9. HEBBIAN LEARNING

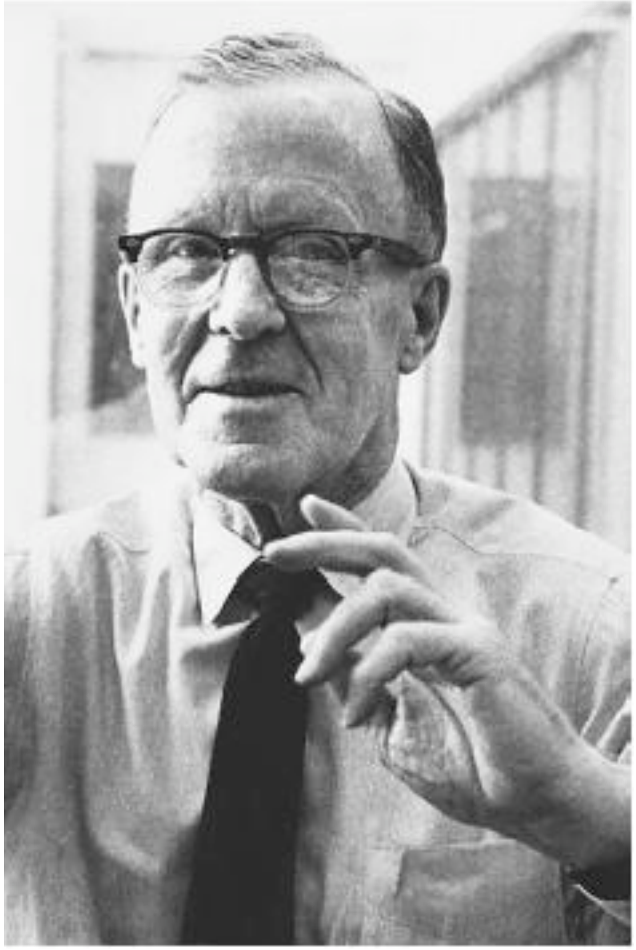
J. Elder

PSYC 6256 Principles of Neural Coding

Hebbian Learning

- "When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

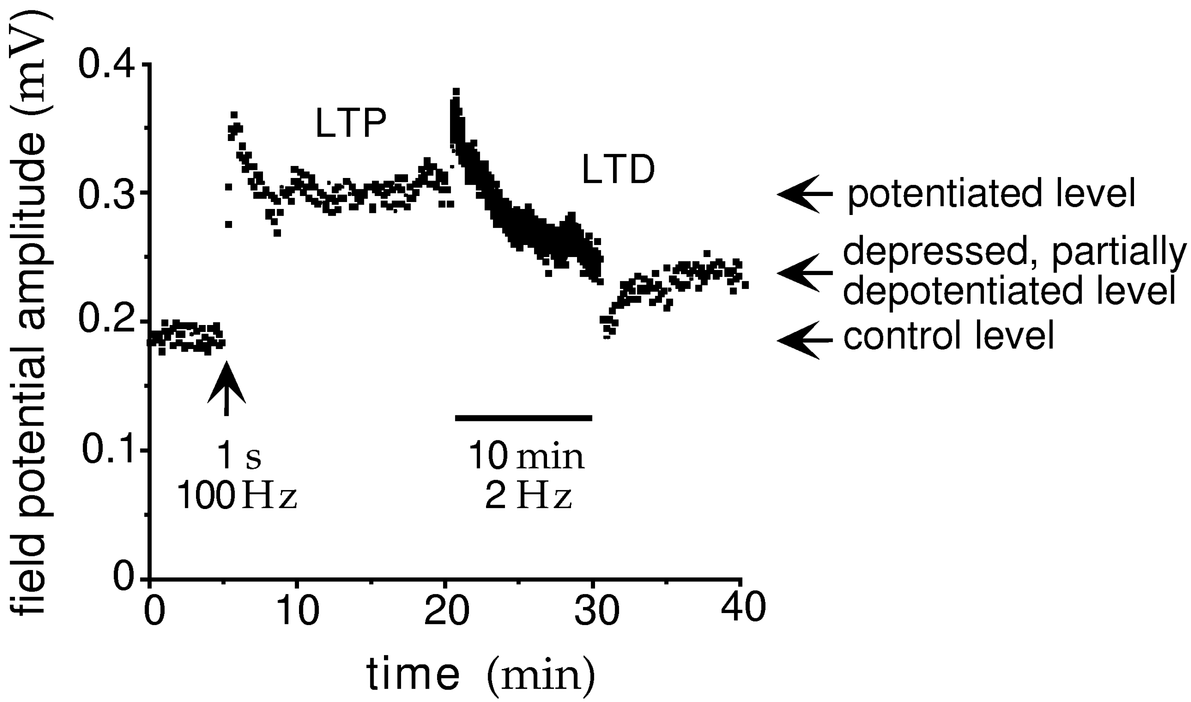
- This is often paraphrased as "Neurons that fire together wire together." It is commonly referred to as Hebb's Law.



