CENG 783
Special topics in Deep Learning

Week 13–14
Boltzmann Machines
Sinan Kalkan
today

• Hopfield Networks
• Boltzmann Machines
• Restricted Boltzmann Machines
A Brief History Of Deterministic And Stochastic Networks

1982 Hopfield Nets
1985 Boltzmann Machines / Harmony Nets
1986 Back Propagation
1992 Sigmoid Belief Networks
2005 Restricted Boltzmann Machines and Deep Belief Nets
2009 Deep Learning With Back Propagation
Hopfield Networks

Neural networks and physical systems with emergent collective computational properties, Hopfield and Tank, PNAS, 1982.
Hopfield networks

- No input-output differentiation. Every neuron is both an input & output

- An **undirected weighted** graph \( G = (V, E) \)
  - The vertices \( V \) are the neurons
  - The edges \( E \) are connections between the neurons (all neurons are connected)
  - The edges are real valued: \( w_{ij} \in \mathbb{R} \) (\( w_{ij} = w_{ji} \) and \( w_{ii} = 0 \))
  - The activations are binary (+1/-1 or 1/0)

- Content addressable memory
• Training on set of patterns causes them to become attractors

• Degraded input is mapped to nearest attractor
• Physical systems can also act as a content-addressable memory (CAM).

• Consider the time evolution of the system in its state space. Some systems can be configured to have locally stable points. If the system is released from an initial state then it will roll down to the closest stable state.
Defining CAM

• CAM can be defined as a system whose stable points can be set as a set of pre-defined states.
• The stored patterns divide the state space into locally stable points, called “basins of attraction” in dynamical systems theory.
State of a neuron

- Let $s_i$ denote the state (value/activation) of neuron $i$.
  - Note that $s_i = -1$ or $+1$

- Then,
  
  $$s_i \left\{ \begin{array}{ll}
  +1, & \sum_j w_{ij}s_j \geq \theta_i \\
  -1, & \text{otherwise}
  \end{array} \right.$$

- $\theta_i$: threshold of neuron $i$. Mostly we set this zero.

- In short:
  
  $$s_i = \text{sign} \left( \left[ \sum_j w_{ij}s_j \right] - \theta_i \right)$$
Updating neurons

• Three possible schemes:
  • Synchronously: all units updated at each step.
  • Asynchronously I: at each time step select a random unit for update.
  • Asynchronously II: each unit independently chooses to update itself with some probability per unit time.

• Use asynchronously I and keep updating until no neuron changes its state.

Adapted from E. Sahin
Learning to **store a single pattern**

- Consider memorizing the pattern $\xi$. The condition is then

$$\text{sgn}(\sum_j w_{ij} \xi_j) = \xi_i$$

for all $i$.

- One solution is to have

$$w_{ij} = \frac{1}{N} \xi_i \xi_j$$

since $\xi_j^2 = 1$.

- If more than half of the bits are the same as $\xi$ then the network will “recall” $\xi$. That is $\xi$ is an attractor of the system.

- Question: what will happen if more than half of the bits are different from $\xi$?

Adapted from E. Sahin
Learning to store many patterns

Make $w_{ij}$ a superposition of correlation terms:

$$w_{ij} = \frac{1}{N} \sum_{\mu} \xi_i^\mu \xi_j^\mu$$

Observations

- weights can be positive or negative
- weight change positive if $\xi_i^\mu = \xi_j^\mu$
- weight change negative otherwise

“An associative memory model using the Hebb rule for all possible pairs $(i, j)$ with binary units and asynchronous updateing, is usually called a Hopfield model.”

Adapted from E. Sahin
Overall procedure

1. Determine the weights for the patterns to be stored (i.e., learning)

2. When you want to recall a pattern,
   a. set the neurons with the available parts of the pattern and
   b. let the network converge to a stable state (a pattern)
Example

Two patterns

\[ \xi^1 = (-1,-1,-1,+1) \]
\[ \xi^2 = (+1,+1,+1,+1) \]

Compute weights

\[ w_{ij} = \frac{1}{4} \sum_{\mu=1}^{2} \xi^\mu_i \xi^\mu_j \]

Recall

\[ s_i = \text{Sgn} \left( \sum_j w_{ij} s_j \right) \]

Input \((-1,-1,-1,+1)\) → \((-1,-1,-1,+1)\) stable
Input \((-1,+1,+1,+1)\) → \((+1,+1,+1,+1)\) stable
Input \((-1,+1,+1,+1)\) → \((-1,+1,+1,+1)\) spurious
Harmony or Energy

- We can define a scalar for the energy of the state of the network:

\[ E = - \sum_i \sum_{j<i} w_{ij} s_i s_j + \sum_i \theta_i s_i \]

or

\[ E = - \frac{1}{2} \sum_i \sum_j w_{ij} s_i s_j + \sum_i \theta_i s_i \]

- Harmony (H) = negative of the energy

- This is called energy since when you update neurons randomly, it either decreases or stays the same.

