COMP596: Brain-inspired artificial intelligence

Week 4: Unsupervised learning

Blake Richards

Mila/McGill/CIFAR

Outline

1. Unsupervised learning in the brain
2. Competitive learning
3. Boltzmann machines

Unsupervised learning in the brain

Reminder from last week

Distributed representations are an approach to representing information using a group of units or “neurons”, in contrast to using local representations where each concept gets its own unit/neuron.

One advantage of distributed representations is that creating new concepts is relatively easy, and they are well-suited for things like multiple constraint satisfaction and spontaneous generalization.

But, how can we learn ***useful*** representations?

How to learn useful representations

What would it mean for a representation to be *useful*?

When we are receiving explicit instruction, that is relatively easy to answer: good representations let us do what we are instructed to do.

How to learn useful representations

But most of the time, we are not receiving explicit instruction. Rather, we just experiencing the world as a stream of data.

This situation, where we must learn good representations from sensory data without instruction is what we call **unsupervised learning**.

How to learn useful representations

But I still haven't answered the question:

> *What does it mean to learn a good distributed representation in unsupervised learning?*

It's difficult to answer that question, a priori. So, a reasonable place to start is to simply ask: what does the brain do?

Seeking guidance from the brain

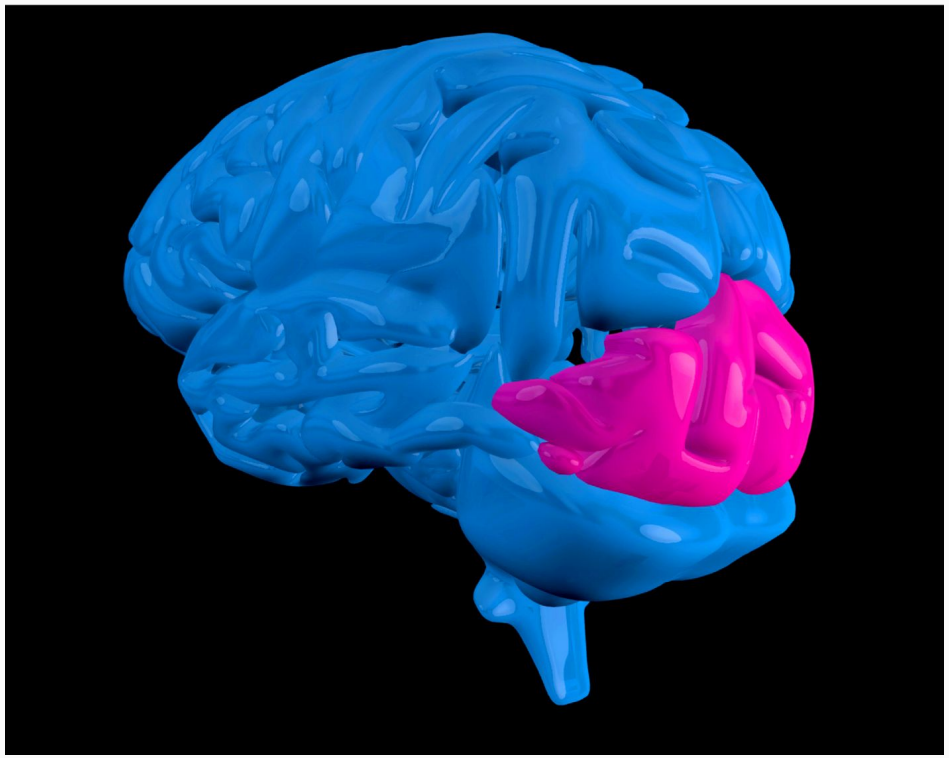
What do we know about what the brain does? A few of things stand out:

- The brain learns to devote representational space to particularly salient or frequent stimuli.
- The brain learns to group stimuli into different categories, based simply on the statistics of their (co-)occurrence.
- The brain learns about constraints on possible arrangements of the world.

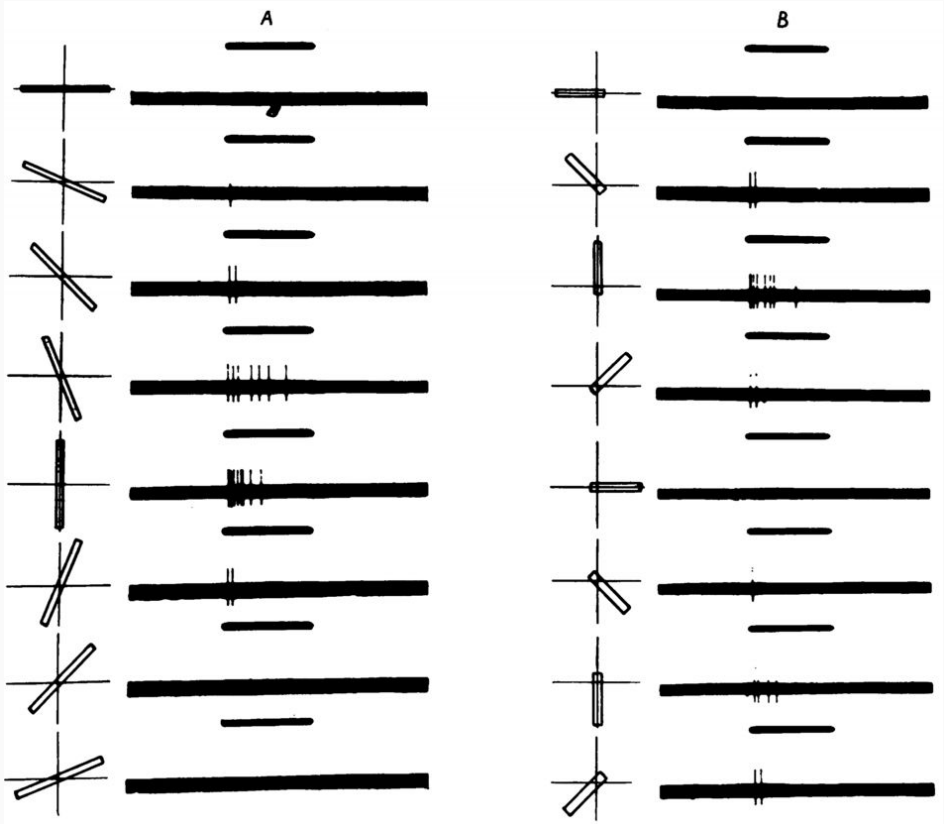
Let's look at a little bit of the psychology/neuroscience data on each of these.

Learning about frequent/salient stimuli

One of the earliest discoveries in “systems neuroscience” (that is neuroscience that studies computations in neural circuits) was the discovery by Hubel & Wiesel of orientation selective neurons in the primary visual cortex of mammals.