Donald Hebb

Example: Rat Hippocampus

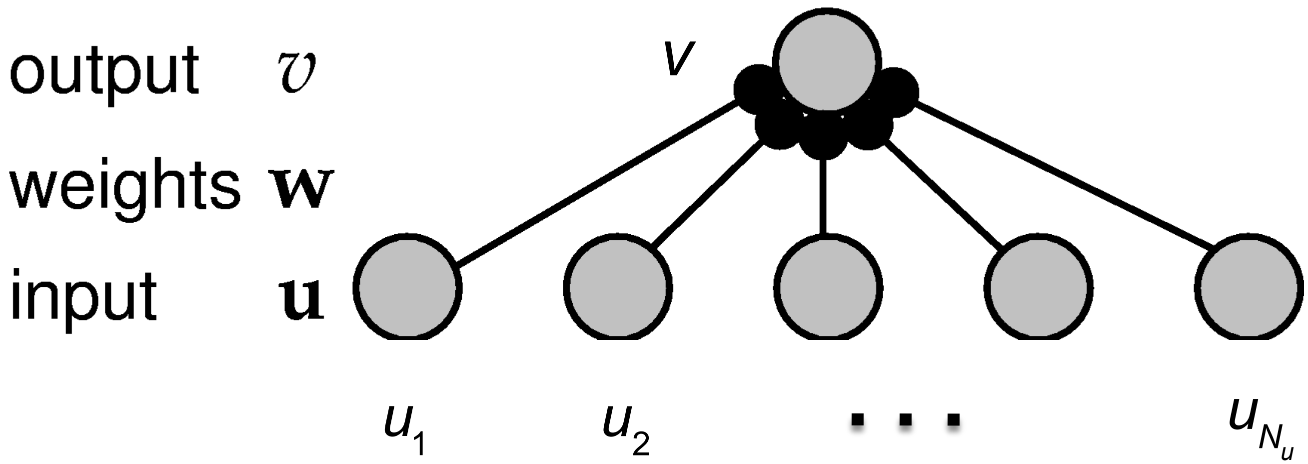
- Long-Term Potentiation
  - Increase in synaptic strength
- Long-Term Depression
  - Decrease in synaptic strength



Single Postsynaptic Neuron

Simplified Functional Model

$$v = \mathbf{w} \cdot \mathbf{u} = \mathbf{w}^t \mathbf{u}$$



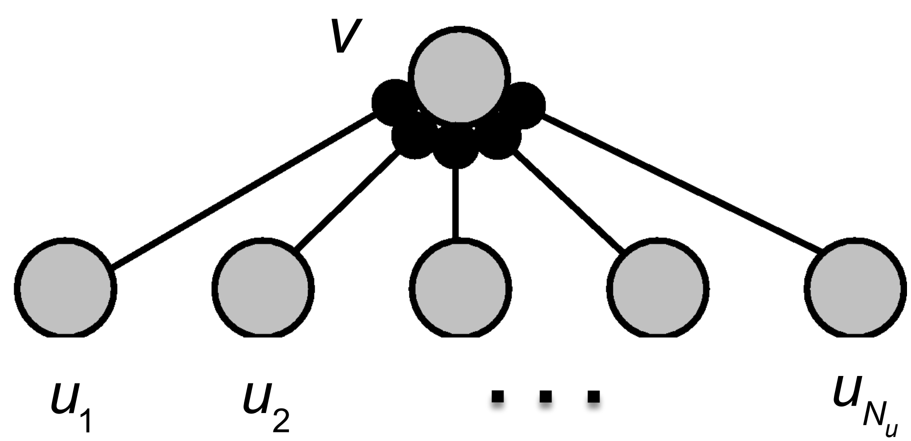
The Basic Hebb Rule

$$\tau_w \frac{d\mathbf{w}}{dt} = v\mathbf{u}$$

output v

weights $\mathbf{w}$

input $\mathbf{u}$



Modeling Hebbian Learning

- Hebbian learning can be modeled as either a continuous or a discrete process:
  - Continuous:

$$\tau_w \frac{d\mathbf{w}}{dt} = v\mathbf{u}$$

  - Discrete:

$$\mathbf{w} \rightarrow \mathbf{w} + \varepsilon v\mathbf{u}$$

Modeling Hebbian Learning

- Hebbian learning can be incremental or batch:
  - Incremental:

$$\tau_w \frac{d\mathbf{w}}{dt} = v\mathbf{u}$$

  - Batch:

$$\tau_w \frac{d\mathbf{w}}{dt} = \langle v\mathbf{u} \rangle$$

Batch Learning: The Average Hebb Rule

$$\tau_w \frac{d\mathbf{w}}{dt} = \langle v\mathbf{u} \rangle$$

$$\rightarrow \quad \tau_w \frac{d\mathbf{w}}{dt} = \mathbf{Q}\mathbf{w}$$

where $\mathbf{Q} = \langle \mathbf{u}\mathbf{u}^t \rangle$ is the input correlation matrix

Limitations of the Basic Hebb Rule

$$\tau_w \frac{d\mathbf{w}}{dt} = v\mathbf{u}$$

- If **u** and *v* are non-negative firing rates, models LTP but not LTD.

Limitations of the Basic Hebb Rule

$$\tau_w \frac{d\mathbf{w}}{dt} = v\mathbf{u}$$

- Even when input and output may be negative or positive, the positive feedback causes the magnitude of the weights to increase without bound:

$$\tau_w \frac{d\left|\mathbf{w}\right|^2}{dt} = 2v^2$$

The Covariance Rule

- Empirically,
  - LTP occurs if presynaptic activity is accompanied by high postsynaptic activity.
  - LTD occurs if presynaptic activity is accompanied by low postsynaptic activity.
  - This can be modeled by introducing a postsynaptic threshold θ_v that determines the sign of learning:

$$\tau_w \frac{d\mathbf{w}}{dt} = \left(v - \theta_v\right)\mathbf{u}$$

The Covariance Rule

- Alternatively, this can be modeled by introducing a presynaptic threshold θ_u :

$$\tau_w \frac{d\mathbf{w}}{dt} = v\left(\mathbf{u} - \theta_u\right)$$

The Covariance Rule

- In either case, a natural choice for the threshold is the average value of the input or output over the training period:

$$\tau_w \frac{d\mathbf{w}}{dt} = \left(v - \langle v \rangle\right)\mathbf{u}$$

or

$$\tau_w \frac{d\mathbf{w}}{dt} = v\left(\mathbf{u} - \langle \mathbf{u} \rangle\right)$$

The Covariance Rule

- Both models lead to the same batch learning rule:

$$\tau_w \frac{d\mathbf{w}}{dt} = \mathbf{C}\mathbf{w}$$

where $\mathbf{C} = \left(\mathbf{u} - \langle \mathbf{u} \rangle\right)\left(\mathbf{u} - \langle \mathbf{u} \rangle\right)^t$ is the input covariance matrix.

