Boltzmann Machines

Learning by Correlation in a Hopfield-like Net

Boltzmann Machine

- Extends Hopfield model with learning.
- Based on probabilistic operation.
- Learning is by correlation (sort of Hebbian in character).

Boltzmann Machine Structure

- The Boltzmann Machine is like a Hopfield network, in which
- the neurons are divided into two subsets:
  - visible, which are further divided into:
    - input
    - output
  - hidden
- As with the Hopfield model, the weights are symmetric.

Boltzmann Machine Operation

- There are two modes of operation:
  - clamped mode
  - free mode
- In clamped mode, the input and output of visible neurons are held fixed, while the hidden neurons are allowed to vary.
- In free mode, only the inputs are held fixed and all other neurons are allowed to vary.

Probabilistic Firing (used in Training)

- All neurons have output in \( (+1, -1) \).
- The activation function determines not the exact next input, but rather the probability of the neuron’s output being set to 1:
  - \( f(\text{net}) = \text{probability that output is set to 1} \)
  - where \( \text{net} \) is the weighted sum input
  - where \( f(x) = 1/(1 + \exp(-2x)) \)
  - where \( [\cdot] \) is a parameter to be determined
  - so the higher the value of \( \text{net} \), the more likely the neuron will be set to 1.
Probabilistic Firing

- \( f(x) = \frac{1}{1 + \exp(-2bx)} \)
- Obviously this is a sigmoid:
  - With \( b = 0 \), the probability of setting output 1 is 0.5, i.e. total randomness.
  - As \( b \) increases, the probability of setting output to 1 approaches 1 if \( x > 0 \), and 0 if \( x < 0 \).

Controlling \( b \)

- In order to achieve a stable probability distribution for the network state, \( b \) is gradually increased from 0 over time.

- We can think of the increase in \( b \) as “cooling” the network.
- Thus, we can govern beta as \( b = \frac{1}{T} \) where \( T \) is the “temperature”, which decreases with time (according to a schedule).
- The overall process is known as “simulated annealing”.

Annealing Schedule

- The annealing schedule determines the temperature \( T \) as a function of the step of the algorithm.
- Example: \( T = \frac{T_0}{1 + \log k} \) where \( k \) is the step number and \( T_0 \) is an initial temperature.

History of Simulated Annealing

- SA was first proposed in 1983 as a method for optimizing wire-routing on VLSI chips (an NP-hard problem)
  by Kirkpatrick, Gelatt, and Vecchi.
- This was a widely-celebrated result.
- SA is now used as a way to avoid local minima in a number of computational problems.

Role of Annealing in Stabilization

- Generally, move in direction of decreasing energy.
- Occasionally, accept a move that increases energy.
- This will be done with high probability at first, but lower probability as annealing progresses.
Role of Annealing in Stabilization

Energy Landscape

- Normal move (toward local minimum)
- Decreasing energy moves are always accepted.
- Occasional contrary move

The probability of making a contrary move is inversely proportional to the energy increase and to the temperature (higher probability earlier in the annealing schedule).

How temperature affects transition probabilities (slide from Hinton)

- High temperature transition probabilities
  - $p(A \rightarrow B) = 0.2$
  - $p(A \rightarrow B) = 0.1$

- Low temperature transition probabilities
  - $p(A \rightarrow B) = 0.001$
  - $p(A \rightarrow B) = 0.000001$

Annealing Advantage (as summarized by Hinton)

- At high temperature the transition probabilities for uphill jumps are much greater.
- At low temperature the equilibrium probabilities of low energy states are much better than the equilibrium probabilities of high energy ones.

Stand-alone demo of Simulated Annealing

http://www.taygeta.com/annealing/demo1.html

Simulated Annealing Result

Energy-Based Simulation

- As we know from Hopfield theory, making a single transition according to the activation function will decrease the energy.
- So we can simply decide to “flip” a neuron based on whether the flip lowers the energy (defined as $-\sum w_{ij} y_i y_j$).
Energy-Based Simulation

- To include the annealing temperature:
  - If a flip lowers the energy, do it.
  - If a flip raises the energy by $|E|$, flip the output with probability $1/(1 + \exp(|E|/T))$.
  - This can be done by generating a random number $r$ between 0 and 1, then setting the output of the neuron depending on whether $r < \exp(|E|/T)$ the probability being greater or less than $1/2$ depending on which case.

Energy-Based Simulation

- A similar technique was originally used in the famous Metropolis, Rosenbluth, Teller equation of state calculations in statistical mechanics (Ising or “spin-glass” model).

Energy-Based Simulation

- In order to compute $|E|$, it is not necessary to fully compute the energy before and after. Instead could just use $|E| = \sum w_i y_i$ where $i$ is the neuron being flipped.