Learning about frequent/salient stimuli



(Image from Hubel & Wiesel, 1959, J. Physiol., 148: 574-591)

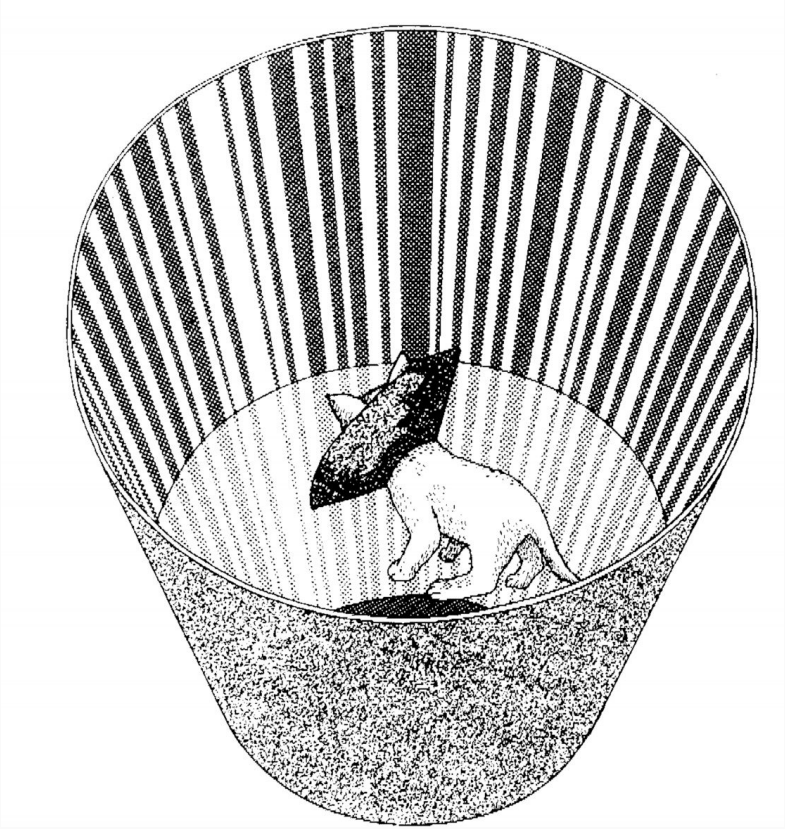
Learning about frequent/salient stimuli

Orientation selectivity is hardwired, but tunable!

If you raise an animal in an environment with an overabundance of vertical or horizontal lines, their primary visual cortex learns to devote greater representational space to those orientations.

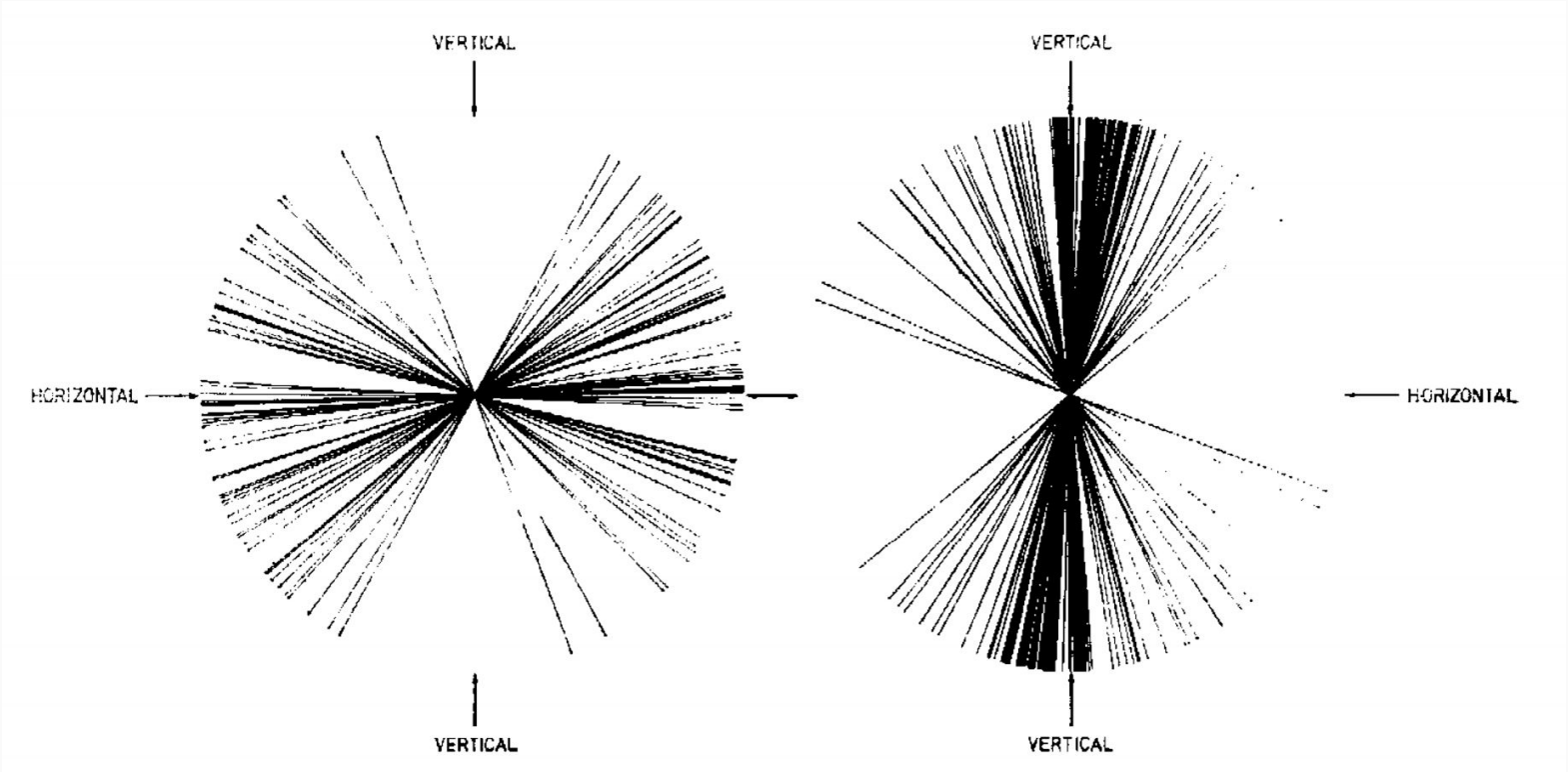
By this I mean that there are more neurons selective for those orientations, which in the PDP framework, means literally a higher-dimensional region of space devoted to those orientations.

Learning about frequent/salient stimuli



(Image from Blakemore & Cooper, 1970, Nature, 228: 477-478)

Learning about frequent/salient stimuli

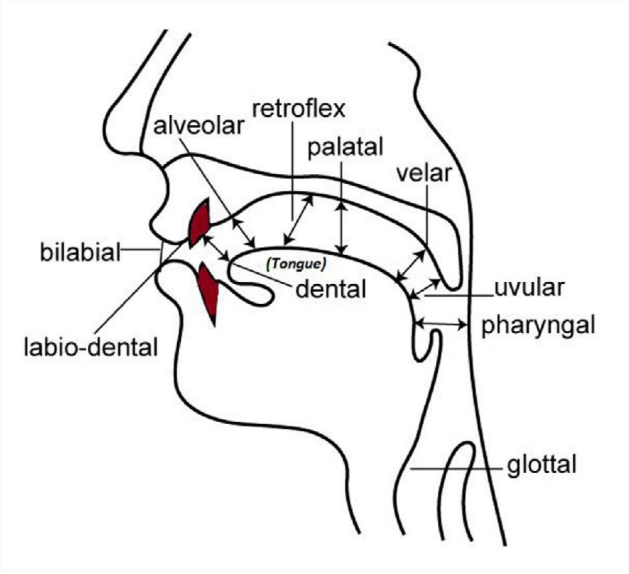


(Image from Blakemore & Cooper, 1970, Nature, 228: 477-478)

Learning about categories

Humans are naturally born with the ability to distinguish many different subtle sounds that are used in human languages around the world.

But, as we grow older, we ***lose*** the ability to distinguish sounds that are not in our mother tongue. For example, English speakers struggle to hear the difference between the Hindi dental vs. retroflex 'T' or 'D'.