The Covariance Rule

- However note that the two rules are different at the incremental level:

$$\tau_w \frac{d\mathbf{w}}{dt} = (v - \theta_v)\mathbf{u} \rightarrow \text{Learning occurs only when there is presynaptic activity.}$$

$$\tau_w \frac{d\mathbf{w}}{dt} = v(\mathbf{u} - \theta_u) \rightarrow \text{Learning occurs only when there is postsynaptic activity.}$$

Problems with the Covariance Rule

- The covariance rule accounts for both LTP and LTD.
- But the positive feedback still causes the weight vector to grow without bound:

$$\tau_w \frac{d|\mathbf{w}|^2}{dt} = 2v\left(v - \langle v \rangle\right)$$

Thus

$$\left\langle \tau_w \frac{d|\mathbf{w}|^2}{dt} \right\rangle = 2\left\langle \left(v - \langle v \rangle\right)^2 \right\rangle \geq 0.$$

Synaptic Normalization

Synaptic Normalization

- Synaptic normalization constrains the magnitude of the weight vector to some value.
- Not only does this prevent the weights from growing without bound, it introduces competition between the input neurons: for one weight to grow, another must shrink.

Forms of Synaptic Normalization

- Rigid Constraint:
  - The constraint must hold at all times
- Dynamic Constraint:
  - The constraint must be satisfied asymptotically

Subtractive Normalization

- This is an example of a rigid constraint.
- Only works for non-negative weights.

$$\tau_w \frac{d\mathbf{w}}{dt} = v\mathbf{u} - \frac{v(\mathbf{n}\cdot\mathbf{u})\mathbf{n}}{N_u},$$

$$\text{where } \mathbf{n} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

$$\text{which satisfies } \tau_w \frac{d\mathbf{n}\cdot\mathbf{w}}{dt} = 0$$

Subtractive Normalization

$$\tau_w \frac{d\mathbf{w}}{dt} = v\mathbf{u} - \frac{v(\mathbf{n}\cdot\mathbf{u})\mathbf{n}}{N_u}$$

- Note that subtractive normalization is non-local.

Multiplicative Normalization: The Oja Rule

$$\tau_w \frac{d\mathbf{w}}{dt} = v\mathbf{u} - \alpha v^2 \mathbf{w}$$

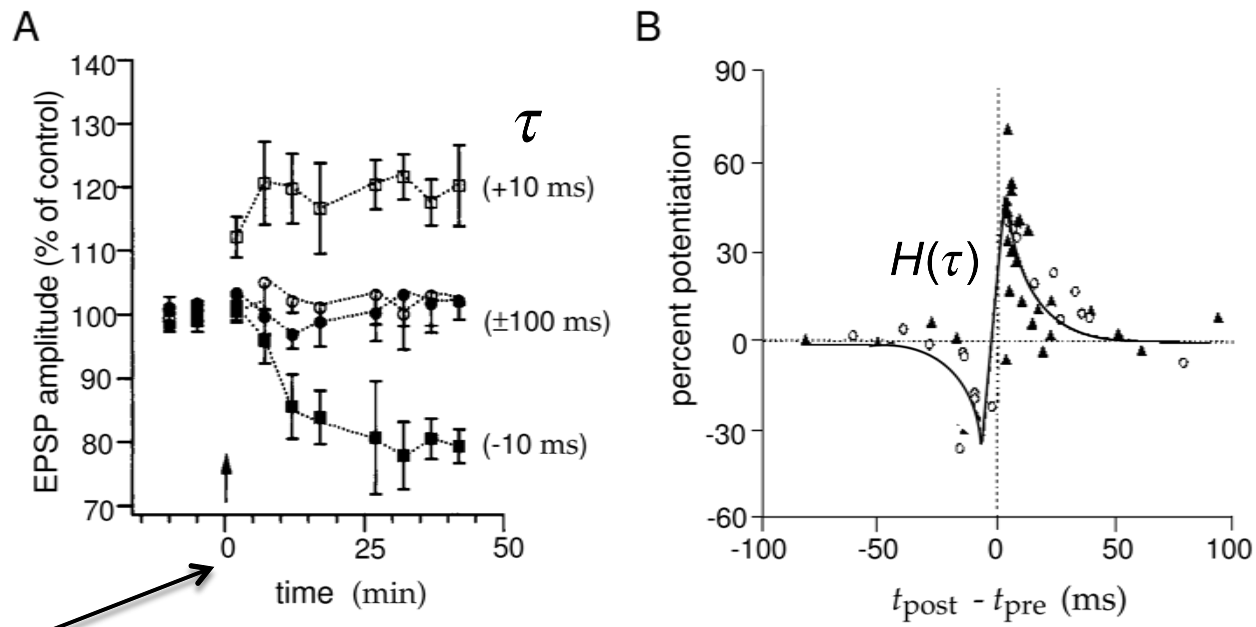
- Here the damping term is proportional to the weights.
- Works for positive and negative weights
- This is an example of a dynamic constraint:

$$\tau_w \frac{d\left|\mathbf{w}\right|^2}{dt} = 2v^2\left(1 - \alpha\left|\mathbf{w}\right|^2\right)$$

Timing-Based Rules

- A more accurate biophysical model will take into account the timing of presynaptic and postsynaptic spikes.

$$\tau_w \frac{d\mathbf{w}}{dt} = \int_0^\infty H(\tau)v(t)\mathbf{u}(t-\tau) + H(-\tau)v(t-\tau)\mathbf{u}(t)d\tau$$



paired stimulation of presynaptic and postsynaptic neurons

Steady State

Steady State

$$\tau_w \frac{d\mathbf{w}}{dt} = \mathbf{Q}\mathbf{w}$$

where $\mathbf{Q} = \langle \mathbf{u}\mathbf{u}^t \rangle$ is the input correlation matrix

- What do the weights converge to?

