Functional Models of Plasticity

Angus Chadwick

School of Informatics, University of Edinburgh, UK

Computational Neuroscience (Lecture 14, 2023/2024)
Outline of Lecture

• Hebb’s rule

• The covariance rule

• Oja’s Rule

• Synaptic normalisation

• Learning with multiple neurons
Functional Models of Plasticity

• What does Hebbian learning actually do?

• Can we use Hebbian learning to do something useful?

• What are the challenges/obstacles to implementing Hebb’s rule?

• Are there parallels between unsupervised learning algorithms and learning rules in the brain?

• What are the underlying algorithms/computations that synaptic plasticity implements, and how do they perform learning and memory?
Hebb’s Rule

• How can we operationalise Hebbian learning?
• “Cells that fire together, wire together”, is not a mathematically precise statement - there are many possibilities

• Simplest choice:
  - Consider one neuron with firing rate $y$ in response to multiple inputs with rates $x$

Linear model of neural activity

$$y = \sum_{i=1}^{N} w_i x_i = \mathbf{w} \cdot \mathbf{x}$$
Hebb’s Rule

• How can we operationalise Hebbian learning?
• “Cells that fire together, wire together”, is not a mathematically precise statement - there are many possibilities

• Simplest choice:
  - Consider one neuron with firing rate \( y \) in response to multiple inputs with rates \( x \)
  - Assume neuron is linear and weight update is multiplicative (pre \( x \) post)

![Linear model of neural activity]

\[
y = \sum_{i=1}^{N} w_i x_i = \mathbf{w} \cdot \mathbf{x}
\]

![Multiplicative model of Hebb’s rule]

\[
\Delta w_i = \epsilon y x_i
\]
Hebb’s Rule

• How can we operationalise Hebbian learning?
• “Cells that fire together, wire together”, is not a mathematically precise statement - there are many possibilities

• Simplest choice:
  - Consider one neuron with firing rate $y$ in response to multiple inputs with rates $x$
  - Assume neuron is linear and weight update is multiplicative (pre x post)

Linear model of neural activity

$$y = \sum_{i=1}^{N} w_i x_i = \mathbf{w} \cdot \mathbf{x}$$

Multiplicative model of Hebb’s rule

$$\Delta w_i = \epsilon y x_i$$

$$= \epsilon x_i \sum_{j=1}^{N} w_j \cdot x_j$$
Consequences of Hebb’s Rule

• Assume we present $M$ input patterns $\mu$ once each:

$$\Delta w_i = \epsilon \sum_{\mu=1}^{M} x^\mu_i \sum_{j=1}^{N} w_j x^\mu_j$$
Consequences of Hebb’s Rule

- Assume we present $M$ input patterns $\mu$ once each:

$$\Delta w_i = \epsilon \sum_{\mu=1}^{M} x_i^{\mu} \sum_{j=1}^{N} w_j x_j^{\mu}$$

- Define $Q_{ij} = \sum_{\mu} x_i^{\mu} x_j^{\mu}$, which is a kind of correlation between inputs:

$$\Delta w = \epsilon Q \cdot w$$
Consequences of Hebb’s Rule

• Assume we present $M$ input patterns $\mu$ once each:

$$
\Delta w_i = \varepsilon \sum_{\mu=1}^{M} x_i^\mu \sum_{j=1}^{N} w_j x_j^\mu
$$

• Define $Q_{i,j} = \sum_{\mu} x_i^\mu x_j^\mu$, which is a kind of correlation between inputs:

$$
\Delta w = \varepsilon Q.w
$$

• Or we can write in continuous time:

$$
\tau \frac{dw}{dt} = Q.w
$$
Assumptions/Approximations of Hebb’s Rule

• We have made several unrealistic assumptions and approximations:

  - Linearity of output neuron
  - Weights can change sign with learning
  - Weight updates are linear/multiplicative
  - Can only produce LTP (not LTD) if firing rates are positive
  - Weight updates are unbounded/independent of the weight itself (ignoring saturation etc.)

• These give the plasticity rule undesirable properties, as we will see
Long-Term Behaviour of Hebb’s Rule

• Hebb’s rule follows the differential equation:
  \[
  \tau \frac{dw}{dt} = Q \cdot w
  \]

• This is a kind of linear dynamical system, which we studied previously. It has solution:
  \[
  w(t) = \sum_{k} c_k w_k e^{\lambda_k t / \tau}
  \]

• Where \( w_k, \lambda_k \) are eigenvectors and eigenvalues of \( Q \).
Long-Term Behaviour of Hebb’s Rule

• Hebb’s rule follows the differential equation:

\[
\frac{dw}{dt} = Q \cdot w
\]

• This is a kind of linear dynamical system, which we studied previously. It has solution:

\[
w(t) = \sum_{k} c_k w_k e^{\frac{\lambda_k t}{\tau}}
\]

• Where \( w_k \), \( \lambda_k \) are eigenvectors and eigenvalues of \( Q \).

