Neural Networks on Energy Landscapes

Keith L. Downing

The Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
keithd@idi.ntnu.no

December 16, 2020
Energy Metrics for Neural Networks

Energy = Conflict between:
- presynaptic activity
- postsynaptic activity
- synaptic relationship

energy = \sum_{pre, post} \left( -X_{pre}X_{post}W_{pre\rightarrow post} \right)

High Conflict = High Energy

Low Conflict = Low Energy

Presynaptic Neuron

Excitatory Connection (Positive Weight)

Postsynaptic Neuron

Inhibitory Connection (Negative Weight)

Postsynaptic Neuron
Energy Gradients

- Minimize energy instead of error.
- Each weight’s contribution to energy is **only local**:

\[
\frac{\partial \text{Energy}}{\partial W_{ab}} = \frac{\partial}{\partial W_{ab}} \sum_{j,k} -X_j X_k W_{jk} = -X_a X_b
\]

where \( X_j, X_k, W_{j,k} \in [-1, 1] \)

- Learning = Adjusting weights to reduce energy.
- Learning = Descending the energy gradient.

\[
\triangle W_{ab} = -\lambda \frac{\partial \text{Energy}}{\partial W_{ab}} = \lambda X_a X_b
\]

where \( \lambda = \) learning rate

- This is very Hebbian, very local, very biological.
- \( X_a, X_b \) and Energy take many forms (in different models), but learning remains Hebbian.
**Hebb Rule: Fire Together, Wire Together**

When an axon of cell A is near enough to excite a 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.

\[ \Delta w_{i,j} = \lambda u_i v \]

Keith L. Downing  
Neural Networks on Energy Landscapes
Each data case has an energy based on the neural activity \( (X_i \forall i) \) that it induces.

Learning = adjusting network weights such that:
- Target data (i.e. positive cases) produce (local or global) energy minima.
- Other data (i.e. negative cases) produce high energy.

Why? Net clearly separates right from wrong.
Hopfield Networks (John Hopfield, 1982)

By invoking statistical mechanics (a hard science), Hopfield rescued neural nets from their Minsky-Papert-induced slump.

Simple model of content-addressable memory.

Energy Model from statistical mechanics (spin-glass theory):

\[ E = -a \sum_{k=1}^{C} \sum_{j=1}^{C} w_{jk} c_j c_k - b \sum_{k=1}^{C} I_k c_k \]  

(1)

\( w \) = weights; \( a, b \) = constants

\( c_k \) = activation level of neuron \( k \);

\( I_k \) = input to neuron \( k \).
Auto-Associative Learning

Training and Testing diagrams with nodes a, b, c, and d, showing relationships and transformations.
Hetero-Associative Learning

Keith L. Downing  Neural Networks on Energy Landscapes
# Learning in Hopfield Nets

## 1. Patterns to Learn

<table>
<thead>
<tr>
<th></th>
<th>p1</th>
<th>p2</th>
<th>p3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

- On (+1)
- Off (-1)
- Neutral (0)

## 2. Hebbian Weight Initialization

<table>
<thead>
<tr>
<th></th>
<th>w_{12}</th>
<th>w_{13}</th>
<th>w_{14}</th>
<th>w_{23}</th>
<th>w_{24}</th>
<th>w_{34}</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>+1</td>
<td>+1</td>
<td>-1</td>
<td>+1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>p2</td>
<td>1/3</td>
<td>-1/3</td>
<td>1/3</td>
<td>1/3</td>
<td>-1/3</td>
<td>1/3</td>
</tr>
</tbody>
</table>

Encode intra-pattern correlations in the weights (P = # input patterns)

\[
\sum_{p=1}^{P} c_{pk} c_{pj}
\]

This is batch Hebbian learning.

## 3. Build Network

- 1
- 2
- 3
- 4

On (+1)  Off (-1)  Neutral (0)
1. Enter a partial pattern
2. Repeat updating activation levels (asynchronously):

\[ c_k(t + 1) \leftarrow \text{sign} \left( \sum_{j=1}^{C} w_{kj} c_j(t) + I_k \right) \]  

3. Updates tend to reduce total energy.
Search on the energy landscape does not always end in global minima → spurious memories.

Hopfield nets have very limited practical use.