Learning about categories

We can test the ability to distinguish these consonants in infants by training them to associate a change in consonants being played with the appearance of a toy. If the infant looks for the toy when the consonant changes, it suggests they can distinguish the sounds, if they don't it suggests they don't perceive the difference.

Learning about frequent/salient stimuli

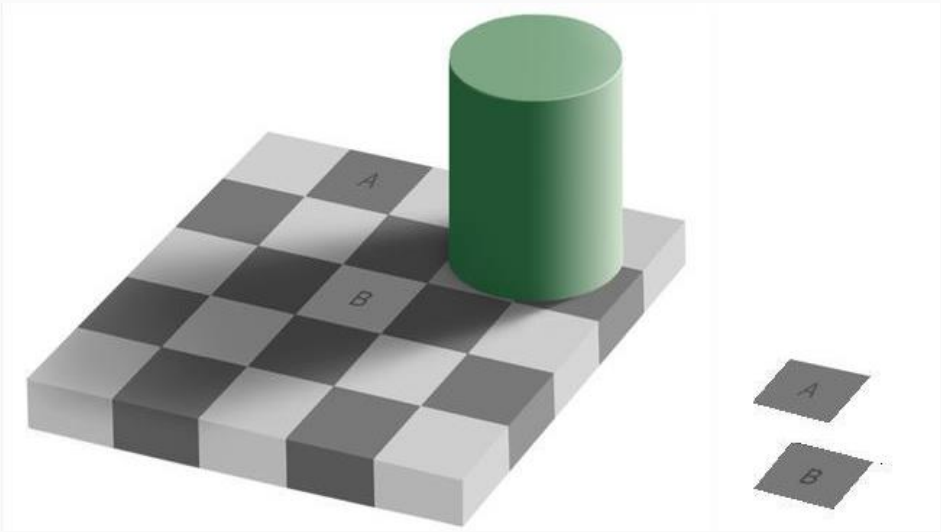
TABLE 1
Infant Discrimination Performance on Two Non-English Speech Contrasts

Reached Criterion	*(1) 6–8 months*	*(2) 8–10 months*	*(3) 10–12 months*
	The Retroflex/Dental Contrast /ʈa/ - /ta/		
Yes	11	8	2
No	1	4	8
	The Velar/Uvular contrast /k̀i/ - /q̀i/		
Yes	8	8	1
No	2	6	9

(From Werker & Tees, 1984, Infant Behavior and Develop., 7: 49-63)

Learning about constraints in the world

What we perceive is not reality. Rather, our brain makes inferences about the world based on whatever best satisfies the set of multiple constraints that we are operating with.



Learning about constraints in the world

We have some very explicit expectations about constraints in the world when we are very young (it is hard to experimentally determine whether this is genetically hard-wired or learned). But, what we can say is that we try to learn from violations of previously known constraints, and update our understanding of the constraints in the world based on this learning.

Interim summary

- Brains learn in an unsupervised manner most of the time, especially in early life.
- Using unsupervised learning, brains are able to:
  - Devote representational space to particularly salient or frequent stimuli
  - Group stimuli into different categories, based simply on the statistics of their (co-)occurrence
  - Learn about constraints on possible arrangements of the world

Competitive learning

Learning about natural groups in the world

One of the earliest ideas that people had about how to model unsupervised learning was to try to use patterns in data to classify different sensory inputs via **competitive learning**. Competitive learning refers to a broad class of learning algorithms for PDP models which obey the following basic principles:

- Start with neurons that are computationally the same, but which start off with different synaptic connections, making each of them respond differently to each stimulus from the outset.
- Limit the overall strength of each neuron's synapses.
- Have groups of neurons compete for the "right" to respond to stimuli.
- Train the network so that "winning" neurons are more likely to respond to stimuli they win in the future.

Learning about natural groups in the world

- Start with neurons that are computationally the same, but which start off with different synaptic connections, making each of them respond differently to each stimulus from the outset.
- Limit the overall strength of each neuron's synapses.
- Have groups of neurons compete for the "right" to respond to stimuli.
- Train the network so that "winning" neurons are more likely to respond to stimuli they win in the future.

The idea is that different neurons will learn to respond to different natural groups in the sensory data, thereby clustering incoming sensations.

The Rumelhart & Zipser model

Rumelhart & Zipser (1985) studied a specific form of competitive learning, which has the following additional specific properties:

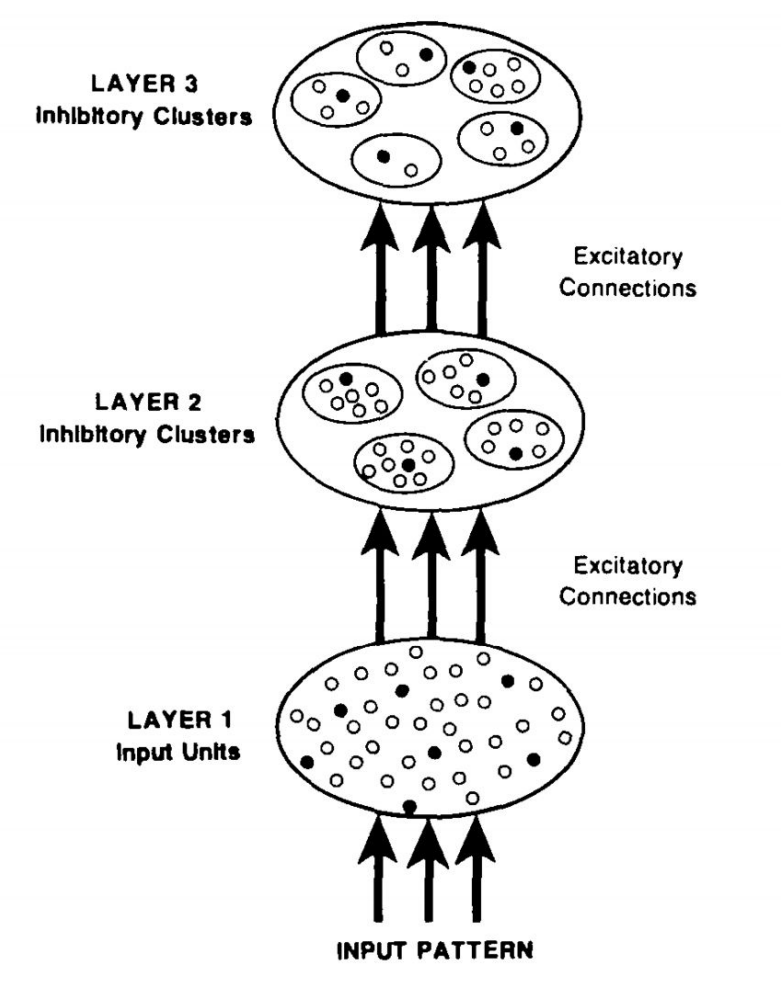
- Build a network with multiple layers, with an initial "input" layer for sensory stimuli. All neurons have binary activity (0 or 1).
- Neurons in each layer are broken into non-overlapping clusters/sets. The clusters are "winner-takes-all" such that the neuron, i, with the strongest input gets $a_i = 1$, and inhibits all the other neurons in the cluster, such that $a_j = 0 \; \forall j \neq i$.
- Every neuron in a cluster receives the same inputs as every other neuron in the cluster.

