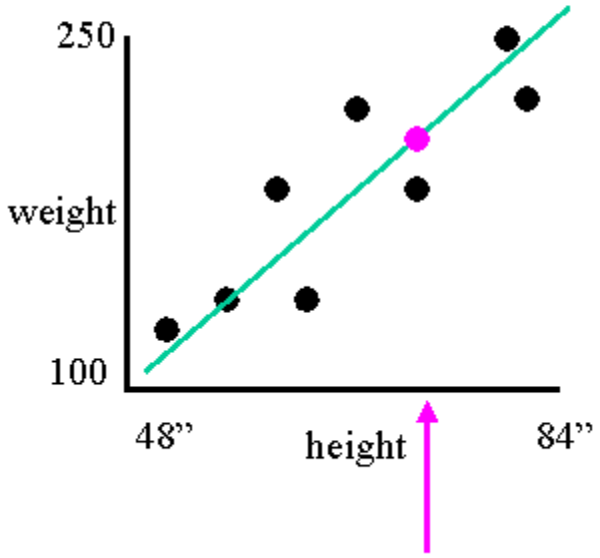
"Correlational" Learning

- Suppose we have a set of *samples* from some domain we're interested in.
  - People in this class: age, height, weight, etc.
- The set of variables (e.g., age, weight) is a vector of numbers (22, 30, 19…), (175, 130, 200)…
- We can compute the *correlation* between any of those vectors. Can *predict* one variable from another.

In Statistics...

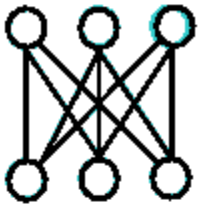


We gather *samples*

We can use a *regression* to predict one variable from another

For novel instances, use regression line to *guess* what predicted variable is.

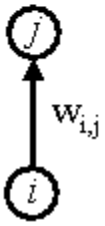
Learning:



- Decomposability of the problem: treat each output unit as a simpler problem to solve

So: how do we update the weights from input units to outputs?

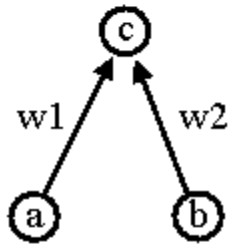
The Hebb Rule



$$\Delta w_{i,j} = \mu a_i a_j$$

- The change in weight from unit *i* to *j* is the product of the two units' activities, and a scaling factor *u*
  - If both units' activities are positive, or both negative, weight goes up
  - If signs are opposite, weight goes down

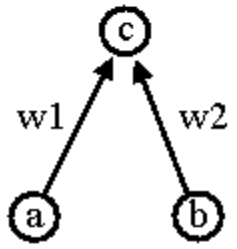
Example: c=a (u=0.25)



a	b	c	Δw1	Δw2
-1	-1	-1	.25	.25
1	-1	1	.25	-.25
-1	1	-1	.25	-.25
1	1	1	.25	.25
		Total	1.0	0.0

$$\Delta w_{i,j} = \mu a_i a_j$$

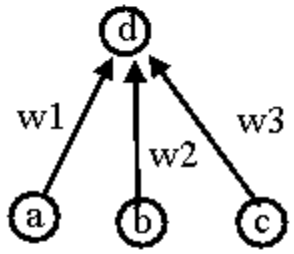
A Failure: c=a *or* b



a	b	c	Δw1	Δw2
-1	-1	-1	.25	.25
1	-1	1	.25	-.25
-1	1	1	-.25	.25
1	1	1	.25	.25
		Total	0.75	0.75

$$\Delta w_{i,j} = \mu a_i a_j$$

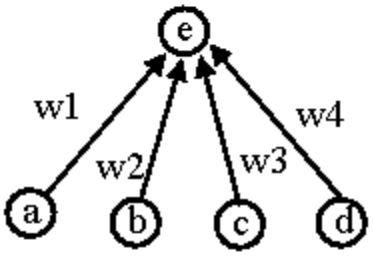
With Biases… d = a *or* b
(unit c as bias unit)



a	b	c	d	Δw1	Δw2	Δw3
-1	-1	1	-1	.25	.25	-.25
1	-1	1	1	.25	-.25	.25
-1	1	1	1	-.25	.25	.25
1	1	1	1	.25	.25	.25
			Total	0.75	0.75	0.75

$$\Delta w_{i,j} = \mu a_i a_j$$

A Real Failure



a	b	c	d	e	Δw1	Δw2	Δw3	Δw4
1	-1	1	-1	1	.25	-.25	.25	-.25
1	1	1	1	1	.25	.25	.25	.25
1	1	1	-1	-1	-.25	-.25	-.25	.25
1	-1	-1	1	-1	-.25	.25	.25	-.25
				Total	0.0	0.0	0.75	0.0

$$\Delta w_{i,j} = \mu a_i a_j$$

Properties of the Hebb Rule

$$\Delta w_{i,j} = \mu a_i a_j$$

- What if unit outputs are always positive?
- What happens over a long period of time?
- *Doesn't always work (even if solution exists).*
- The essential problem: weight value is *only* a function of behavior of units it connects.