But they are extremely important, theoretically, and basis for more advanced energy-based models.
Boltzmann Machines

- Similar to Hopfield net, but with a hidden layer.
- Uses link between probability and energy via Boltzmann equation.
- Central philosophy: a network properly interprets (understands) its inputs only when it can **generate** them.
- Functions as an autoencoder with two phases: **recognition** (clamped, awake) and **generative** (sleep, dreaming).
- Learning = **Contrastive Divergence**, both Hebbian and Anti-Hebbian, based on activations achieved during both phases.
The Boltzmann Distribution

- $E(s_i)$ = Energy of the $i$th system state.
- $p(s_i)$ = probability that system is in the $i$th state.
- $T$ = temperature (Kelvin)
- $k$ = Boltzmann constant

**Statistical Mechanics**

$$p(s_i) = rac{e^{-\frac{E(s_i)}{kT}}}{Z}$$

where $Z = \sum_j e^{-\frac{E(s_j)}{kT}}$ := partition function

**Machine Learning (often ignore $k$ and $T$)**

$$p(s_i) = \frac{e^{-E(s_i)}}{Z}$$

where $Z = \sum_j e^{-E(s_j)}$ := partition function

Low energy states have higher probabilities
Differentiating The Boltzmann Distribution (1)

In many situations, we want to know how:

- changes in a state affect its energy: \( \frac{\partial E(s_i)}{\partial s_i} \)
- changes in a state affect its Boltzmann probability: \( \frac{\partial p(s_i)}{\partial s_i} \)

By the chain rule of Calculus:

\[
\frac{\partial p(s_i)}{\partial s_i} = \frac{\partial E(s_i)}{\partial s_i} \frac{\partial p(s_i)}{\partial E(s_i)}
\]

In a complex system (e.g. a neural network), the state consists of many components (e.g. activations and weights), and we will want to know the effect of any such component (e.g. \( w_{j,k} \)) upon \( E(s_i) \) and \( p(s_i) \). Again, we can use the chain rule:

\[
\frac{\partial p(s_i)}{\partial w_{j,k}} = \frac{\partial E(s_i)}{\partial w_{j,k}} \frac{\partial p(s_i)}{\partial E(s_i)}
\]
It is often easier to work with $\log_e(p(s_i)) = \ln(p(s_i))$ than with $p(s_i)$, especially when $p(s_i)$ is represented as a complicated fraction.

$$\ln(p(s_i)) = \ln\left(\frac{e^{-E(s_i)}}{Z}\right) = \ln(e^{-E(s_i)}) - \ln(Z) = -E(s_i) - \ln(Z)$$

To find the effect of $w_{j,k}$ upon $\ln(p(s_i))$:

$$\frac{\partial \ln(p(s_i))}{\partial w_{j,k}} = -\frac{\partial E(s_i)}{\partial w_{j,k}} - \frac{\partial \ln(Z)}{\partial w_{j,k}}$$

As shown earlier, $\frac{\partial E(s_i)}{\partial w_{j,k}} = -X_j X_k$, so:

$$\frac{\partial \ln(p(s_i))}{\partial w_{j,k}} = X_j X_k - \frac{\partial \ln(Z)}{\partial w_{j,k}}$$

Hebbian term (red) minus deriv log of partition function
Differentiating The Boltzmann Distribution (3)

Since $\frac{\partial \ln(f)}{\partial f} = \frac{\partial f}{f}$:

\[
\frac{\partial \ln(Z)}{\partial w_{j,k}} = \frac{\frac{\partial Z}{\partial w_{j,k}}}{Z} = \frac{1}{Z} \frac{\partial Z}{\partial w_{j,k}} = \frac{1}{Z} \frac{\partial (\sum_a e^{-E(s_a)})}{\partial w_{j,k}} = 1 \frac{1}{Z} \sum_a \frac{\partial e^{-E(s_a)}}{\partial w_{j,k}} =
\]

\[
= \sum_a \frac{e^{-E(s_a)}}{Z} \frac{-\partial E(s_a)}{\partial w_{j,k}}
\]

Each term in the summation is easy: 1) definition of $p(s_a)$ and 2) deriv of $E$ w.r.t. a weight:

\[
\frac{\partial \ln(Z)}{\partial w_{j,k}} = \sum_a p(s_a) X_j^{(a)} X_k^{(a)}
\]

Putting it all together:

\[
\frac{\partial \ln(p(s_i))}{\partial w_{j,k}} = X_j X_k - \sum_a p(s_a) X_j^{(a)} X_k^{(a)}
\]

Hebb term (red) + Anti-Hebb term avg’d over all possible states (blue)
Gradient Ascent Learning

To maximize the log likelihood that the model \((g)\) generates the target dataset \((D)\), we want to change weights to maximize the average of 
\[ \ln(p_g(s_d)) \] over all \(d \in D\), also written as 
\[ \langle \ln(p_g(d)) \rangle_{d \in D} \]