The Rumelhart & Zipser model

Rumelhart & Zipser (1985) studied a specific form of competitive learning, which has the following additional specific properties (continued):

- Neurons learn only if they win a competition.
- The synaptic weights for each neuron are constrained such that for neuron i, all synapses onto it sum to 1, i.e. $\sum_j W_{ij} = 1$.
- When a neuron wins, it reduces its synaptic weights evenly, and adds to those weights that were receiving input.

The Rumelhart & Zipser model



The Rumelhart & Zipser model

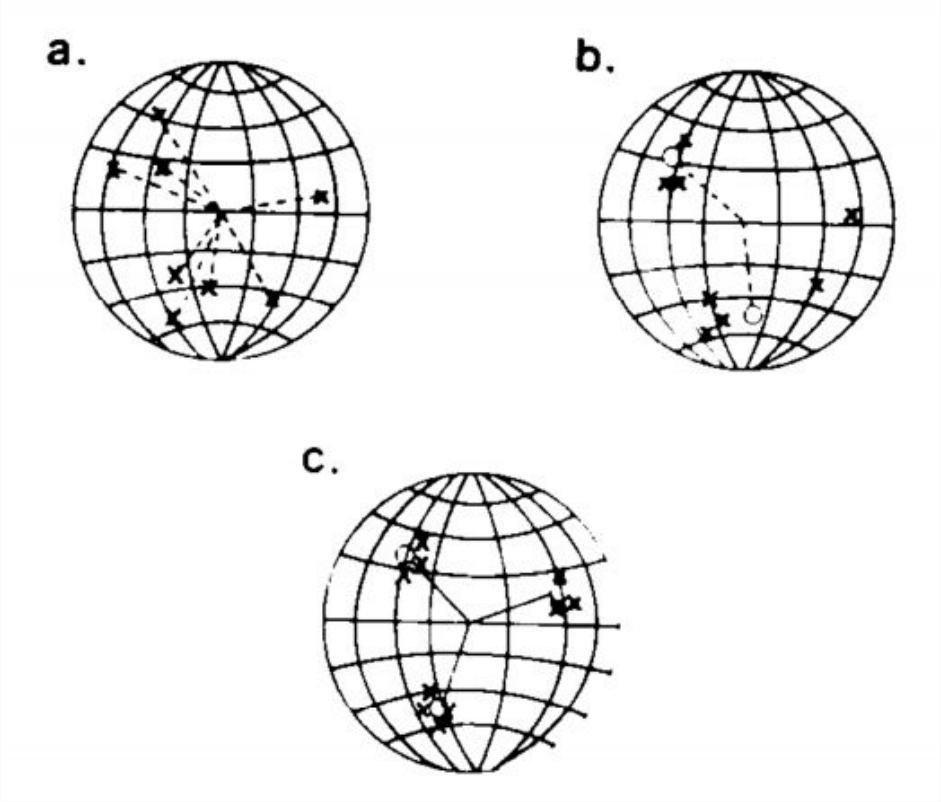
The specific learning algorithm Rumelhart & Zipser (1985) was as follows, for an excitatory synapse from neuron i in a lower layer to neuron j in a higher layer, the weight between them W_{ij} updates in response to simulus S_k with:

$$\Delta W_{ij} = \begin{cases} 0, \text{ if unit } i \text{ loses on stimulus } k \\ \eta(\frac{a_j^k}{n^k} - W_{ij}), \text{ if unit } i \text{ wins on stimulus } k \end{cases}$$

where a_j^k is the activity of unit j for stimulus k, η is the learning rate, and n_k is the number of units on in the layer below ($n_k = \sum_j a_j^k$).

The Rumelhart & Zipser model

What does this learning rule learn? Rumelhart & Zipser provide the following geometric intuition:

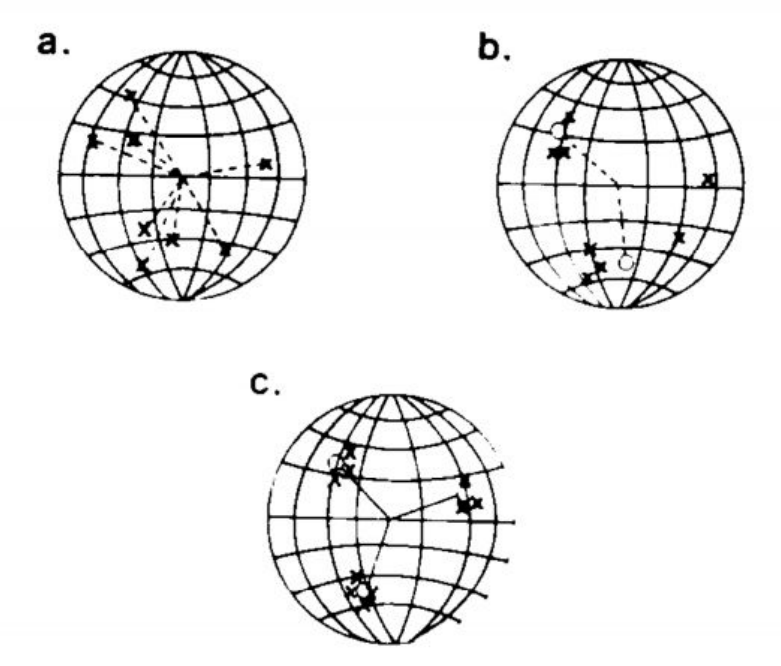


The Rumelhart & Zipser model

Note though:
This geometric example is technically incorrect. In reality, the activity patterns sit on the corners of a hypercube, and the weights sit on the face of a hyperplane contained within the cube. This learning rule will push the weights within the plane in order to be as close as possible to all the corners for which the neuron wins the competition.

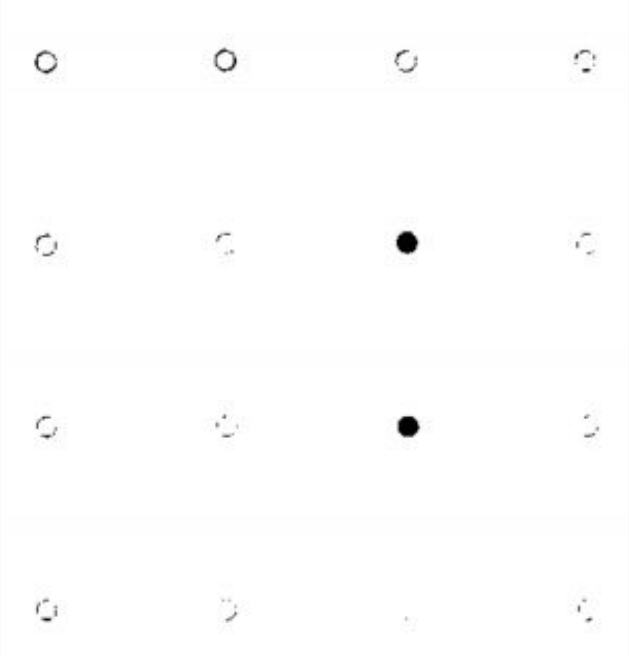
The Rumelhart & Zipser model

Either way, if the data falls into M "natural groups" (i.e. clusters), and there are $N \geq M$ neurons in a competitive cluster, this learning algorithm will assign at least one neuron from each cluster to each of the natural groups. The representation for each group is distributed across the clusters in this manner.