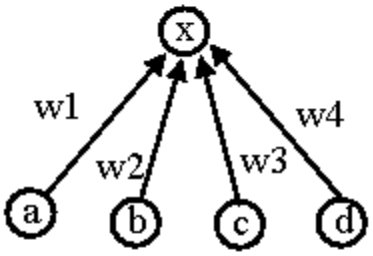
Something More Sophisticated: The Delta Rule

$$\Delta w_{i,j} = \mu e_i a_j$$

$$e_i = t_i - a_i$$

- Weight change a function of activity in the *from* unit (unit *j*), and the *error, e* on the *to* unit *i*.
- *This means:* weight updates are implicitly a function of the *overall* network behavior

The "Failure" Revisited



a	b	c	d	x	e	Δw1	Δw2	Δw3	Δw4
1	-1	1	-1	1	1	.25	-.25	.25	-.25
1	1	1	1	1	1	.25	.25	.25	.25
1	1	1	-1	-1	-2	-.5	-.5	-.5	.5
1	-1	-1	1	-1	-2	-.5	.5	.5	-.5
				Total		-.5	0.0	0.5	0.0

$$\Delta w_{i,j} = \mu e_i a_j$$

$$e_i = t_i - a_i$$

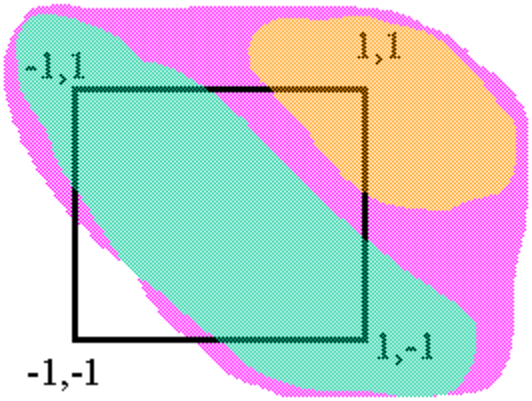
And so on… .

Properties of the Delta Rule

- Can be proven that it will converge to the weight vector that produces the global minimum possible sum squared error between targets and outputs.

Limitations

a	b	a or b	a and b	a xor b
-1	-1	-1	-1	-1
1	-1	1	-1	1
-1	1	1	-1	1
1	1	1	1	-1

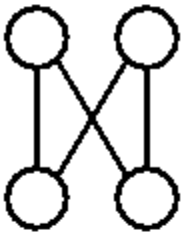


- Cannot solve problems that are not ***linearly separable*** (e.g., XOR)

I'm going to a Penguins game on Monday or Wednesday.

Implies that I'm not going to BOTH games!

A Pattern Associator



- Consists of a set of *input* units, and *output* units, and connections from input to output.
- .. And a training set of examples, consisting of inputs and their corresponding outputs

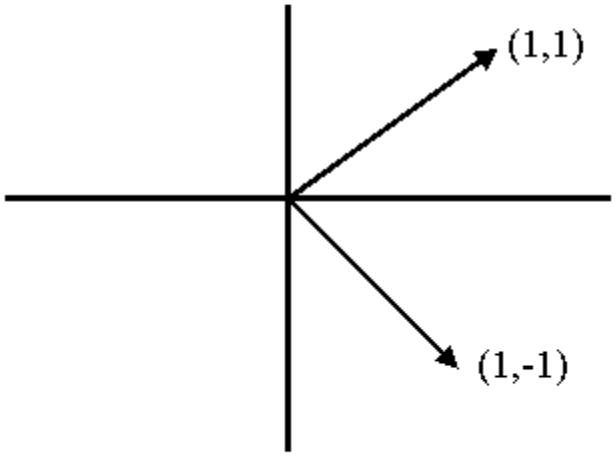
Simple Generalization In Hebbian Learning

$$
\begin{aligned}
w_{i,j} &= i_{j,l} o_{i,l} \mu \\
o_{i,t} &= \sum_j w_{i,j} i_{j,t} \\
&= \sum_j i_{j,l} o_{i,l} i_{j,t} \mu \\
&= o_{i,l} \mu \sum_j i_{j,t} i_{j,l}
\end{aligned}
$$

The Dot Product

- The sum of the products of elements of two vectors
- When normalized for length, is basically the *correlation* between the vectors
- The *angle* between the vectors is the inverse cosine of the dot product
- When the dot product is 0 (or, angle is 90 degrees), vectors are *orthogonal*

Geometrically...