- Repeatedly updating the network will eventually make the network converge to a local minimum, i.e., a stable state.
Why does energy stay constant or decrease?

To show that the activation rule can only decrease energy, define $S'_i$ as the updated value of $S_i$:

$$S'_i = \text{sgn}(\sum_j w_{ij}S_j)$$

If there is a change in $S_i$ then the change in the energy function is

$$H' - H = \Delta H = -\sum_{j \neq i} w_{ij}S'_iS_j + \sum_{j \neq i} S_iS_j w_{ij}$$

$$= 2S_i \sum_{j \neq i} w_{ij}S_j$$

$$= 2S_i \sum_j w_{ij}S_j - 2w_{ii}$$

The first term negative due to the first equation in this slide. The second is negative since $w_{ii} = p/N$.

Therefore, energy decreases with every update. Adapted from E. Sahin
Skipping many details

- Stability conditions
- Storage capacity
- Increasing robustness
- Extension for continuous-valued patterns
- ...

Ludwig Boltzmann
(1844 –1906)

Boltzmann Machines
Motivation

• A Hopfield net always makes decisions that reduce the energy.
  – This makes it impossible to escape from local minima.

• We can use random noise to escape from poor minima.
  – Start with a lot of noise so its easy to cross energy barriers.
  – This may mean we occasionally increase the energy
  – Slowly reduce the noise so that the system ends up in a deep minimum. This is “simulated annealing”.

Slide adapted from G. Hinton, U of Toronto
Boltzmann (Gibbs) Distribution

- Probability distribution of particles in a system over possible states:
  \[ F(\text{state}) \propto e^{-\epsilon/kT} \]
  \( \epsilon \): the energy of the state, \( k \): Boltzmann’s constant, \( T \): temperature.

- The probability that a system will be in a certain state:
  \[ p_i = \frac{e^{-\epsilon_i/kT}}{\sum_{j=1}^{M} e^{-\epsilon_j/kT}} \]
  where \( \epsilon_i \) is the energy of state \( i \).

- Interpretation: states with lower energy are more probable.

- The denominator is called the partition function (denoted by \( Q \) or \( Z \) in the literature):
  \[ Z = \sum_{j=1}^{M} e^{-\epsilon_j/kT} \]
Boltzmann Machines

- By Hinton & Sejnowski (1985)
- Boltzmann machines can be seen as the stochastic counterpart of Hopfield nets
- In fact, they have the same energy definition:
  \[ E = -\sum_i \sum_{j<i} w_{ij} s_i s_j + \sum_i \theta_i s_i \]
- However, we have hidden neurons now
  - The availability of hidden variables \(\Rightarrow\) bigger class of distributions that can be modeled \(\Rightarrow\) in principle, we can model distributions of arbitrary complexity
- Moreover, it is stochastic
Boltzmann Machines (cont’d)

• They are powerful machines
• Can be used to learn internal representations
• Can be used to represent and learn difficult combinatoric problems (given sufficient time)
• However, if the connectivity is unconstrained, learning/inference is not tractable
Probability of a neuron’s state

• Turning on a neuron \( i \) (i.e., \( s_i \) is changed to 1 from 0) causes change \( \Delta E_i \) in energy:

\[
\Delta E_i = E_{i=0} - E_{i=1} = -k_B T \ln(Z p_{i=0}) - (-k_B T \ln(Z p_{i=1}))
\]

\[
\Delta E_i = -k_B T \ln \left( \frac{Z p_{i=0}}{Z p_{i=1}} \right) = -k_B T \ln \left( \frac{p_{i=0}}{p_{i=1}} \right) = -k_B T \ln \left( \frac{1-p_{i=1}}{p_{i=1}} \right)
\]

• \( k_B \) is a constant. Assume that artificial concept of temperature \( (T) \) “absorbs” that:

\[
- \frac{\Delta E_i}{T} = \ln \left( \frac{1-p_{i=1}}{p_{i=1}} \right)
\]

\[
\exp \left( - \frac{\Delta E_i}{T} \right) = \frac{1}{p_{i=1}} - 1
\]