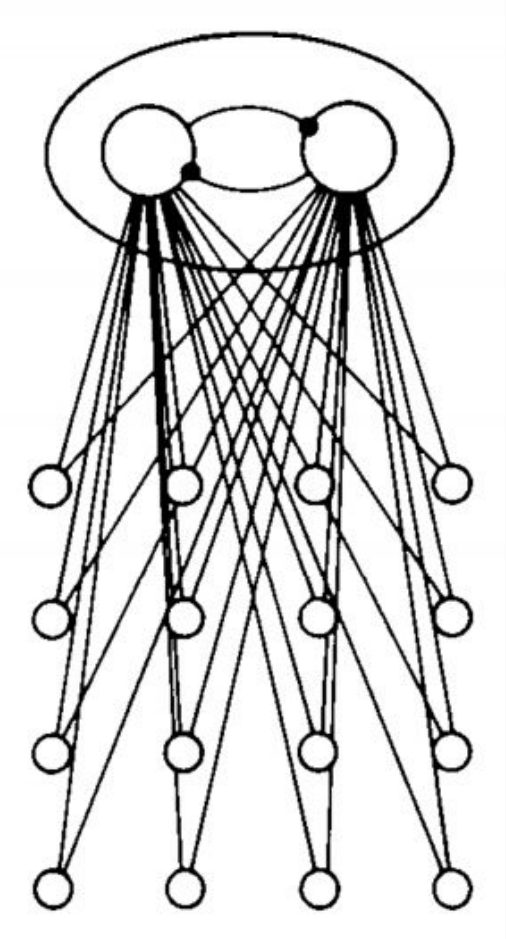


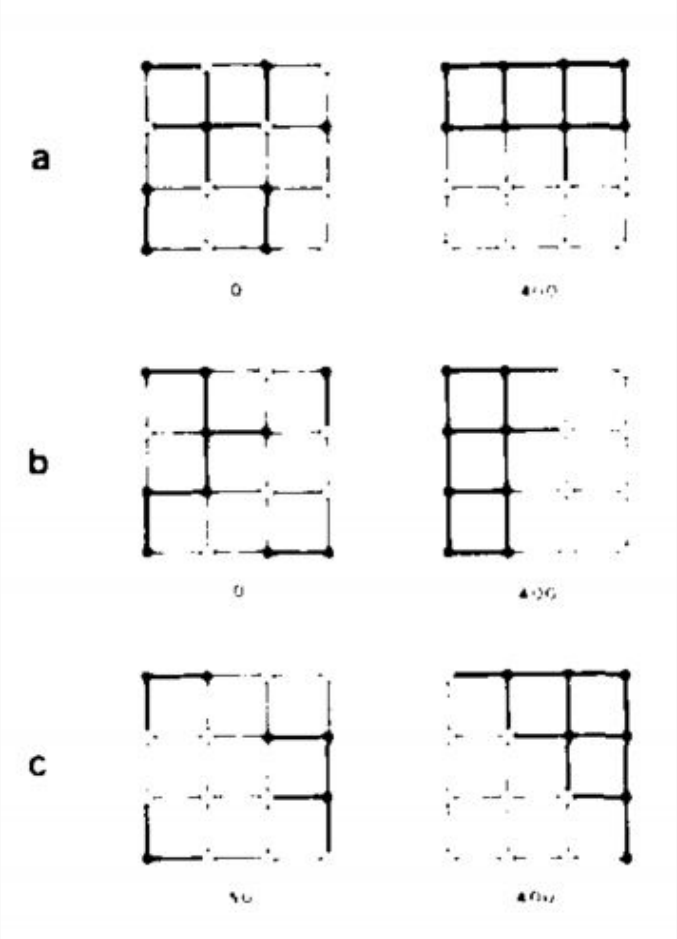
Toy example: dipole learning

The network is presented with points from a grid, and every stimulus consists of two adjacent points from the grid. (But, note: the actual structure of space is *never* given to the network.)

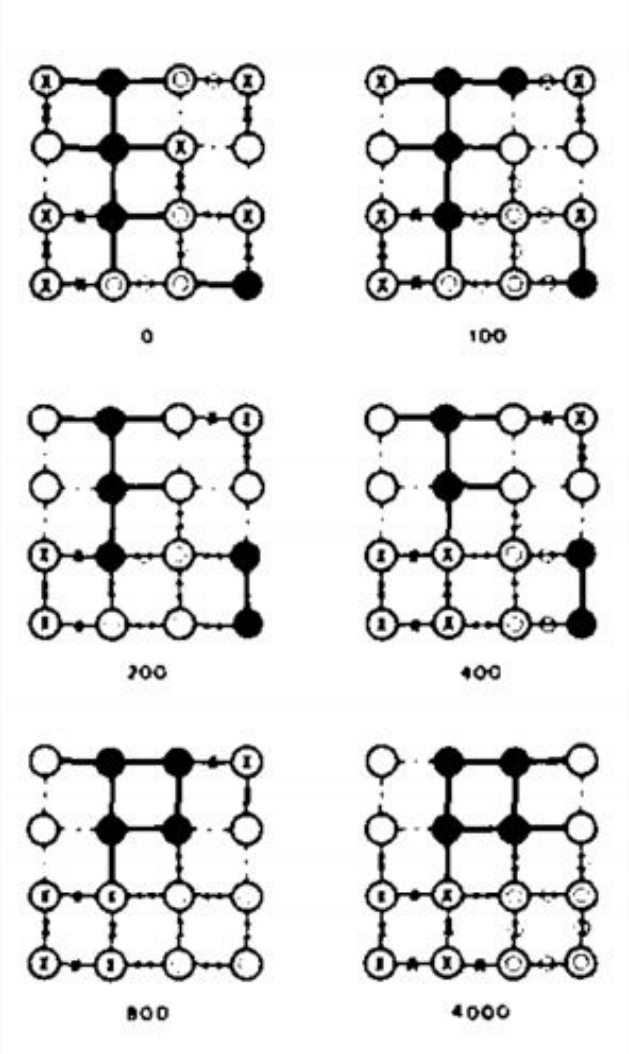


Toy example: dipole learning





Toy example: dipole learning



Formal analysis

The learning algorithm will converge when:

$$W_{ij} = \frac{\frac{\sum_k p^k a_i^k v_i^k}{n^k}}{\sum_k p^k v_i^k}$$

where p^k is the probability that stimulus k is presented on any trial, and v_i^k is the probability that neuron i wins when stimulus k is presented.

Formal analysis

The learning algorithm will converge when:

$$W_{ij} = \frac{\frac{\sum_k p^k a_i^k v_i^k}{n^k}}{\sum_k p^k v_i^k}$$

This is really just the conditional probability that input neuron j is on when unit i has won the competition.

Formal analysis

However, the stability of these equilibrium points depends on whether the data falls into natural groups or not. Specifically, the stability will depend on how much the units win all their points by on average.

If the data tends to have groups where there is roughly the same overlap between any set of patterns, then the units will usually just barely win the competitions, and the algorithm can be unstable. (See the reading for more analysis of the algorithm's stability.)

Why don’t we use competitive learning more nowadays?

There are more principled means of clustering data (e.g. k-means), and these methods are more stable and less finicky than competitive learning.

Perhaps there would be a way to make a better version of these algorithms, and maybe the brain uses something kind of like this...

Summary

- Competitive learning refers to a class of PDP models wherein groups of neurons compete with each other to represent a stimulus.
- If the data falls into natural clusters, competitive learning will discover those clusters and will come to represent the data using a different neuron in each group for each cluster.
- If done in a hierarchical circuit competitive learning can identify higher-order clusters within the data (see the reading).