Let $\mathbf{e}_\mu$ be the eigenvectors of $\mathbf{Q}$, $\mu = 1, 2, \ldots, N_u$, satisfying

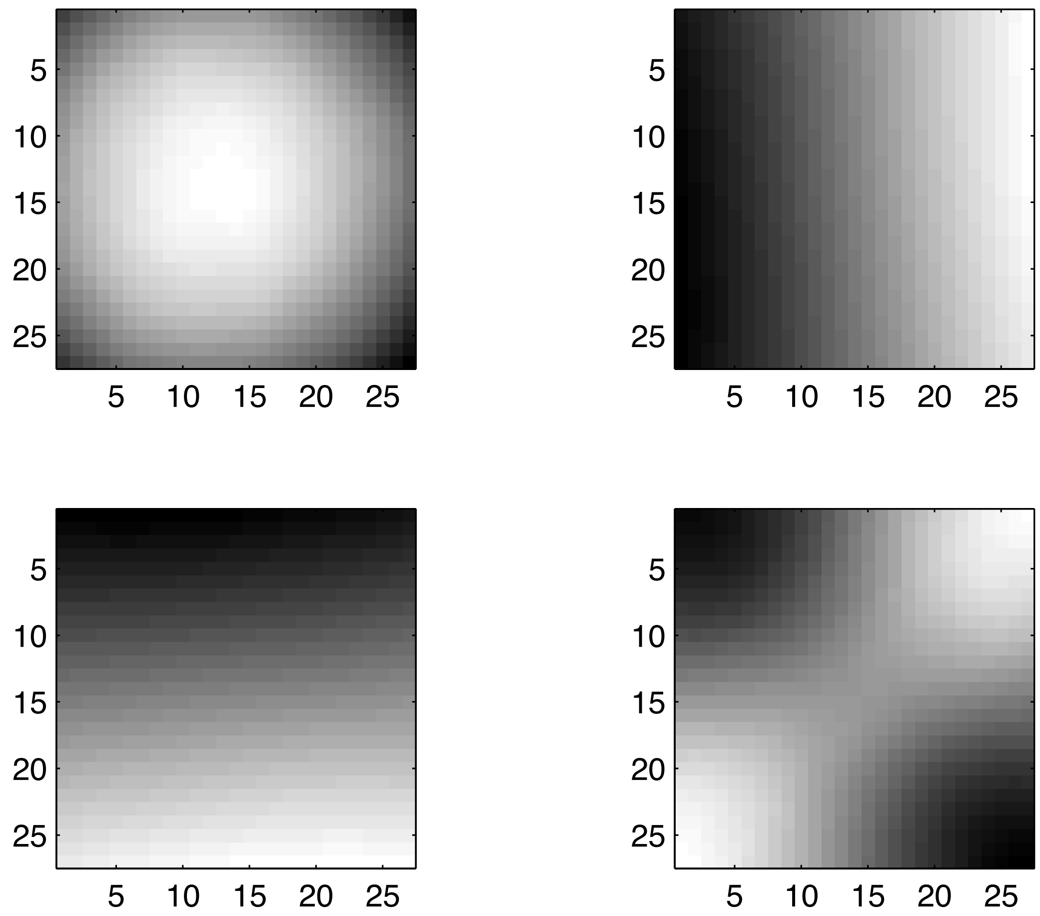
$$\mathbf{Q}\mathbf{e}_\mu = \lambda_\mu \mathbf{e}_\mu$$

$$\lambda_1 \geq \lambda_2 \geq \cdots \lambda_{N_u}$$

Eigenvectors


```
function lgneig(lgnims,neigs,nit)
%Computes and plots first neigs eigenimages of LGN inputs to V1
%lgnims = cell array of images representing normalized LGN output
%nit = number of image patches on which to base estimate

dx=1.5; %pixel size in arcmin.  This is arbitrary.
v1rad=round(10/dx); %V1 cell radius (pixels)
Nu=(2*v1rad+1)^2; %Number of input units

nim=length(lgnims);

Q=zeros(Nu);
for i=1:nit

    u=im(y-v1rad:y+v1rad,x-v1rad:x+v1rad);
    u=u(:);

    Q=Q+u*u'; %Form autocorrelation matrix
end
Q=Q/Nu; %normalize
[v,d]=eigs(Q,neigs); %compute eigenvectors
```

Output



5
10
15
20
25
5 10 15 20 25
5
10
15
20
25
5 10 15 20 25
5
10
15
20
25
5 10 15 20 25
5
10
15
20
25
5 10 15 20 25