• This yields the famous logistic / sigmoid function:

\[
p_{i=1} = \frac{1}{1 + \exp \left( - \frac{\Delta E_i}{T} \right)}
\]

Using:

\[
p_i = \frac{e^{-\varepsilon_i/kT}}{Z}
\]
Interpretation of a state’s probability

\[ p_{i=1} = \frac{1}{1 + \exp\left(-\frac{\Delta E_i}{T}\right)} \]

a. If \( T = 0 \), the probability for turning on the neuron is \( \approx 1 \) if \( \Delta E_i \) is positive (energy reduced). If \( \Delta E_i \) is negative, \( p_{i=1} \approx 0 \).
   • Does it become equal to a Hopfield network then?

b. If \( T \) is high, then \( p_{i=1} \approx 1/2 \).
   • Half the chance is given to updating the neuron.

c. For a fixed \( T \), if \( \Delta E_i \) is zero, same as case – (b).

d. For a fixed \( T \), if \( \Delta E_i \) is very high, same as case – (a).
   • When the temperature is big, the network covers the whole state space.
   • In the cooling phase, when the temperature is small, the network converges to a minima, hopefully the global one.
How temperature affects transition probabilities

$p(A \rightarrow B) = 0.2$

$p(A \leftarrow B) = 0.1$

High temperature transition probabilities

Low temperature transition probabilities

Slide adapted from G. Hinton, U of Toronto
Energy change in Hopfield vs. Boltzmann Machines

- Energy decreases in Hopfield networks

- Although we have the same definition, why can energy increase in Boltzmann machines?
  - Because state updates that normally would not be possible in a Hopfield network is possible in Boltzmann machines.
An example of how weights define a distribution

\[ p(v^\alpha, h^\beta) = \frac{e^{-E_{\alpha \beta}}}{\sum_{\gamma \delta} e^{-E_{\gamma \delta}}} \]

<table>
<thead>
<tr>
<th>v</th>
<th>h</th>
<th>-E</th>
<th>e^{-E}</th>
<th>p(v, h)</th>
<th>p(v)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>.186</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>.186</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>.069</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>.025</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>.069</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>.025</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>.025</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>.025</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>.025</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>.025</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>.025</td>
<td></td>
</tr>
</tbody>
</table>

total = 39.70
Equilibrium

• We select a neuron and update its state according to the following probability:

\[ p_{i=on} = \frac{1}{1 + \exp\left(-\frac{\Delta E_i}{T}\right)} \]

• If this is repeated long enough for a certain temperature, the state of the network will depend on the state’s energy, and not on the initial state.

• In this condition, the log probabilities of global states become linear in their energies.

• This is called thermal equilibrium.

• Start from a high temperature, gradually decrease it until thermal equilibrium, we may converge to a distribution where energy level is close to the global minimum. ➔ Simulated Annealing.
Thermal Equilibrium

Start the system running at some temperature $T$.

Immediate behavior of system will be dependent on initial state.

At high temperatures, system quickly moves away from initial state.

At low temperatures, system takes much longer.
Thermal Equilibrium

• How do we understand we have reached it?
  • The average activation of neurons don’t change over time.
  • I.e., the probability of being in a state does not change.

• How can we know that we can reach thermal equilibrium?
  • “any network that always went downhill in some Lyapunov function in its deterministic version is guaranteed to reach a thermal equilibrium in its stochastic version” (S. Roweis)
  • In the deterministic case, our Lyapunov function is:
    \[ E = - \sum_{i,j<i} w_{ij} s_i s_j \]

• The initial state is not important!

• At low temperature:
  • There is a strong bias for states with low energy
  • But this is too slow to reach

• At high temperature:
  • Not a strong bias for low energy
  • Equilibrium is reached faster
Simulated annealing

- With a probability dependent on the temperature, choose to either stay in the current state or move to a neighboring state.
Thermal equilibrium is a difficult concept!

- It does not mean that the system has settled down into the lowest energy configuration.
- The thing that settles down is the **probability distribution** over configurations.

The best way to think about it is to imagine a huge ensemble of systems that all have exactly the same energy function.