Boltzmann machines

The need for higher-order weak constraints

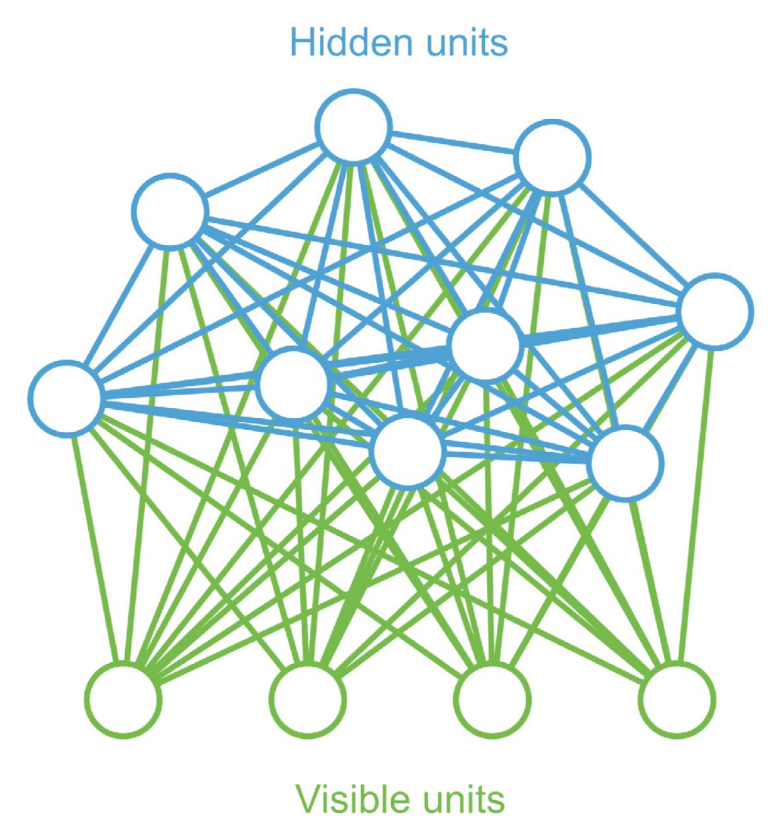
Consider the sorts of data we get in the natural world, such as images. There are constraints that determine how the pixels in the image will appear, but those constraints are “higher-order” and “weak”, meaning that you can’t express them as hard constraints on pairs of pixels, but rather as soft constraints, on soft constraints, on soft constraints (etc.) of pairs of pixels.

The need for higher-order weak constraints

How can we capture higher-order, weak constraints? How can we learn them from sensory data?

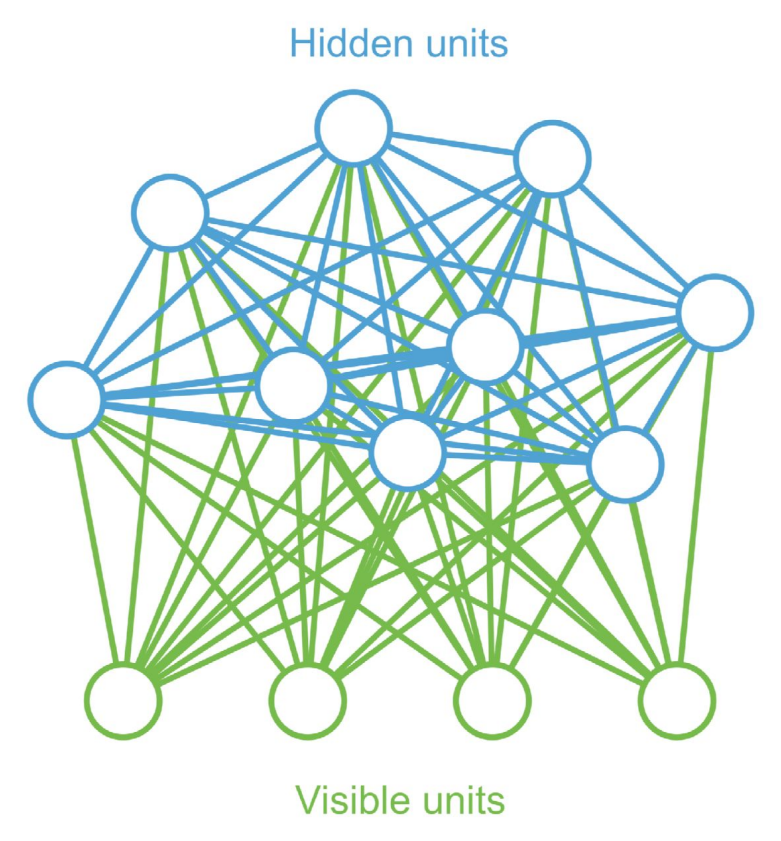
The **Boltzmann machine** is a model that can accomplish exactly this!

Boltzmann machines



We have two groups of neurons, one that receives sensory data (the "visible" units, V) and one that represents the higher-order, weak constraints (the "hidden" units, H).

Boltzmann machines



There are no connections between visible units. Hidden units connect to all visible units and to each other. All weights are symmetric (i.e. $W_{ij} = W_{ji}$).

Boltzmann machines

The activity is binary (0 or 1), but the activation function is stochastic. Specifically, the probability that a unit i has $a_i(t) = 1$ is:

$$p(a_i(t) = 1 | \mathbf{a}(t-1)) = \frac{1}{1 + \exp \frac{-\sum_j W_{ij} a_j(t-1) + \theta_i}{T}} \tag{1}$$

where $\mathbf{a} = [a_1, \ldots, a_n]]^T$, θ_i is the threshold, T is the "temperature".

Boltzmann machines

This makes the Boltzmann machine akin to a system in statistical mechanics. The "energy" of any state $\mathbf{a}^k$ of this system is:

$$E^k = -\sum_{i<j} W_{ij} a_i^k a_j^k + \sum_i \theta_i a_i^k \tag{2}$$

On average, the system will always move to lower energy states.

Boltzmann machines

Specifically, you can show using the same derivation of Maxwell-Boltzmann statistics from statistical mechanics that the probability of state k is given by:

$$p^k = \frac{\exp \frac{-E^k}{T}}{\sum_j \exp \frac{-E^j}{T}}$$

Boltzmann machines

This also means that at thermal equilibrium (when the probability distribution has stabilized), that the ratio of probabilities for any two states k and l follows a Boltzmann distribution (hence the name of the model):

$$\frac{p^k}{p^l} = \exp \frac{-(E^k - E^l)}{T}$$

Note that this equals 1 when the temperature is infinite, and goes towards $\infty/0$ as it goes to zero. In other words, the temperature controls the entropy of the system, infinite temperature corresponds to the maximum entropy (just as in physics), zero temperature corresponds to no entropy.

Training Boltzmann machines

Our goal with the Boltzmann machine is to get the hidden unit weights to capture the higher-order, weak constraints on the visible units that the world provides. How can we train the system to do this?

The basic idea:
If the hidden units capture the correct constraints, then when the system runs without any sensory inputs it should generate data on the visual units that looks like data from the real world. Put another way: **the goal of learning is to make the network's dreams look as much like reality as possible**.

Training Boltzmann machines

The goal of learning is to make the network's dreams look as much like reality as possible. How can we do this?

Let $\mathbf{v}^k$ represent the setting for the visible units with stimulus k is presented, i.e. $\mathbf{v}^k \rightarrow a_i(t) = \{0, 1\}, \forall i \in V$.

Let $p^+(\mathbf{v}^k)$ represent the probability of $\mathbf{v}^k$ according to the world, and $p^-(\mathbf{v}^k)$ represent the probability of $\mathbf{v}^k$ according to the Boltzmann machine when it's free running (i.e. "dreaming").