Steady State

$$\tau_w \frac{d\mathbf{w}}{dt} = \mathbf{Q}\mathbf{w}$$

- Now we can express the weight vector in the eigenvector basis:

$$\mathbf{w}(t) = \sum_{\mu=1}^{N_\mu} c_\mu(t)\mathbf{e}_\mu$$

- Substituting into the Hebb rule, we get

$$\tau_w \sum_{\mu=1}^{N_\mu} \frac{dc_\mu(t)}{dt}\mathbf{e}_\mu = \mathbf{Q}\sum_{\mu=1}^{N_\mu} c_\mu(t)\mathbf{e}_\mu$$

$$\rightarrow \tau_w \frac{dc_\mu(t)}{dt} = \lambda_\mu c_\mu(t) \qquad \rightarrow c_\mu(t) = a_\mu \exp\left(\left(\lambda_\mu / \tau_w\right)t\right)$$

$$\text{and thus } \mathbf{w}(t) = \sum_{\mu=1}^{N_\mu} a_\mu \exp\left(\left(\lambda_\mu / \tau_w\right)t\right)\mathbf{e}_\mu$$

Steady State

$$\mathbf{w}(t) = \sum_{\mu=1}^{N_\mu} a_\mu \exp\left(\left(\lambda_\mu / \tau_w\right)t\right)\mathbf{e}_\mu$$

As $t \to \infty$, the term with the largest eigenvalue λ_μ dominates, so that

$$\lim_{t\to\infty} \mathbf{w}(t) \propto \mathbf{e}_1$$

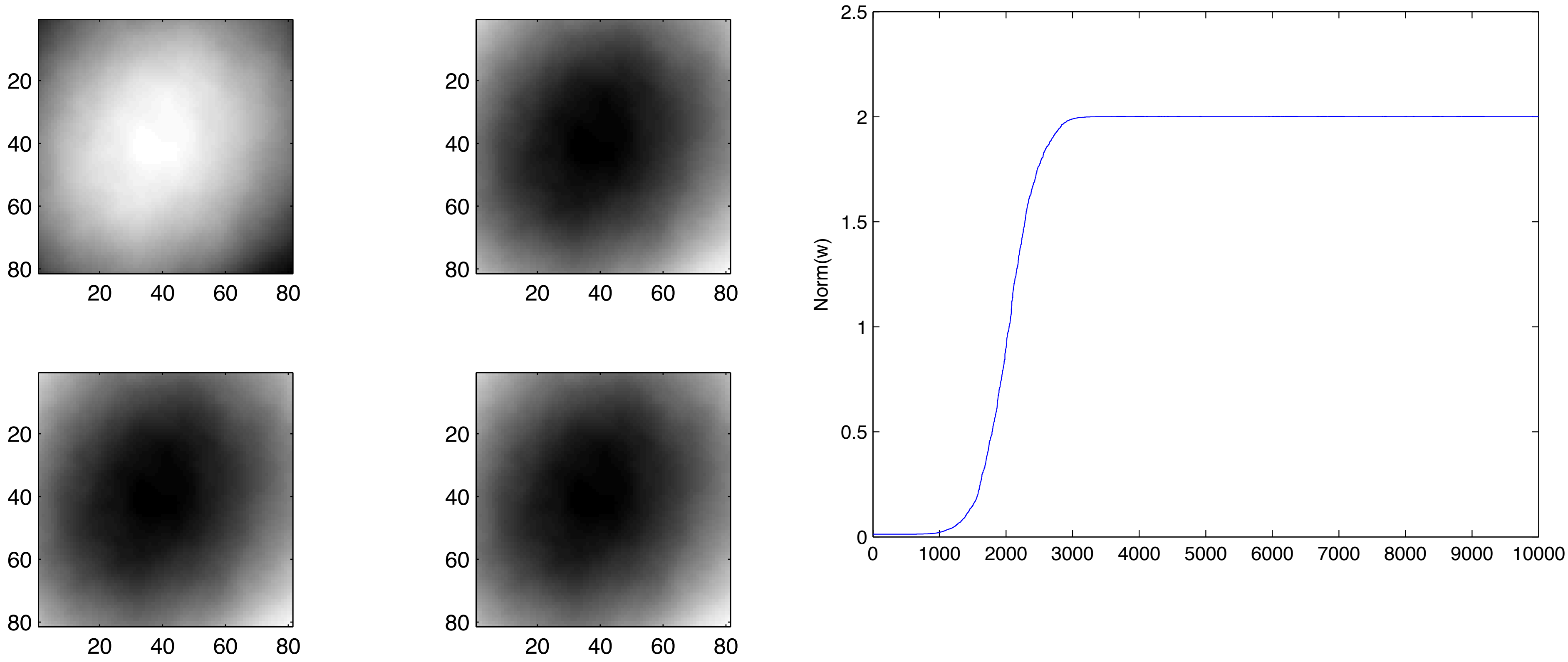
- For the simple form of the Hebb rule, the weights grow without bound.
- If the Oja rule is employed, it can be shown that

$$\lim_{t\to\infty} \mathbf{w}(t) = \mathbf{e}_1 / \sqrt{\alpha}$$

Hebbian Learning (Feedforward)


```
function hebb(lgnims,nv1cells,nit)
%Implements a version of Hebbian learning with the Oja rule, running on simulated LGN
%inputs from natural images.

%lgnims = cell array of images representing normalized LGN output
%nv1cells = number of V1 cells to simulate
%nit = number of learning iterations

dx=1.5; %pixel size in arcmin.  This is arbitrary.
v1rad=round(60/dx); %V1 cell radius
Nu=(2*v1rad+1)^2; %Number of input units
tauw=1e+6; %learning time constant

nim=length(lgnims);

w=normrnd(0,1/Nu,nv1cells,Nu); %random initial weights

for i=1:nit
   u=im(y-v1rad:y+v1rad,x-v1rad:x+v1rad);
   u=u(:);

   %See Dayan Section 8.2
   v=w*u; %Output

   %update feedforward weights using Hebbian learning with Oja rule
   w=w+(1/tauw)*(v*u'-repmat(v.^2,1,Nu).*w);
end
```