- The probability distribution is just the fraction of the systems that are in each possible configuration.
- We could start with all the systems in the same configuration, or with an equal number of systems in each possible configuration.
- After running the systems stochastically in the right way, we eventually reach a situation where the number of systems in each configuration remains constant even though any given system keeps moving between configurations.
An example

In this network, there are four possible states with the following energies:

\[ E_{00} = 0.0 \quad E_{10} = -0.5 \]
\[ E_{01} = -0.5 \quad E_{11} = 0.0 \]

You can calculate transition probabilities between states:

\[ p_{00 \rightarrow 01} = \frac{1}{2} \left( \frac{1}{1 + \exp(-0.5)} \right) = 0.31 \]

An example

This can be summarized in the following transition matrix:

\[
P = \begin{pmatrix}
\frac{1}{2} \frac{1}{1+e^{1/2}} & \frac{1}{2} \frac{1}{1+e^{-1/2}} & \frac{1}{2} \frac{1}{1+e^{1/2}} & 0 \\
\frac{1}{2} \frac{1}{1+e^{1/2}} & \frac{1}{1+e^{-1/2}} & 0 & \frac{1}{2} \frac{1}{1+e^{1/2}} \\
\frac{1}{2} \frac{1}{1+e^{1/2}} & 0 & \frac{1}{1+e^{-1/2}} & \frac{1}{2} \frac{1}{1+e^{1/2}} \\
0 & \frac{1}{2} \frac{1}{1+e^{1/2}} & \frac{1}{2} \frac{1}{1+e^{-1/2}} & \frac{1}{1+e^{1/2}}
\end{pmatrix}
= \begin{pmatrix}
0.38 & 0.31 & 0.31 & 0 \\
0.19 & 0.62 & 0 & 0.19 \\
0.19 & 0 & 0.62 & 0.19 \\
0 & 0.31 & 0.31 & 0.38
\end{pmatrix}
\]
An example

Let us start from an arbitrary state. This has the following distribution:

\[ v_0 = (0.25, 0.25, 0.25, 0.25) \]

If we update this distribution using the transition matrix:

\[ v_1 = v_0 P \]

\[ = (0.25, 0.25, 0.25, 0.25) \begin{pmatrix} 0.38 & 0.31 & 0.31 & 0 \\ 0.19 & 0.62 & 0 & 0.19 \\ 0.19 & 0 & 0.62 & 0.19 \\ 0 & 0.31 & 0.31 & 0.38 \end{pmatrix} \]

\[ = (0.19, 0.31, 0.31, 0.19) \]

An example

We can continue doing that:

\[ \mathbf{v}_t = \mathbf{v}_{t-1} \mathbf{P}. \]

In thermal equilibrium:

\[ \mathbf{v} = \mathbf{v} \mathbf{P} \]

We could also calculate it directly in this case since the network is so small:

\[
\mathbf{v} = \frac{1}{e^{-E_{00}} + e^{-E_{01}} + e^{-E_{10}} + e^{-E_{11}}} (e^{-E_{00}}, e^{-E_{01}}, e^{-E_{10}}, e^{-E_{11}})
\]

\[
= \frac{1}{2} \left( \frac{1}{1 + e^{0.5}} \right) (1, e^{0.5}, e^{0.5}, 1)
\]

\[
= (0.19, 0.31, 0.31, 0.19).
\]

An example

We can continue doing that:

\[ v_t = v_{t-1} P. \]

However, the transition matrix is also updated in Boltzmann machines:

\[ v_t = v_{t-1} P = v_{t-2} P^2 = \cdots = v_0 P^t \]

Once the stable state is reached, the transition probabilities also stabilize:

\[ \lim_{t \to \infty} P^t = \begin{pmatrix}
0.19 & 0.31 & 0.31 & 0.19 \\
0.19 & 0.31 & 0.31 & 0.19 \\
0.19 & 0.31 & 0.31 & 0.19 \\
0.19 & 0.31 & 0.31 & 0.19
\end{pmatrix}. \]
Training

• The neurons are divided into two:
  • Visible units: $V$
  • Hidden units: $H$

• Distribution over the training set: $P^+(V)$

• The distribution over global states converges as the Boltzmann machine reaches thermal equilibrium.
  • $P^-(V)$ to denote this distribution

• Our goal: Approximate the real distribution $P^+(V)$ from $P^-(V)$

• The similarity bw the distributions:

$$G = D_{KL}(P^+(V), P^-(V)) = \sum_v P^+(v) \ln \left( \frac{P^+(v)}{P^-(v)} \right)$$

summation over all possible states of $V$.

• $G$ is a function of weights.
  • We can use gradient descent on $G$ to update the weights to minimize it.
Training (cont’d)

- Two phases:
  - Positive phase: visible units are initialized to a random sample from the training set.
  - Negative phase: the network runs freely. The units are not initialized to external data.