• But \( Q \) is symmetric and therefore has positive real eigenvalues, so all terms must grow exponentially

• Hebb’s rule is therefore unstable, and always leads to exponentially growing weights
The Covariance Rule

- If firing rates are positive, Hebb’s rule can only generate LTP, not LTD...
- Perhaps synapses only update when activity is above a certain threshold:

\[
\Delta w_i = \epsilon (x_i - \langle x_i \rangle) (y - \langle y \rangle)
\]

\[
\langle x_i \rangle = \frac{1}{M} \sum_{\mu} x_i^\mu
\]
The Covariance Rule

• If firing rates are positive, Hebb’s rule can only generate LTP, not LTD...
• Perhaps synapses only update when activity is above a certain threshold:

\[ \Delta w_i = \epsilon (x_i - \langle x_i \rangle)(y - \langle y \rangle) \]

\[ \langle x_i \rangle = \frac{1}{M} \sum_{\mu} x^\mu_i \]

• Averaging over patterns, this gives the same rule as before but with \( Q \) now the covariance matrix of input patterns:

\[ \tau \frac{dw}{dt} = Q \cdot w \]

\[ Q_{ij} = \sum_{\mu} (x^\mu_i - \langle x \rangle)(x^\mu_j - \langle x \rangle) \]
What does Hebb’s/Covariance Rule Learn?

• In the long run limit, all terms in the sum grow to infinity, but the term with largest eigenvalue dominates:

\[ w(t) = \sum_{k} c_k w_k e^{\lambda_k t} \xrightarrow{t \to \infty} c_1 w_1 e^{\lambda_1 t} \]

• In other words, this learning rule picks out the eigenvector of the matrix \( Q \) with largest eigenvalue

• The interpretation depends on the matrix \( Q \) (different for Hebb vs covariance rule)
What does Hebb’s/Covariance Rule Learn?

- If data aren’t zero mean, Hebb’s rule is sensitive to the mean
What does Hebb’s/Covariance Rule Learn?

- If data aren’t zero mean, Hebb’s rule is sensitive to the mean
- Covariance rule picks out largest eigenvector of input covariance matrix
- This is just principal component analysis (but with only one PC)
Hebbian Learning of Orientation Tuning

- Hebbian learning in a neuron receiving multiple LGN ON-OFF receptive field inputs
- Requires some special constraints and assumptions, but can learn Gabor receptive fields

Receptive fields learned via Hebbian plasticity

Miller, 1994 (see Dayan and Abbott Ch. 8)
Summary of Hebb/Covariance Rule

• Both Hebb’s rule and covariance rule are unstable, leading to exponentially growing weights

• Hebb’s rule can only produce LTP, but covariance rule can produce both LTP and LTD

• Both rules cause the neuron to learn the dominant eigenvector of $Q$ (but $Q$ is slightly different for the two rules)

• A major limitation of both rules is the lack of stability/competition between synapses (all synapses update independently and grow to infinity)
Normalisation

- We saw that Hebb/covariance rule leads to infinite weights
- In reality, weights must saturate/be regulated somehow
- Simplest choice: impose a hard limit
Normalisation

• We saw that Hebb/covariance rule leads to infinite weights
• In reality, weights must saturate/be regulated somehow
• Simplest choice: impose a hard limit

• For two inputs with anticorrelated $Q$, can produce 3 stable weight configurations depending on initial conditions
Normalisation

- We saw that Hebb/covariance rule leads to infinite weights
- In reality, weights must saturate/be regulated somehow
- Simplest choice: impose a hard limit

- For two inputs with anticorrelated $Q$, can produce 3 stable weight configurations depending on initial conditions
- For positively correlated input $Q$, both weights must saturate (not shown)
Multiplicative and Subtractive Normalisation

• Instead of a hard bound, add a term to the weight update to scale down weights over time

\[
\tau \frac{dw}{dt} = Q.w - \gamma(w)w(t)
\]
Multiplicative and Subtractive Normalisation

• Instead of a hard bound, add a term to the weight update to scale down weights over time