For any given weight \((w_{j,k})\), find its average effect \(\forall d \in D\) (where \(\|D\| = N\)):

\[
\frac{1}{N} \sum_{d \in D} \frac{\partial \ln(p_g(s_d))}{\partial w_{j,k}} = \frac{1}{N} \sum_{d \in D} \{ X_j^{(d)} X_k^{(d)} - \sum_a p_g(s_a) X_j^{(a)} X_k^{(a)} \}
\]

Since the blue term is the same value for each \(d \in D\):

\[
= \frac{1}{N} \sum_{d \in D} X_j^{(d)} X_k^{(d)} - \sum_a p_g(s_a) X_j^{(a)} X_k^{(a)}
\]

These are two different expected values, over the dataset and universal state set \((S)\), respectively:

\[
= \langle X_j^{(d)} X_k^{(d)} \rangle_{d \in D} - \langle X_j^{(a)} X_k^{(a)} \rangle_{a \in S} = \delta
\]
An alternative to maximizing log likelihood of the data ($D$) (averaged over its distribution) is minimizing the KL Divergence between D’s distribution $Q_D^0$ and the distribution of patterns produced by the model (g) (e.g. neural network) when processing the data D: $Q_D^g$.

$$\begin{align*}
D_{KL}(Q_D^0 \parallel Q_D^g) &= \sum_{d \in D} p(d) \ln \left( \frac{p(d)}{p_g(d)} \right) \\
&= \left\langle \ln(p(d)) \right\rangle_{d \in D} - \left\langle \ln(p_g(d)) \right\rangle_{d \in D} = -H(Q_D^0) - \left\langle \ln(p_g(d)) \right\rangle_{d \in D}
\end{align*}$$

To derive a learning rule for $w_{j,k}$, find the deriv. of $D_{KL}$ w.r.t. $w_{j,k}$:

$$\frac{\partial D_{KL}(Q_D^0 \parallel Q_D^g)}{\partial w_{j,k}} = -\frac{\partial H(Q_D^0)}{\partial w_{j,k}} - \frac{\partial \left\langle \ln(p_g(d)) \right\rangle_{d \in D}}{\partial w_{j,k}} = 0 - \frac{\partial \left\langle \ln(p_g(d)) \right\rangle_{d \in D}}{\partial w_{j,k}}$$

D’s entropy ($H(Q_D^0)$ is independent of M’s weights, hence the (red) zero. Using results from the previous slide (note the negative sign):

$$\sum_{d \in D} \frac{\partial \ln(p_g(s_d))}{\partial w_{j,k}} = \left\langle X_j^{(a)} X_k^{(a)} \right\rangle_{a \in S} - \left\langle X_j^{(d)} X_k^{(d)} \right\rangle_{d \in D} = -\delta$$
The Learning Rule

- To **maximize** log likelihood, \( \langle \ln(p_g(d)) \rangle_{d \in D} \) (where \( \lambda \) = learning rate), move **up** its gradient:
  \[
  \triangle w_{j,k} = (+1) \lambda \delta
  \]

- To **minimize** \( KL(Q_0, Q^g_D) \), move **down** its gradient:
  \[
  \triangle w_{j,k} = (-1) \lambda (-\delta) = \lambda \delta
  \]

So in either case:

\[
\triangle w_{j,k} = \lambda \delta = \lambda \left( \langle X_j^{(d)} X_k^{(d)} \rangle_{d \in D} - \langle X_j^{(a)} X_k^{(a)} \rangle_{a \in S} \right)
\]

- For any sizeable network, it is infeasible to generate all states \( s_a \in S \), so we estimate the blue term from a sample of states, \( S' \).

- Neural nets are often run incrementally, on minibatches of the data \( D' \subset D \), with weight updates after each minibatch is processed.

- The more realistic weight update is therefore:
  \[
  \triangle w_{j,k} = \lambda \left( \langle X_j^{(d)} X_k^{(d)} \rangle_{d \in D'} - \langle X_j^{(a)} X_k^{(a)} \rangle_{a \in S'} \right)
  \]
Boltzmann Machine Summary

- Models (e.g. neural networks) are trained to optimize an objective function, $F$, given a data distribution, $D$.
- Learning rules for optimization represent updates to model parameters (e.g. weights $(w_{j,k})$) that help push $F$ toward maxima or minima.
- These rules are often based on the gradient: $\frac{\partial F}{\partial w_{i,j}}$.
- When $F$ is Energy, then $\frac{\partial F}{\partial w_{i,j}} = -X_j X_k$. Thus, when the goal is to minimize energy, $\Delta w_{j,k} = \lambda X_j X_k = \text{classic Hebbian learning}$.
- When $F$ involves probabilities and maximizing the likelihood of the data, and when these probs are derived from energy via the Boltzmann equation, then gradient = Hebbian and anti-Hebbian terms, giving a learning rule of this form:

$$\Delta w_{j,k} = \lambda \left( \langle X_j^{(d)} X_k^{(d)} \rangle_{d \in D'} - \langle X_j^{(a)} X_k^{(a)} \rangle_{a \in S'} \right)$$