- Then:
  \[
  \frac{\partial G}{\partial w_{ij}} = \frac{1}{R} \left[ p_{ij}^+ - p_{ij}^- \right]
  \]
  - R: learning rate
  - \( p_{ij}^+ \): probability that both units are on at thermal equilibrium on the positive phase. \( E[s_i s_j] \)
  - \( p_{ij}^- \): probability that both units are on at thermal equilibrium on the negative phase.

- \( w_{ij} = w_{ij} - \frac{\partial G}{\partial w_{ij}} \)

- Needs only local information (compare it to backprop)

See the following for the derivation of the learning rule:

Training (cont’d)

• Another interpretation (Rojas, 1996):

Note that Boltzmann learning resembles Hebbian learning. The values of $\langle x_i x_j \rangle_{fixed}$ and $\langle x_i x_j \rangle_{free}$ correspond to the entries of the stochastic correlation matrix of network states. The second term is subtracted and is interpreted as Hebbian “forgetting”. This controlled loss of memory should prevent the network from learning false, spontaneously generated states. Obviously, Boltz-
Why do we need the negative phase?

The positive phase finds hidden configurations that work well with $v$ and lowers their energies.

$$p(v) = \sum_{u} \frac{\sum_{g} e^{-E(u,g)} e^{-E(v,h)}}{\sum_{h} \sum_{g} e^{-E(u,g)}}$$

The negative phase finds the joint configurations that are the best competitors and raises their energies.

---

Slide adapted from G. Hinton, U of Toronto
The batch learning algorithm

• Positive phase
  – Clamp a datavector on the visible units.
  – Let the hidden units reach thermal equilibrium at a temperature of 1 (may use annealing to speed this up)
  – Sample $s_i s_j$ for all pairs of units
  – Repeat for all datavectors in the training set.

• Negative phase
  – Do not clamp any of the units
  – Let the whole network reach thermal equilibrium at a temperature of 1
  – Sample $s_i s_j$ for all pairs of units
  – Repeat many times to get good estimates

• Weight updates
  – Update each weight by an amount proportional to the difference in $< s_i s_j >$ in the two phases.
Why Boltzmann Machine Failed

Too slow

- loop over training epochs
  - loop over training examples
    - loop over 2 phases (+ and -)
    - loop over annealing schedule for $T$
      - loop until thermal equilibrium reached
    - loop to sample $<o_i o_j>$

Sensitivity to annealing schedule

Difficulty determining when equilibrium is reached

As learning progresses, weights get larger, energy barriers get hard to break -> becomes even slower

Back prop was invented shortly after

- The need to perform pattern completion wasn’t necessary for most problems (feedforward nets sufficed)
References

• R. Rojas, “Neural Networks”, Ch14, 1996.


• S. Roweis, “Boltzmann Machines”, lecture notes:

  https://www.cs.nyu.edu/~roweis/notes/boltz.pdf

• For links and inspirations to physical systems (like Ising models and renormalization group theory), see:
  • https://charlesmartin14.wordpress.com/2015/04/01/why-deep-learning-works-ii-the-renormalization-group
Restricted Boltzmann Machines

http://blog.echen.me/2011/07/18/introduction-to-restricted-boltzmann-machines/
Restricted Boltzmann Machines (RBM)

- Invented by Smolensky (1986), improved by Hinton et al. (2006)

- RBM: Boltzmann Machine with restricted connectivity
  - Connections between hidden-visible units only!

- Smolensky called it Harmonium
  - Also called Harmony networks

- A simple learning method is used
  - Contrastive Divergence
Restricted Boltzmann Machines

• We restrict the connectivity to make inference and learning easier.
  – Only one layer of hidden units.
  – No connections between hidden units.
• In an RBM it only takes one step to reach thermal equilibrium when the visible units are clamped.
  – So we can quickly get the exact value of:
    \[ \langle s_i s_j \rangle_v \]

\[
p(s_j = 1) = \frac{1}{1 + e^{- (b_j + \sum_{i \in \text{vis}} s_i w_{ij})}}
\]
RBM Generative Model As A Product Of Experts

Restricted architecture of Boltzmann machine: no connections among the visible units, or among the hidden units.