Training Boltzmann machines

The goal of learning is to make the network's dreams look as much like reality as possible. The loss function that we want to reduce is a measurement of the difference between the "awake" probabilities and the "dream" probabilities. Specifically, it is:

$$\mathcal{L} = \sum_k p^+(\mathbf{v}^k) \ln \left(\frac{p^+(\mathbf{v}^k)}{p^-(\mathbf{v}^k)} \right) \tag{3}$$

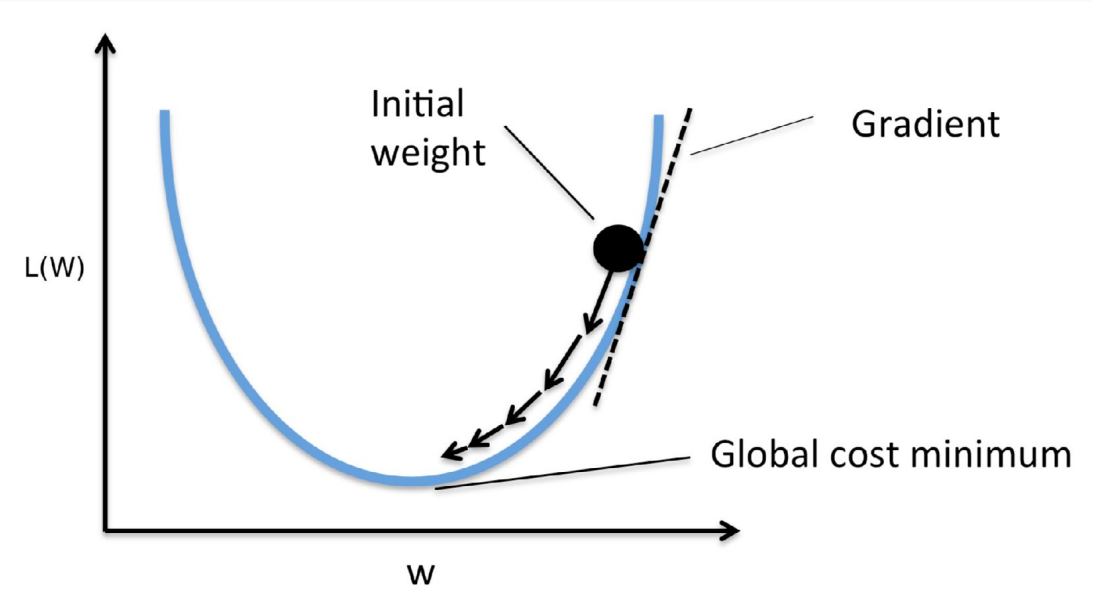
Training Boltzmann machines

This is the **Kullback-Leibler divergence** between $p^+(\mathbf{v}^k)$ and $p^-(\mathbf{v}^k)$. It measures the number of bits required to encode data generated from $p^+(\mathbf{v}^k)$ using $p^-(\mathbf{v}^k)$, and it is zero iff $p^+(\mathbf{v}^k) = p^-(\mathbf{v}^k)$.

$$\begin{aligned}\mathcal{L} &= \sum_k p^+(\mathbf{v}^k) \ln \left(\frac{p^+(\mathbf{v}^k)}{p^-(\mathbf{v}^k)}\right) \\ &= D_{KL}(p^+(\mathbf{v}^k)||p^-(\mathbf{v}^k))\end{aligned}$$

Training Boltzmann machines

How can we reduce $\mathcal{L}$? A very general technique for reducing a loss function is to perform **gradient descent**.



Deriving the weight update

What are the partial derivatives of $\mathcal{L}$ with respect to W_{ij} in a Boltzmann machine?

To derive this we need a couple of initial observations. First, remember that:

$$p(\mathbf{a}^k) = \frac{\exp \frac{-E^k}{T}}{\sum_j \exp \frac{-E^j}{T}} \tag{4}$$

Deriving the weight update

And...

$$\frac{p(\mathbf{a}^k)}{p(\mathbf{a}^l)} = \exp \frac{-(E^k - E^l)}{T} \tag{5}$$

Deriving the weight update

Also, note that if we define $\mathbf{a}^{kl} = \mathbf{v}^k \wedge \mathbf{h}^l$:

$$\begin{aligned} p^-(\mathbf{v}^k) &= \sum_l p^-(\mathbf{a}^{kl}) \\ &= \sum_l p^-(\mathbf{v}^k \wedge \mathbf{h}^l) \\ &= \sum_l \frac{\exp \frac{-E^l}{T}}{\sum_{x,y} \exp \frac{-E^{xy}}{T}} \end{aligned} \tag{6}$$

Deriving the weight update

Taking the partial derivative of the energy with respect to the weights we get:

$$\frac{\partial E^{xy}}{\partial W_{ij}} = -a_i^{xy} a_j^{xy} \tag{7}$$

See equation 2.

Deriving the weight update

Using equation 7 we get:

$$\frac{\partial \exp -E^{xy}/T}{\partial W_{ij}} = \frac{1}{T} \exp\left(-E^{xy}/T\right) a_i^{xy} a_j^{xy} \tag{8}$$

Deriving the weight update

Using equation 8 and equation 6 we get:

$$\begin{aligned}\frac{\partial p^-(\mathbf{v}^k)}{\partial W_{ij}} &= \frac{\frac{1}{T}\sum_l \exp\left(-E^l/T\right)a_i^{kl}a_j^{kl}}{\sum_{x,y}\exp\left(\frac{-E^{xy}}{T}\right)} \\ &\quad - \frac{\sum_l \exp\left(-E^l/T\right)\frac{1}{T}\sum_{r,s}\exp\left(-E^{rs}/T\right)a_i^{rs}a_j^{rs}}{\left(\sum_{r,s}\exp\left(-E^{rs}/T\right)\right)^2} \\ &= \frac{1}{T}[\sum_l p^-(\mathbf{v}^k \wedge \mathbf{h}^l)a_i^{kl}a_j^{kl} - p^-(\mathbf{v}^k)\sum_{r,s} p^-(\mathbf{v}^r \wedge \mathbf{h}^s)a_i^{rs}a_j^{rs}]\end{aligned} \tag{9}$$

Deriving the weight update

As a final piece of preparation, note that:

$$p^-(\mathbf{h}^l|\mathbf{v}^k) = p^+(\mathbf{h}^l|\mathbf{v}^k)$$

So...

$$p^-(\mathbf{v}^k \wedge \mathbf{h}^l)\frac{p^+(\mathbf{v}^k)}{p^-(\mathbf{v}^k)} = p^+(\mathbf{h}^l|\mathbf{v}^k) \tag{10}$$

Deriving the weight update

Now, finally, using equations 9 and 10 the partial of the loss becomes:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial W_{ij}} &= -\sum_k \frac{p^+(\mathbf{v}^k)}{p^-(\mathbf{v}^k)} \cdot \frac{\partial p^-(\mathbf{v}^k)}{\partial W_{ij}} \\
&= -\sum_k \frac{p^+(\mathbf{v}^k)}{p^-(\mathbf{v}^k)} \cdot \frac{1}{T}[\sum_l p^-(\mathbf{v}^k \wedge \mathbf{h}^l) a_i^{kl} a_j^{kl} \\
&- p^-(\mathbf{v}^k) \sum_{r,s} p^-(\mathbf{v}^r \wedge \mathbf{h}^s) a_i^{rs} a_j^{rs}] \\
&= \frac{1}{T}[\langle a_i a_j \rangle^+ - \langle a_i a_j \rangle^-]
\end{aligned} \tag{11}$$

where $\langle \cdot \rangle_{+/-}$ indicates the expected value during the "wake" versus "dreaming" states.