This is called a **Contrastive Hebb rule** with the Hebbian term based on the behavior of the net when constrained by $D$ (a.k.a. *clamped*) and embodying target(s), while the anti-Hebbian term comes from a more freely-running net and embodies prediction(s).
Contrastive Divergence Learning

\[ \Delta w_{j,k} = \lambda \left( \bigoplus \langle X_j^{(d)} X_k^{(d)} \rangle_{d \in D'} - \langle X_j^{(a)} X_k^{(a)} \rangle_{a \in S'} \bigoplus \right) \]

+ positive wake phase (based on data)
- negative sleep / dream phase (based on model-generated patterns)
- Weights between pre-trained layers are bi-directional: $w_{j,k} = w_{k,j}$
- Layer K+1 serves as hidden layer for recreating (autoencoding) Layer K \textbf{back on} Layer K.
- Thus, Layer K+1 forms a representation of Layer K.
Feed the input pattern (V) forward over weights W.

Hidden layer (H) uses sigmoid ($\sigma$) activation function.

Use H values as probabilities for Gibbs sampling.

$$p(h_k = 1|V) = \sigma(b_k + \sum_j v_j w_{j,k})$$

where $b_k$ = bias for kth hidden node.
Send hidden-layer pattern back over the same weights ($W$).
Input layer becomes a reconstruction layer and also uses $\sigma$ activation.
Use reconstructed values ($V$) as probabilities for Gibbs sampling.

$$p(v_k = 1 \mid H) = \sigma(c_k + \sum h_j w_{k,j})$$

where $c_k$ = bias for $k$th reconstruction node.

Keith L. Downing  
Neural Networks on Energy Landscapes
RBM Training Overview

1. Present an input case
2. Alternate forward and reconstruction phases until binary states $V$ and $H_0$ stabilize (i.e., change very little between rounds).
3. Modify weights via Contrastive Divergence.
4. Go to step 1

After many rounds through the training set, consider the RBM “trained” and move on to the next RBM, which will have $H_0$ as its input layer. The case set for $H_0$ is the mapping of the original input cases to $H_0$: The representation at $H_0$ is the data set for the $H_0$-$H_1$ RBM.

Keep training RBMs and generating new representations deeper and deeper into the network.

When all RBMs are trained, return to the original inputs and their corresponding targets and apply supervised learning throughout the net for fine tuning.
Dayan, Hinton, Radford and Zemel (1995)
Layered, Hierarchical NN version of Expectation Maximization (EM).
Uses Boltzmann Wake-Sleep model but with unidirectional links.
Recognition -vs- Generation weights, trained in alternating phases.
Uses gradients of Helmholtz free energy w.r.t. weights for $\Delta w$. 
Compared to earlier, we now separate data \( (d) \) from internal network states \( (s) \), which represent **causes** of the data.

**Basic notation and relationships**

- \( d \in D \) : environmental states (data) experienced by NN
- \( s \in S \) : internal neural network states
- \( p_g(s, d) \) : joint prob of \( s \) and \( d \) based on running NN \( g \).
- \( -\ln(p_g(s, d)) \) : surprisal of joint occurrence of \( s \) and \( d \).
- \( p_g(s|d) \) : prob that NN produces state \( s \) in response to \( d \)
- \( p_g(d) = \sum_s p_g(s, d) \) : marginal prob that NN produces \( d \).

**Expressions of conditional probability:**

\[
p_g(s|d) = \frac{p_g(s, d)}{p_g(d)} = \frac{p_g(s)}{\sum_s p_g(s, d)}
\]

**Defining **energy of explanation** (s explains d) as surprisal:**

\[
E_g(s; d) = -\ln(p_g(s, d))
\]

**semicolon \( \rightarrow \)** \( E_g \) is a function of \( s \), with \( d \) fixed.
Solving for $p_g$:

$$p_g(s, d) = e^{-E_g(s; d)}$$

Dividing both sides by the marginal prob $p_g(d) = \sum_s p_g(s, d)$:

$$\frac{p_g(s, d)}{p_g(d)} = \frac{e^{-E_g(s; d)}}{\sum_s p_g(s, d)}$$

A few simple substitutions:

$$p_g(s|d) = \frac{e^{-E_g(s; d)}}{\sum_s e^{-E_g(s; d)}}$$

In the Boltzmann distribution: $Z = \sum_s e^{-E_g(s; d)}$, so:

$$p_g(s|d) = \frac{e^{-E_g(s; d)}}{Z}$$

Note that $Z = p_g(d)$: the Boltzmann partition function is the marginal probability of producing data point $d$ over all NN states $s \in S$. 