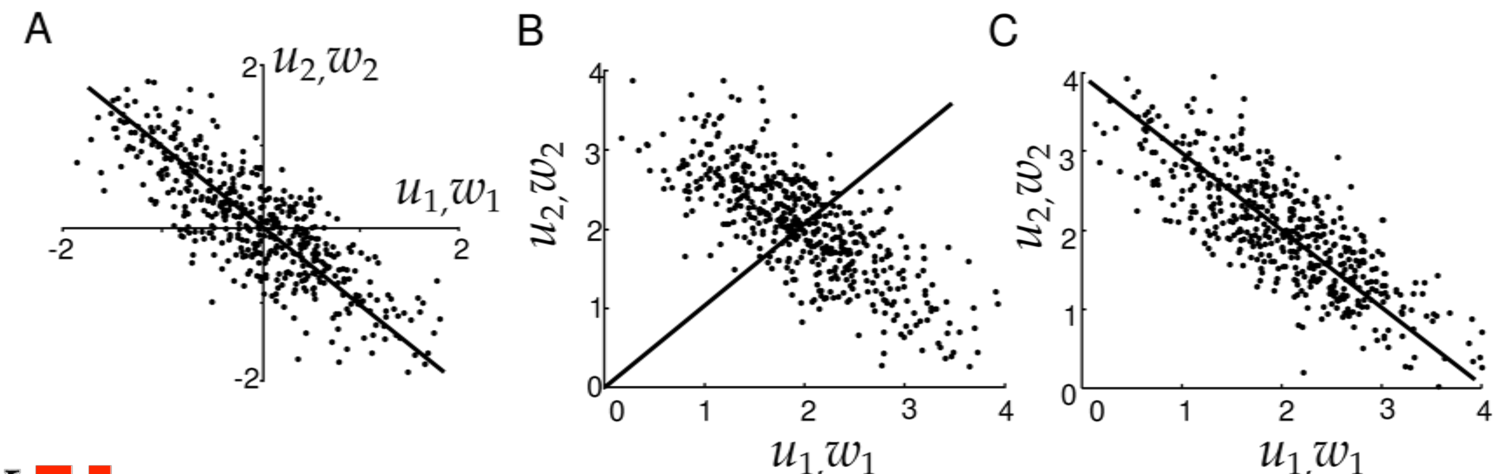
Output


20
40
60
80
20
40
60
80
2.5
2
1.5
1
0.5
0
Norm(w)
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

Steady State

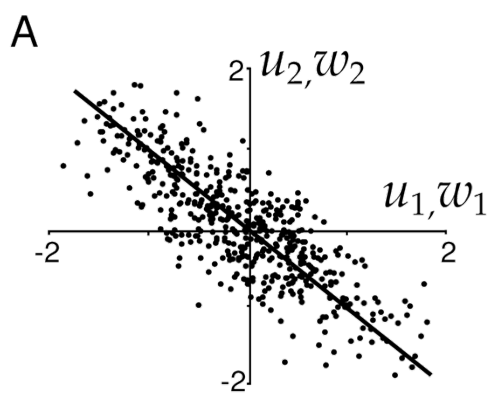
- Thus the Hebb rule leads to a receptive field representing the first principal component of the input.
- There are several reasons why this is a good computational choice.

Correlation Rule | Correlation Rule | Covariance Rule



Optimal Coding

- Coding/Decoding
  - Suppose the job of the neuron is to encode the input vector *u* as accurately as possible with a single scalar ouput *v*.
  - The choice $v = \mathbf{u} \cdot \mathbf{e}_1$ is optimal in the sense that the estimate $\hat{\mathbf{u}} = v\mathbf{e}_1$ minimizes the expected squared error over all possible receptive fields.



Information Theory

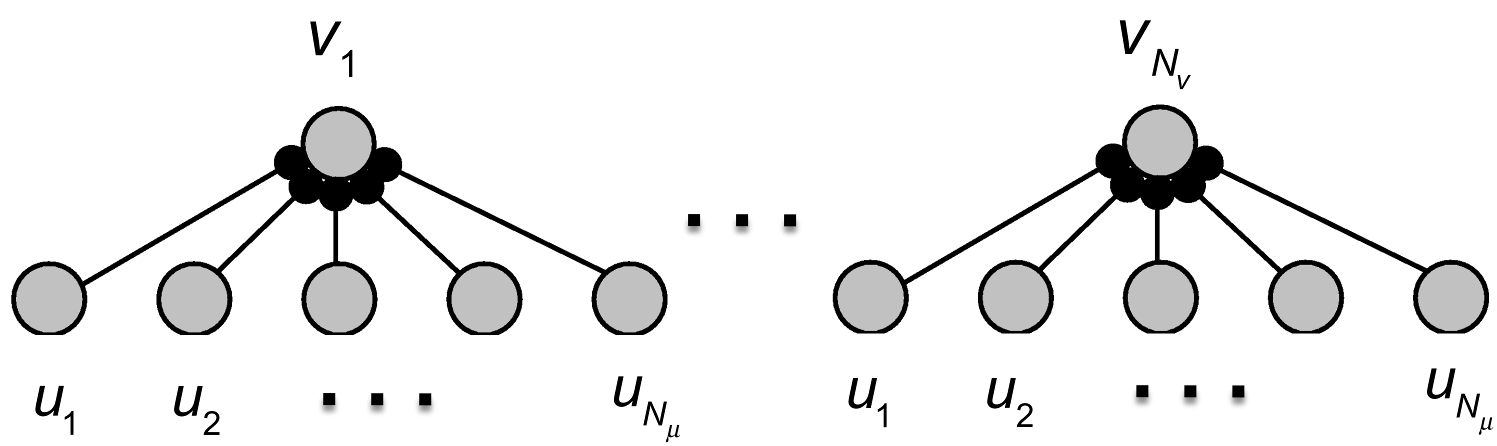
$$v = \mathbf{u} \cdot \mathbf{e}_1$$

- Projection of the input onto the first principal component of the input correlation matrix maximizes the variance of the output.
- For a Gaussian input, this also maximizes the output entropy.
- If the output is corrupted by Gaussian noise, this also maximizes the information the output v carries about the input $\mathbf{u}$.