Deriving the weight update

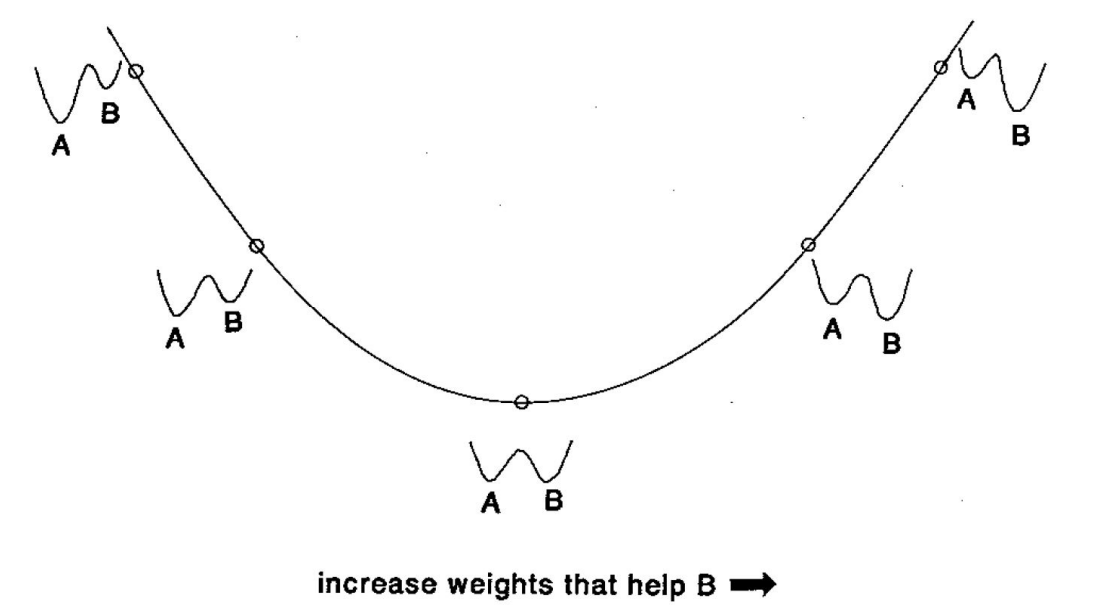
Thus, our weight update is:

$$\Delta W_{ij} = \eta \frac{1}{T}[\langle a_i a_j \rangle^+ - \langle a_i a_j \rangle^-] \tag{12}$$

This is Hebbian plasticity during the awake phase, and anti-Hebbian plasticity during the dreaming phase. Put another way: ***remember reality and forget your dreams***.

Deriving the weight update

From a statistical mechanics perspective, this effectively lowers the energy of states that are encountered during waking, and increases the energy of the states encountered during dreams.



Calculating the weight update

In order to calculate $\langle a_i a_j \rangle^{+/-}$ we need to be able to sample from the probability distributions $p^+(\mathbf{a})$ and $p^-(\mathbf{a})$. How can we do that?

Sampling from the Boltzmann machine

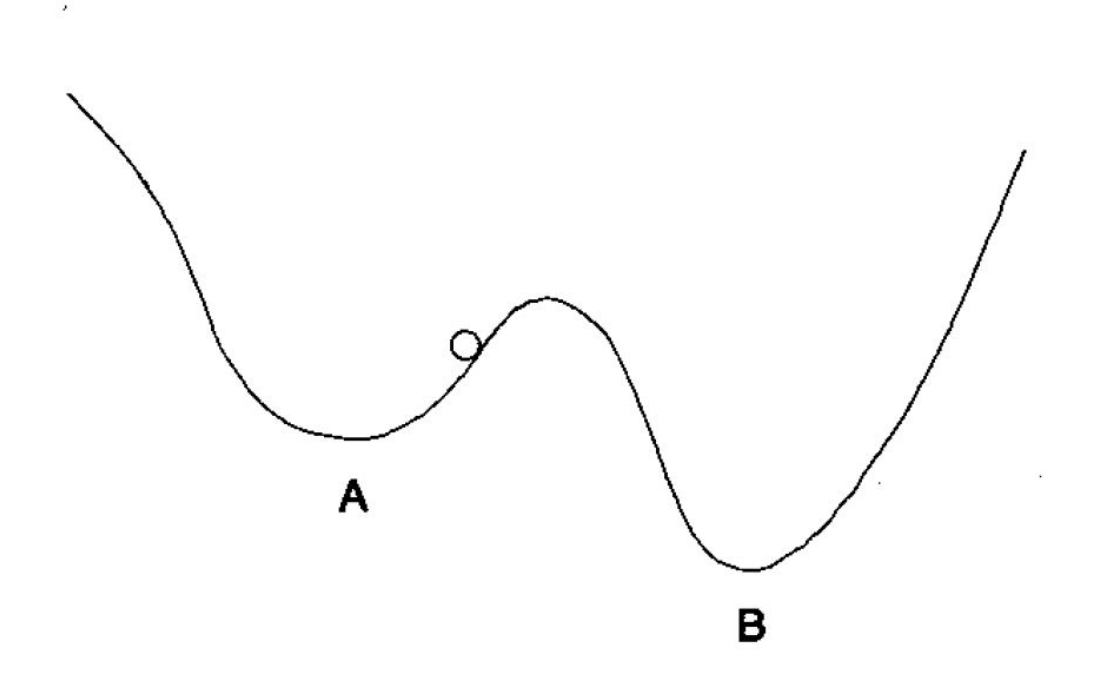
To sample from the Boltzmann machine distributions you need to use something called “Gibbs sampling”. This basically means:

- Select a neuron at random.
- Update its state (and only its state) according to equation 1.
- Keep doing this until the system is at equilibrium.

That last item is tough...

Sampling from the Boltzmann machine

To sample from the equilibrium distribution you need to ensure that the system has not gotten trapped in a local minima of the energy landscape. With infinite time and non-zero temperature, you're guaranteed to at some point, but what if you don't have infinite time?



Sampling from the Boltzmann machine

We can use something called “simulated annealing”, which is akin to real annealing that they do when making alloys to ensure that they get good mixtures. Basically, you heat it up, then slowly cool it down.

Sampling from the Boltzmann machine

We use simulated annealing with the temperature, T, in a Boltzmann machine to successfully sample from the equilibrium distributions for the wake and dream phases.

This requires the definition of an “annealing schedule” where you define a high temperature to start with, and then steadily reduce the temperature over time.

Doing so helps you to estimate your weight updates accurately.

Sampling from the Boltzmann machine

This last item is why Boltzmann machines are not used more widely. Training them is very inefficient, because the larger your Boltzmann machine the longer you have to do Gibbs sampling for to get to equilibrium.

Thus, Boltzmann machines do not scale to large problems, unfortunately.

Learning to hallucinate reality

They are such a cool idea though! Basically, you learn about the constraints in the real world by learning to hallucinate things that look like reality.

Learning to hallucinate reality

Perhaps this is indeed how our own brains work. Perhaps this is how we learn as babies.

But, we are still very far from being able to answer these sorts of questions in neuroscience...

Next week

No class on Tuesday!

A short quiz on Thursday!

Give the practice questions a try, and remember that anything covered in the lecture, or anything in the readings clearly related to the lecture materials is fair game.

Note: you do not need to memorize any equations.