Keith L. Downing
Neural Networks on Energy Landscapes
KL Divergence of $p$ and $p_g$

Goal: NN’s output distribution = distribution of $D$ $\iff$ minimize $D_{KL}$:

$$D_{KL}(p(D), p_g(D)) = \sum_d p(d) \ln \left( \frac{p(d)}{p_g(d)} \right) = \sum_d p(d) \ln(p(d)) - \sum_d p(d) \ln(p_g(d))$$

$$D_{KL}(p(D), p_g(D)) = -H(D) - \sum_d p(d) \ln(p_g(d)) = -H(D) + \langle -\ln(p_g(D)) \rangle$$

$H(D)$ indep of NN $\rightarrow$ Minimizing $D_{KL} = \text{Minimizing} \, \langle -\ln(p_g(D)) \rangle$

$$\langle -\ln(p_g(D)) \rangle = \sum_d p(d)(-\ln(p_g(d)))$$

$d_1, d_2 \in D$ are independent $\rightarrow$ minimize $\langle -\ln(p_g(D)) \rangle = \minimize -\ln(p_g(d)) \forall d \in D$

$-\ln(p_g(d))$ is called the NN’s free energy using weights $g$ and with output clamped to $d$. This is denoted $F_g(d)$. 
Free Energy

- Statistical mechanics defines Helmholtz free energy ($F$) as **average energy minus entropy**:
  \[ F = \langle E \rangle - TH \]
  where $\langle \rangle$ denotes average, $T = $ temperature, and $H = $ entropy.

- In Deep Learning, we equate negative log likelihood of the data with free energy, because as shown below:
  \[ -\ln(p_g(d)) = \langle E_g(s; d) \rangle_g - H_g(s|d) \]
  Average energy over all possible states produced when the model ($g$) experiences input state $d$.*

- Minus the entropy of those states (which are viewed as the **causal explanations** of $d$).

  * These NNs are stochastic, so an input will produce a probability distribution over the internal states.
Showing that $-\ln(p_g(d)) = F_g(d)$

Since $\sum_s p_g(s|d) = 1$ over all NN states $s \in S$

$$-\ln(p_g(d)) = - \sum_s p_g(s|d) \ln(p_g(d))$$

Since $p_g(d) = \frac{p_g(s,d)}{p(s|d)}$:

$$-\ln(p_g(d)) = - \sum_s p_g(s|d) \ln\left(\frac{p_g(s,d)}{p_g(s|d)}\right)$$

$$= - \sum_s p_g(s|d) \ln(p_g(s,d)) + \sum_s p_g(s|d) \ln(p_g(s|d))$$

$$= \sum_s p_g(s|d) (-\ln(p_g(s,d))) - H_g(s|d)$$

Since $E_g(s;d) = -\ln(p_g(s,d))$:

$$-\ln(p_g(d)) = \sum_s p_g(s|d) E_g(s;d) - H_g(s|d) = \langle E_g(s;d) \rangle_g - H_g(s|d) = F_g(d)$$
Goal: Free Energy $\downarrow \iff$ Energy $\downarrow$ - Entropy $\uparrow$

Energy (of network-environment coupling)

Neural Network

Correlation

Environment

Entropy (of network states)

Joint (Network + Environment) States

Low Entropy

Prob

Low Free Energy

Network States

High Entropy

Prob

Keith L. Downing
Neural Networks on Energy Landscapes
NN adapts to achieve useful mappings between internal states (S) and environmental states (D).

Mapping success = similarity between the prob. distr. of outputs produced by the NN ($p_g(D)$) and D’s natural prob. distr : $p(D)$.

These probabilities stem from a measure of energy based directly on the concept of **surprisal** from information theory.

This relationship between prob and energy is exactly the same as given by the Boltzmann distribution.

The process of making $p_g(D)$ similar to $p(D) = \text{minimizing Kullback-Leibler divergence: } D_{KL}(p_g(D), p(D))$.

This turns out to be equivalent to minimizing $-\ln(p_g(D)) = -\ln(Z)$, where again, Z is from Boltzmann distr., and $-\ln(Z)$ = free energy.