Multiple Postsynaptic Neurons

Clones

- If we simply replicate the architecture for the single postsynaptic neuron model, each postsynaptic neuron will learn the same receptive field (the first principal component)



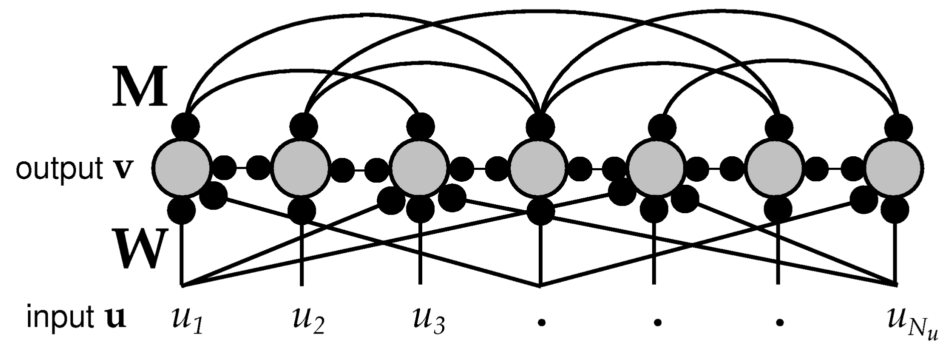
Competition Between Output Neurons

- In order to achieve diversity, we must incorporate some form of competition between output neurons.
- The basic idea is for each output neuron to inhibit the others when it is responding well. In this way it 'takes ownership' of certain inputs.
- This leads to diversification in receptive fields.
- This inhibition is achieved through recurrent connections between output neurons

Reccurent Network

$$\mathbf{v} = \mathbf{W}\mathbf{u} + \mathbf{M}\mathbf{v}$$

$$\rightarrow \mathbf{v} = \mathbf{K}\mathbf{W}\mathbf{u}, \text{ where } \mathbf{K} = (\mathbf{I} - \mathbf{M})^{-1}$$



- **W** is the feedforward weight matrix:

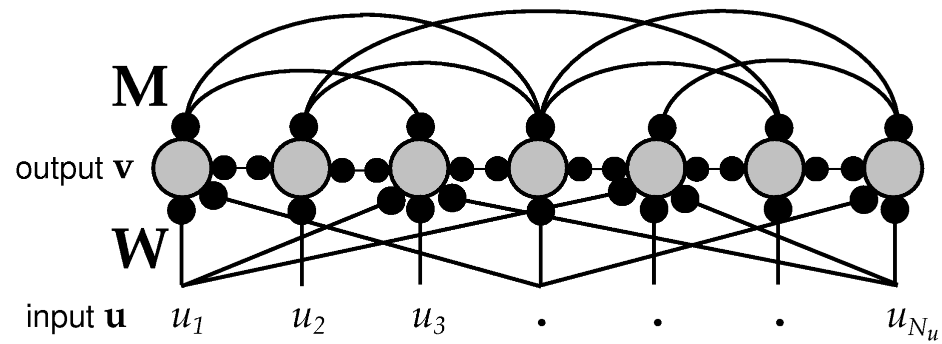
  $\mathbf{W}_{ab}$ is the strength of the synapse from input neuron *b* to output neuron *a*.

- **M** is the recurrent weight matrix:

  $\mathbf{M}_{aa'}$ is the strength of the synapse from output neuron *a′* to output neuron *a*.

Combining Hebbian and Anti-Hebbian Learning

(Foldiak, 1989)



- Feedforward weights can be learned as before through normalized Hebbian learning with the Oja rule:

$$\tau_w \frac{dW_{ab}}{dt} = v_a u_b - \alpha v_a^2 W_{ab}$$

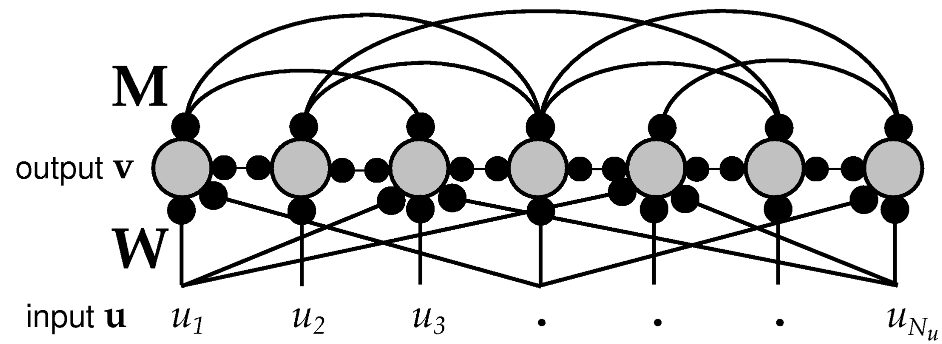
- Inhibitory reccurent weights can be learned concurrently through an anti-Hebbian rule:

$$\tau_M \frac{dM_{aa'}}{dt} = -v_a v_{a'}$$

where the weights are constrained to be non-positive: $M_{aa'} \leq 0$

Steady State

(Foldiak, 1989)