• Simplest options: multiplicative or subtractive scaling

\[ \tau \frac{dw}{dt} = Qw - \gamma(w)w(t) = Qw - \left[ \frac{n.Q.w}{n.w} \right]w \] Multiplicative

\[ n = (1, 1, 1, ...) \]
Multiplicative and Subtractive Normalisation

- Instead of a hard bound, add a term to the weight update to scale down weights over time

- Simplest options: multiplicative or subtractive scaling

\[
\tau \frac{dw}{dt} = Q \cdot w - \gamma(w)w(t) = Q \cdot w - \left[ \frac{n \cdot Q \cdot w}{n \cdot w} \right] w \quad \text{Multiplicative}
\]

\[
= Q \cdot w - \epsilon(w)n = Q \cdot w - \left[ \frac{n \cdot Q \cdot w}{n \cdot n} \right] n \quad \text{Subtractive}
\]

\[
n = (1, 1, 1, \ldots)
\]
Divisive and Subtractive Normalisation

• Both multiplicative and subtractive keep sum of weights constant in time (easy to verify analytically)

• This implicitly sets *competition* between weights – one weight can only increase if others decrease

• Such competition is called *heterosynaptic* plasticity, and could be implemented by homeostatic plasticity mechanisms. Heterosynaptic plasticity requires weight changes even when pre-synaptic neuron is inactive; homosynaptic plasticity requires coactivity of pre and post.

• In practice, subtractive normalisation is more strongly competitive than multiplicative normalisation (and unrealistically so)
Oja’s Rule

• Normalisation and synaptic competition can also be implicitly incorporated using other learning rules

• One example is Oja’s rule: 

$$ \Delta w_i = \epsilon(x_i y - w_i y^2) $$
Oja’s Rule

• Normalisation and synaptic competition can also be implicitly incorporated using other learning rules

• One example is Oja’s rule:

\[ \Delta w_i = \epsilon (x_i y - w_i y^2) \]

\[
\Delta w_i = \epsilon \sum_{\mu,j} w_j x_i^\mu x_j^\mu - \epsilon \sum_{\mu,j,k} w_i w_j w_k x_j^\mu x_k^\mu
\]
Oja’s Rule

- Normalisation and synaptic competition can also be implicitly incorporated using other learning rules.

- One example is Oja’s rule:

\[ \Delta w_i = \epsilon (x_i y - w_i y^2) \]

\[
\Delta w_i = \epsilon \sum_{\mu,j} w_j x_i^\mu x_j^\mu - \epsilon \sum_{\mu,j,k} w_i w_j w_k x_j^\mu x_k^\mu
\]

\[ 0 = Q \cdot w - (w \cdot Q \cdot w)w \quad \text{(at steady state)} \]

- The quadratic term normalises/stabilises the weights.
- The final equation tells us that, at steady state, the weights \( w \) are an eigenvector of \( Q \).
• Oja’s rule implements a kind of multiplicative normalisation

• Oja’s rule is not biologically motivated – what is the interpretation of the quadratic dependence on $y$?

• Theoretical motivation: Oja’s rule does PCA (finding the first PC) while maintaining stable weights

• This alone is ultimately not very powerful – if we have multiple such neurons, they will all learn the same PC...
Learning with Multiple Neurons

• Oja’s rule for one neuron is: \( \Delta w_i = \epsilon (x_i y - w_i y^2) \)

• Now assume we have \( M \) neurons. To avoid all neurons learning the same PC, we can add “interactions” between the neurons:

\[
\Delta w_{ij} = \epsilon (x_i y_j - y_j \sum_{k=1}^{M} w_{ik} y_k)
\]

• We can interpret these interactions as lateral inhibition (sort of...)
• This rule can be shown to learn the first \( M \) principal components of the input covariance matrix \( Q \)
Other Learning Rules: Generative Models

• Earlier in the course we looked at sparse coding, ICA, and predictive coding

• Each of these has a learning rule for the weight updates

• However, in those models the learning rules are derived from an underlying generative model of the input data

• There are two approaches to studying plasticity: 1) incorporate detail from biology and study the consequences 2) start from a generative model/objective function and derive a learning rule
Summary

- Hebbian learning picks out the dominant eigenvector of the input

- Hebbian learning is unstable without mechanisms to limit synaptic weights

- Competition between weights can help with stability and learning of interesting patterns

- Competition between neurons can lead to different neurons learning different input patterns

- Synaptic learning rules can be linked to unsupervised algorithms (e.g., PCA)
• Lecture notes Ch. 13

• Dayan and Abbott Ch. 8