Thus, adapting an NN = minimizing free energy.
Weight modifications based on **local** network relationships $\rightarrow$ System moves down the global free-energy gradient.
Expectation Maximization (EM)

E (Expectation) Phase

Modify the **interpretation** of the data by altering the distribution of classes / causes / (internal states of the network) as a function of the input data.

- Clustering algorithm: Update the **current** class of each data point.
- Neural network: Update the recognition weights (Sleep Phase)
M (Maximization) Phase

Modify the **production** of data by altering the parameters that generate output data as a function of classes / causes / (internal network states).

- Clustering algorithm: Update the parameters (e.g. mean, variance) that define each class based on data points it **currently** contains.
- Neural network: Update the generator weights (Wake Phase)
In statistical mechanics, $Z$ (and thus $F$) is based on the true prob distr $(P(s))$ over all states, $s$.

But if we only have an estimate of $P(s)$, denoted $Q(s)$, then $\hat{F}(Q) := \text{variational free energy}$:

$$\hat{F}(Q) = -\ln Z + D_{KL}(Q\|P) = F + D_{KL}(Q\|P)$$

where $D_{KL}(Q\|P) \geq 0$ (Gibbs Inequality).

$\hat{F}(Q)$= upper bound on $F$

→ Minimizing $\hat{F}(Q)$ will minimize $F$.

$D_{KL} :=$ Kullback-Leibler Divergence, a.k.a. Relative Entropy:

$$D_{KL}(Q\|P) = \sum_s Q(s)\ln\left(\frac{Q(s)}{P(s)}\right) = \sum_s Q(s)\ln(Q(s)) - \sum_s Q(s)\ln(P(s))$$
Variational Free Energy = Avg(Energy) - Entropy

Begin with previous definition of variational free energy:

\[ \hat{F}(Q) = D_{KL}(Q \| P) - \ln Z = \sum_s Q(s) \ln \left( \frac{Q(s)}{P(s)} \right) - \ln Z \]

Since \( \sum_s Q(s) = 1 \) and \( Z \) is independent of \( Q(s) \):

\[ \hat{F}(Q) = \sum_s Q(s) \ln(Q(s)) - \sum_s Q(s) \ln(P(s)) - \sum_s Q(s) \ln Z \]

Rearranging a little, where \( -\sum_s Q(s) \ln(Q(s)) = H(Q(s)) = \) entropy.

\[ \hat{F}(Q) = - \sum_s Q(s)(\ln(P(s)) + \ln Z) + \sum_s Q(s) \ln(Q(s)) \]

\[ = \sum_s Q(s)(-\ln(P(s)Z)) - H(Q(s)) \]

Rearrange definition of \( P(s) \) to get: \( E(s) = -\ln(P(s)Z) \), so:

\[ \hat{F}(Q) = \sum_s Q(s)E(s) - H(Q(s)) = \langle E(s) \rangle_Q - H_Q \]

Keith L. Downing  
Neural Networks on Energy Landscapes
EM and Variational Free Energy

Goal: Minimize $F_g^r(d)$, the variational free energy.

Free Energy = $F_g(d) = -\ln(p_g(d))$

$$F_g^r(d) = D_{KL}(p_r(S|d), p_g(S|d)) + F_g(d) = D_{KL}(p_r(S|d), p_g(S|d)) - \ln(p_g(d))$$

where $p_g(S|d)$ is normally an approximation:

$$p_g(S|d) \approx \frac{\text{likelihood} \cdot \text{prior}}{\sum_s p_g(d|s)}$$

E Phase - Modify recognition (interpretation, representation) weights to bring the distribution of causal states (based on the data) closer to the generator’s distribution).... thereby reducing $D_{KL}$.

M Phase - Modify the generative weights to better produce the data.... to maximize $\ln(p_g(d))$ (i.e. minimize the free energy).
Helmholtz Wake-Sleep Algorithm

Wake Phase (Bottom Up)

Input (Data) Layer

Bias

Learn

Connect
Excite
Inhibit
Recognition Weights
Generative Weights
“Target”
“Prediction”
“Error”
Activation Function
Step #

Keith L. Downing
Neural Networks on Energy Landscapes
Helmholtz Dreaming

Sleep Phase (Top Down)

Input (Data) Layer

1. "Dreaming"
2. W^R
3. V^G
4. F(.)
5. W^G
6. E^X
7. P^X
8. T^X

Bias

Excite
Inhibit
Recognition Weights
Generative Weights
"Target"
"Prediction"
"Error"
Activation Function
Step #

