j.

CPCA Equations

$$\Delta w_{ij} = \epsilon y_j (x_i - w_{ij}) \quad (3)$$

Weight moves towards x_i, conditional on y_j.

$\rightarrow$ learns principal components conditional on whether unit y_j is active... principal component that activates that unit.

Achieves conditional probability goal:

$$\begin{aligned} w_{ij} &= P(x_i = 1 | y_j = 1) \\ &= P(x_i | y_j) \end{aligned} \quad (4)$$

Weight = prob. that the sender x_i is active given that receiver y_j is.