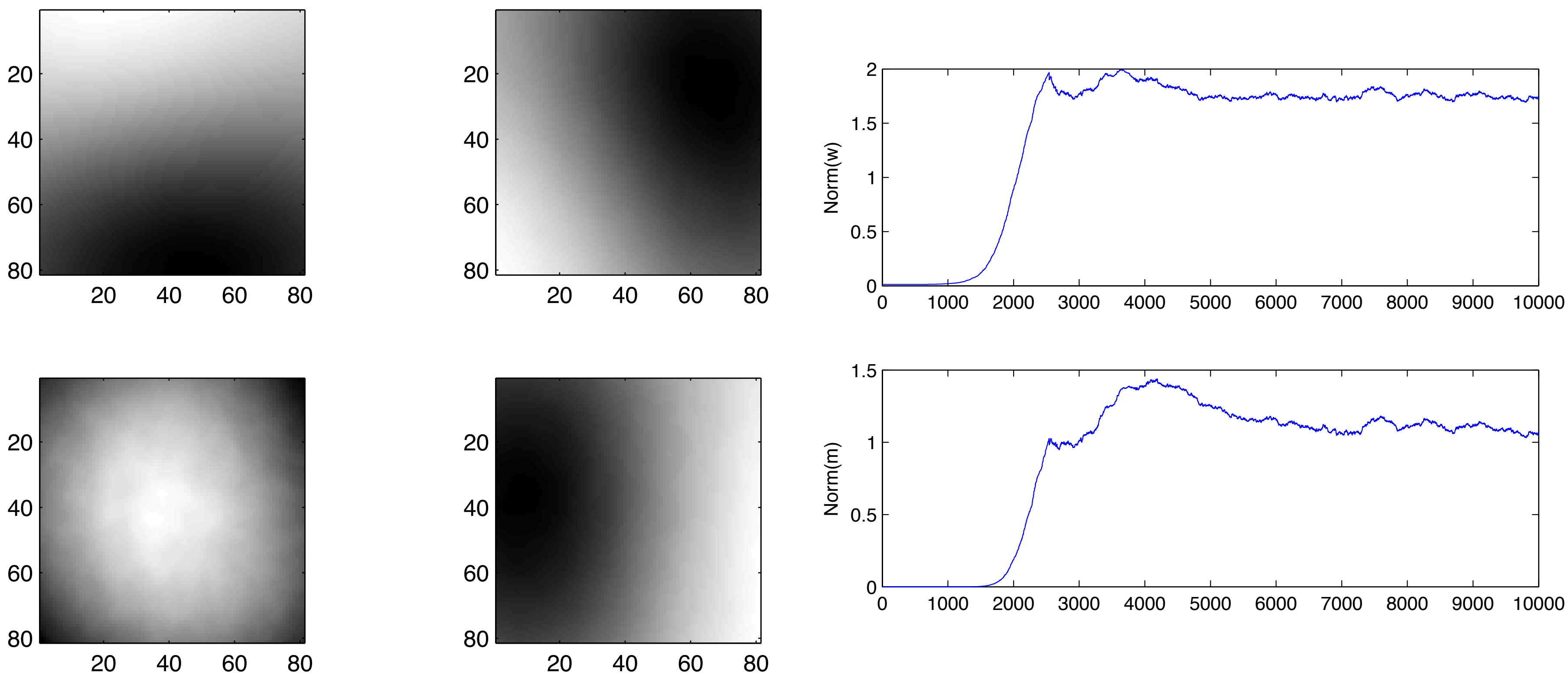
$$\tau_w \frac{dW_{ab}}{dt} = v_a u_b - \alpha v_a^2 W_{ab} \qquad \tau_M \frac{dM_{aa'}}{dt} = -v_a v_{a'}$$

- It can be shown that, with appropriate parameters, the weights will converge so that:
  - The rows of **W** are the eigenvectors of the correlation matrix **Q**
  - **M** = *0*

Hebbian Learning (With Recurrence)


```
function hebbfoldiak(lgnims,nv1cells,nit)
%Implements a version of Foldiak's 1989 network, running on simulated LGN
%inputs from natural images.  Incorporates feedforward Hebbian learning and
%recurrent inhibitory anti-Hebbian learning.

%lgnims = cell array of images representing normalized LGN output
%nv1cells = number of V1 cells to simulate
%nit = number of learning iterations

dx=1.5; %pixel size in arcmin.  This is arbitrary.
v1rad=round(60/dx); %V1 cell radius (pixels)
Nu=(2*v1rad+1)^2; %Number of input units
tauw=1e+6; %feedforward learning time constant
taum=1e+6; %recurrent learning time constant

zdiag=(1-eye(nv1cells)); %All 1s but 0 on the diagonal

w=normrnd(0,1/Nu,nv1cells,Nu); %random initial feedforward weights
m=zeros(nv1cells);

for i=1:nit
    u=im(y-v1rad:y+v1rad,x-v1rad:x+v1rad);
    u=u(:);

    %See Dayan pp 301-302, 309-310 and Foldiak 1989
    k=inv(eye(nv1cells)-m);
    v=k*w*u; %steady-state output for this input

    %update feedforward weights using Hebbian learning with Oja rule
    w=w+(1/tauw)*(v*u'-repmat(v.^2,1,Nu).*w);

    %update inhibitory recurrent weights using anti-Hebbian learning
    m=min(0,m+zdiag.*((1/taum)*(-v*v')));
end
```

Output


20
40
60
80
20
40
60
80
20
40
60
80
20
40
60
80
Norm(w)
2
1.5
1
0.5
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Norm(m)
1.5
1
0.5
0