Learn

Keith L. Downing
Neural Networks on Energy Landscapes
Learning in the Wake Phase

Delta Rule ($\lambda = \text{learning rate}; \ T^Y = \text{target}$)

$$\Delta W^G = \lambda \ T^X [T^Y - F(T^X \bullet W^G)]$$
Learning in the Sleep (Dreaming) Phase

Top-Down Signaling creates Targets for Training Bottom-Up (Recognition) Weights

Postsynaptic Layer

$F(.)$

$V^R$

Sleep Phase:
Update Recognition Weights

Presynaptic Layer

$T^D$

$E^Y$

$P^Y$

Learn

Delta Rule ($\lambda = \text{learning rate}; \ T^Y = \text{target}$)

$$\Delta V^R = \lambda T^D [T^Y - F(T^D \cdot V^R)]$$
Energy Networks (Summary)

- Hopfield Net
  - Unidirectional Links
  - Recog - vs- Gen Links
  - Free Energy

- Boltzmann Machine
  - Many Hidden Layers
  - Intralayer Links

- Helmholz Machine
  - Unidirectional Links
  - Recog - vs- Gen Links
  - Free Energy

- Restricted Boltzmann Machine
  - Many Hidden Layers
  - Intralayer Links

Keith L. Downing  Neural Networks on Energy Landscapes
Consider a different setting of NN parameters (i.e. model) $r$. Compare $p_r$ to $p_g$ on data point $d$:

$$D_{KL}(p_r(S|d), p_g(S|d)) = \sum_s p_r(s|d) \ln \left( \frac{p_r(s|d)}{p_g(s|d)} \right) = \sum_s p_r(s|d) \ln(p_r(s|d)) - \sum_s p_r(s|d) \ln(p_g(s|d)) =$$

Using definition of entropy ($H$) and $p_g(s|d) = \frac{p_g(s,d)}{p_g(d)}$:

$$= -H_r(S|d) - \sum_s p_r(s|d) \ln \left( \frac{p_g(s,d)}{p_g(d)} \right) =$$

$$= -H_r(S|d) - \sum_s p_r(s|d) \ln(p_g(s, d)) + \sum_s p_r(s|d) \ln(p_g(d)) =$$

Since $\sum_s p_r(s|d) = 1$ and $p_g(d)$ is indep of $p_r(s|d)$:

$$= -H_r(S|d) - \sum_s p_r(s|d) \ln(p_g(s, d)) + \ln(p_g(d))$$
Variational Free Energy (2)

Since $ln(p_g(d)) = -F_g(d)$ (free energy) and $ln(p_g(s; d)) = -E_g(s; d)$:

$$D_{KL}(p_r(S|d), p_g(S|d)) = -H_r(S|d) + \sum_s p_r(s|d)E_g(s; d) - F_g(d)$$

The blue term is the average energy in model g ($E_g$) but weighted by the distribution of states given by $p_r$.

From earlier: Free Energy = $F_g(d) = \langle E_g(s; d) \rangle_g - H_g(s|d)$.

The Variational Free Energy (from r to g) is defined as:

$$F^r_g(d) = \langle E_g(s; d) \rangle_r - H_r(S|d)$$

From above, it can also be expressed as:

$$F^r_g(d) = D_{KL}(p_r(S|d), p_g(S|d)) + F_g(d)$$
Using Variational Free Energy

Since $D_{KL}(p_r(S|d), p_g(S|d)) \geq 0 \rightarrow F'_g(d) \geq F_g(d)$.

So $F'_g(d)$ is an **upper bound (u.b.)** on $F_g(d)$.

By minimizing $F'_g(d)$, we minimize $F_g(d)$.

This is useful when $p_r(S|d)$ is a more accurate approximation of $p(S|d)$ than $p_g(S|d)$.

Or when $p_r(S|d)$ is accessible, but $p_g(S|d)$ is not.
In the Helmholtz Machine, minimizing $F_r^g(d)$ results from a version of the Delta Rule, a Hebbian variant.

$r$ = states arising from recognition flow; $g$ = those from generation flow.
In the Wake phase, recognition flow sets binary targets, $x_j$.

These binary values stem from Gibbs sampling of $s_j$, which represents a probability derived from a sigmoid activation function.

Thus, the probability of $x_j$ having its current value (a 0 or a 1), is:

$$p(x_j) = s_j^{x_j} (1 - s_j)^{(1-x_j)}$$

And the probability of a complete internal state, $x$, is:

$$p(x) = \prod_i s_i^{x_i} (1 - s_i)^{(1-x_i)}$$

In the Wake Phase, $s_j$ and $x_j$ are first determined by recognition flow, which involves the recognition weights. These states represent targets.