So, Generalization in Hebb...

$$o_{i,l} \mu \sum_{j} i_{j,t} i_{j,l}$$

- After single learning trial, generalization is proportional to:
  - Output from trained trial, and
  - Correlation between new test input and learned input

After Multiple Training Trials..

$$o_{i,t} = k\sum_{l} o_{i,l}(i_{i,t} \cdot i_{i,l})$$

- Output of a test pattern is a function of the sum of all dot products between test input and all training input patterns, multiplied by the output that each test pattern produced.

Properties of Hebb Generalization

- If input is uncorrelated with *all* training inputs, output is zero
- Otherwise, is weighted average of all outputs from all training trials
  - Weighted by correlations with inputs on training trials
- If all training trials orthogonal to each other, no *cross-talk*

Cross-Talk in Delta Rule Learning

- Suppose we learn a given pattern in the delta rule
- What happens when we present a test pattern that is similar to that learned pattern?
- Difference in output is a function of error on the learned pattern, and dot product of learned input and test input

What Does This Mean?

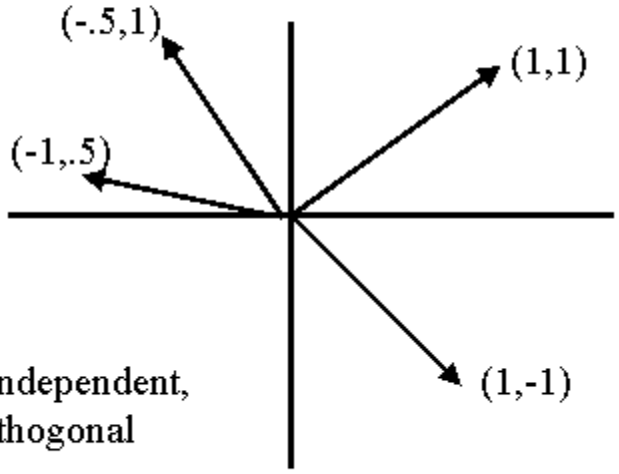
- When our new item is similar to what we've been trained on, learning is *easier* if the output we *want* is close to the output we *get* from other examples.
- So, *regular* items (ones that have similar input-output relationships) don't need a lot of training
- *Exceptions* need more training.

Constraints on Learning

- With Hebb rule, each training input needs to be *orthogonal* to every other one in order to be separable and avoid cross-talk
- With delta rule, the inputs just have to be *linearly independent* from each other to prevent one training trial from wrecking what was learned on other trials
  - Linearly independent: can't produce vector A by multiplying vector B by a scalar

More Geometry…

Orthogonal vectors



Linearly independent,
but not orthogonal

Different Types of Activation Functions

- Linear: output of a unit is simply the summed input to it
- Linear threshold: output of a unit is summed input, but not above or below a threshold
- Stochastic: Roll dice as to what output is based on input
- Sigmoid: 1/(1+exp(-net))

Delta Rule for Non-Linear Units

$$o_i = f(net_i)$$

$$net_i = \sum_j w_{i,j} o_j$$

$$E_i = (t_i - o_i)^2$$

$$\frac{\partial E_i}{\partial w_{i,j}} = \frac{\partial E_i}{\partial o_i} \frac{\partial o_i}{\partial net_i} \frac{\partial net_i}{\partial w_{i,j}}$$

For linear units,

$\frac{\partial o_i}{\partial net_i}$ Equals 1.

Otherwise, it's the derivative of our activation function f

So Delta Rule Works Well For Any Activation Function *f* that is *differentiable*

- Linear: easily differentiable
- Sigmoid: easily differentiable
- Threshold…. Not so much so.
- (what about other error functions besides sum-squared?)

Minimizing Sum Squared Error

- With unambiguous input, will converge to correct output
- With ambiguous or noisy input, will converge to output that minimizes average squared distance from all targets
  - This is effectively regression!
- Can read outputs as a probability distribution (recall IA Reading Model)