"Probability of generating visible vector is prop. to product of probabilities that the visible vector would be generated by each of the hidden units acting alone."

\[
p(v=1 | \exists H_i = 1 \exists) = p(V | H_i)
\]

\[
= \frac{1}{Z} \prod_i p(v | H_i)
\]

\[
= \frac{\prod_i p(v | H_i)}{\prod_i p(v | H_i) + \prod_i p(v | H_i)}
\]

\[
= \frac{1}{1 + \prod_i p(v | H_i) / \prod_i p(v | H_i)}
\]

\[
= \frac{1}{1 + e^{-\sum_j w_j h_j}}
\]

\[
w_j = \log \left( \frac{p(v | H_j)}{p(v | H_j)} \right)
\]

Slide: Michael Mozer
Overview of training

• Again, we should update the weights according to the “positive” phase and the “negative” phase:
  \[ \Delta w_{ij} = R(<v_i h_j>_{data} - <v_i h_j>_{model}) \]

• For the “data” part:
  • Clamp the visible units with a pattern and determine the visible units with probability:
    \[ p(h_j = 1|v) = \sigma(b_j + \sum_i v_i w_{ij}) = \frac{1}{1 + \exp(b_j + \sum_i v_i w_{ij})} \]

• The “model” part is more problematic and slow.

• Shortcut (Hinton 2002):
  \[ \Delta w_{ij} = R(<v_i h_j>_{data} - <v_i h_j>_{recons}) \]
Start with a training vector on the visible units.

Then alternate between updating all the hidden units in parallel and updating all the visible units in parallel.

\[
\Delta w_{ij} = \epsilon \left( <s_is_j>^0 - <s_is_j>^\infty \right)
\]
A surprising short-cut

Start with a training vector on the visible units.

Update all the hidden units in parallel.

Update the all the visible units in parallel to get a “reconstruction”.

Update the hidden units again.

This is not following the gradient of the log likelihood. But it works very well.

\[ \Delta w_{ij} = \varepsilon \left( <s_is_j>^0 - <s_is_j>^1 \right) \]
Overview of training

- Take a training sample \( v \), sample a hidden vector using:
  \[
p(h_j = 1|v) = \sigma \left( b_j + \sum_i v_i w_{ij} \right)
  \]

- Compute the outer product of \( v \) and \( h \) and call this the “positive gradient”.

- Sample a reconstruction \( v' \) from \( h \) using:
  \[
p(v'_i = 1|h) = \sigma \left( b_i + \sum_j h_j w_{ij} \right)
  \]
  then resample the hidden activations \( h' \) from this.

- Compute the outer product of \( v' \) and \( h' \) and call this the “negative gradient”.

- Update the weights:
  \[
  \Delta w_{ij} = R(<v_i h_j>_{data} - <v_i h_j>_{recons})
  \]
Skipping many details

- Collecting the statistics in the binary or real-valued cases
- Initial values for the weights
- Overfitting
- Momentum
- ...

RBM vs. Autoencoders

• Both try to encode an input and to minimize its reconstructions from its encoding

• The objective functions minimized are different
  • RBMs directly model the joint distribution of the hidden & visible units
  • dAEs rely on the score of the learned decoding.

• There are many powerful extensions of both approaches

• Also many links, e.g., Vincent, “A Connection Between Score Matching and Denoising Autoencoders”, 2011.
Deep Belief Networks
DBN

- A stacked RBM
- First used by Hinton & Salakhutdinov (2006)
- Models the distribution:

\[ P(x, h^1, \ldots, h^\ell) = \left( \prod_{k=0}^{\ell-2} P(h^k|h^{k+1}) \right) P(h^{\ell-1}, h^\ell) \]

- Training is similar to autoencoders:

1. Train the first layer as an RBM that models the raw input \( x = h^{(0)} \) as its visible layer.
2. Use that first layer to obtain a representation of the input that will be used as data for the second layer. Two common solutions exist. This representation can be chosen as being the mean activations \( p(h^{(1)} = 1|h^{(0)}) \) or samples of \( p(h^{(1)}|h^{(0)}) \).
3. Train the second layer as an RBM, taking the transformed data (samples or mean activations) as training examples (for the visible layer of that RBM).
4. Iterate (2 and 3) for the desired number of layers, each time propagating upward either samples or mean values.
5. Fine-tune all the parameters of this deep architecture with respect to a proxy for the DBN log-likelihood, or with respect to a supervised training criterion (after adding extra learning machinery to convert the learned representation into supervised predictions, e.g. a linear classifier).
Deep RBM Autoencoder

Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the “data” for training the next RBM in the stack. After the pretraining, the RBMs are “unrolled” to create a deep autoencoder, which is then fine-tuned using backpropagation of error derivatives.

Hinton & Salakhutdinov (2006)