But only the generation wgts (e.g. $w_{kj}$) are updated, based upon how they affect the probs of the (target) states (e.g. $p(s)$).

Update rule for $w_{kj}$ results from calculating deriv $F_g^r(d)$ w.r.t. $w_{kj}$. 
Differentiating the Variational Free Energy

- In Wake Phase, the distribution of target states is determined by the recognition flow \( r \)
- So we need to minimize \( F^r_g(d) = \langle E_g(s; d) \rangle_r - H_r(S|d) \),
- instead of \( F_g(d) = \langle E_g(s; d) \rangle_g - H_g(s|d) \), based on generative (g) distr.

\[
\frac{\partial F^r_g(d)}{\partial w_{kj}} = \frac{\partial \langle E_g(s; d) \rangle_r}{\partial w_{kj}} - \frac{\partial H_r(S|d)}{\partial w_{kj}}
\]

Orange term = 0, since entropy of r distr \( (H_r) \) is indep of the gen wgts:

\[
\frac{\partial F^r_g(d)}{\partial w_{kj}} = \frac{\partial \langle E_g(s; d) \rangle_r}{\partial w_{kj}}
\]

Since \( E_g(s; d) = -\ln(p_g(s, d)) \):

\[
\frac{\partial F^r_g(d)}{\partial w_{kj}} = -\frac{\partial \langle \ln(p_g(s, d)) \rangle_r}{\partial w_{kj}}
\]
Deriving The Learning Rule (1)

- $w_{kj}$ updated incrementally, after processing one (or a few) data cases.
- Thus, we consider the energy associated with each individual case $(d)$, and the goal is to **minimize** that energy.

$$
\triangle w_{kj} = -\lambda \frac{\partial E_g(s; d)}{\partial w_{kj}} = -\lambda \frac{-\partial \ln(p_g(s, d))}{\partial w_{kj}} = \lambda \frac{\partial \ln(p_g(s, d))}{\partial w_{kj}}
$$

- $p_g(s, d)$ = the probability of the current state of the network. The state is actually based on the binary values $(x_i)$, which include the input values $(d)$ (also binary). So we focus on $p_g(x)$:

$$
p_g(x) = \prod_i x_i^{s_i} (1 - s_i)^{(1-x_i)}
$$

- Calculating it’s natural log:

$$
\ln(p_g(x)) = \sum_i x_i \ln(s_i) + (1 - x_i) \ln(1 - s_i)
$$

- Weight $w_{kj}$ affects only one neuron, the jth, so the deriv simplifies:

$$
\frac{\partial \ln(p_g(x))}{\partial w_{kj}} = \frac{\partial [x_j \ln(s_j) + (1 - x_j) \ln(1 - s_j)]}{\partial w_{kj}}
$$
Remember that $x_j$ stems from Gibbs sampling of $s_j$, whose value was derived from recognition weights, though $s_j$ (but not $x_j$) was modified on the downward pass.

Hence, $x_j$ is independent of the generative weight, $w_{kj}$. Key Fact!!

This simplifies the derivative even more:

$$\frac{\partial \ln(p_g(x))}{\partial w_{kj}} = x_j \frac{\partial \ln(s_j)}{\partial w_{kj}} + (1 - x_j) \frac{\partial \ln(1 - s_j)}{\partial w_{kj}}$$

Since $s_j$ is the output of a sigmoid, whose derivative is output $x$ (1-output):

$$\frac{\partial s_j}{\partial w_{kj}} = x_k s_j (1 - s_j)$$
Deriving The Learning Rule (3)

Using the substitution of $\Phi = x_k s_j (1 - s_j)$:

$$\frac{\partial \ln(p_g(x))}{\partial w_{kj}} = x_j \frac{\Phi}{s_j} + (1 - x_j) \frac{-\Phi}{(1 - s_j)} = \frac{x_j \Phi (1 - s_j) - s_j \Phi (1 - x_j)}{s_j (1 - s_j)}$$

Simplifying:

$$\frac{\partial \ln(p_g(x))}{\partial w_{kj}} = x_k [x_j (1 - s_j) - s_j + s_j x_j] = x_k [x_j - s_j]$$

Thus, the update is the Delta Rule:

$$\triangle w_{kj} = \lambda \cdot \text{pre-synaptic-output} \cdot [\text{target} - \text{post-synaptic-output}]$$

$\triangle w_{kj} = \lambda \cdot \text{pre-synaptic-output} \cdot \text{target} - \lambda \cdot \text{pre-synaptic-output} \cdot \text{output}$