Code from a Boltzmann Machine

```c
/* Change the value for only 1 node, on temperature $t$. At this
temperature accept the change if it increases the energy, and accept it
with some probability, if it decreases it. Probability depends on $t$ */
void anneal_1_step(struct machine *p, double t)
{
  int node, layer;
  double dE;
  select_node(p, &layer, &node);
  dE = energy_change(p, layer, node);
  if( accept_change(dE, t) )
    flip_state(p, layer, node);
}
```

Code from a Boltzmann Machine

```c
/* Is a change of $dE$, acceptable at temperature $t$ */
accept_change(double dE, double t)
{
  double prob, rand;
  /* Always accept changes that decrease the energy */
  if( dE < 0 )
    return( 1 );
  /* If the change increases the energy, accept it with a
  certain probability */
  prob = 1 / (1 + exp(dE/t));
  rand = get_rand(0.0, 1.0);
  return( rand < prob );
```

Code from a Boltzmann Machine

```c
/* Simulated annealing over machine p. The temperature is constantly
decreasing over a set of values. For each temperature a set of state
changes are performed on randomly selected (non clamped!) nodes.
Each state change is selected with a certain probability. If it decreases
the total energy, it is selected; if not it is selected with a probability
that is a function of the temperature */
void anneal(struct machine *p)
{
  double temp = p->t0;
  int i, n;
  int node, layer;
  double dE;
```
Code from a Boltzmann Machine

```c
/* Vary the temperature */
while( temp >= p->tmin )
{
    /* For each temperature, perform a number of operations which 
       is a function of the temperature of annealing. */
    n = get_num_changes(temp, p);
    for( i=0; i<n; i++ )
    {
        select_node(p, &layer, &node);
        dE = energy_change(p, layer, node);
        if( accept_change(dE, temp) )
            flip_state(p, layer, node);
    }
    temp *= p->beta;
}
```

Boltzmann Distribution

- The name of the machine derives from the fact that, at steady state, if s and t are two states with energies $E_s$ and $E_t$ respectively, then the probabilities of being in those states $P[s]$ vs. $P[t]$ satisfy
  \[
  \frac{P[s]}{P[t]} = \exp(\frac{E_t - E_s}{T})
  \]
  where $T$ is the temperature. This is known as the “Boltzmann distribution” or “Boltzmann-Gibbs distribution”.

Learning in the Boltzmann Machine

- Suppose we set the input and output neurons according to a specific sample.
- We then anneal the network. The final state reached is not necessarily unique, due to the probabilistic moves made along the way.
- We can observe, over several such simulations, which neurons’ outputs are correlated at the ends, represented as a correlation $r_{ij} = \mathbb{E}[y_i y_j]$, the expected (average) value of the product of the outputs of neurons i and j.

Learning Rule

- Let $r^+$ be the correlation value when the network is run in \textit{clamped} mode, and $r^-$ be the correlation value when the network is run in \textit{free} mode.
- The Boltzmann learning rule is
  \[
  \Delta w_{ij} = h (r^+ - r^-)
  \]
  where $h$ is the learning rate.
- In other words, whether weights are changed depends on the difference between the correlations in clamped vs. free mode.

Batch Learning Algorithm (from Hinton)

- **Clamped phase**
  - For each data vector in the training set:
    - Clamp the data vector on the visible units.
    - Let the hidden units reach thermal equilibrium at a temperature of 1 (may use annealing to speed this up).
    - Sample $y_i y_j$ for all pairs of units.
- **Free phase**
  - Repeat many times to get good estimates
    - For each data vector in the training set:
      - Do not clamp any of the [output?] units.
      - Let the whole network reach thermal equilibrium at a temperature of 1.
      - Sample $y_i y_j$ for all pairs of units.
- **Weight updates**
  - Update each weight by an amount proportional to the difference in $\mathbb{E}[y_i y_j]$ in the two phases.
Ackley, Hinton, and Sejnowski, 1985 presented the following example:

- 4 line one-hot encoder-decoder, 2 hidden units

To prevent weights from growing too large, used a “noisy clamping” technique: each on bit of a clamped vector is set to off with prob. 0.15 and each off bit set to on with prob. 0.05.

- Network was unclamped and allowed to reach equilibrium. Statistics were gathered for the same number of annealings as in the clamped case.

- Annealing schedule: (time units @ temperature) 2@20, 2@15, 2@12, 4@10.

- 1 time unit = interval giving each neuron a chance to flip.

Additional Examples

- 4-2-4 encoder/decoder converged quickly
- 8-3-8: more difficult
- 40-10-40: converged in 98.6% of runs.

Boltzmann Simulators

- \texttt{/cs/cs152/boltz}
  - Simulates only the distribution, not learning.
  - It is analogous to a spin-glass simulation.
- \texttt{/cs/cs152/boltz}
  - Learning, weight-saving, etc.
  - An example, bxor, provides a demo of training a Boltzmann machine to implement xor. Run the shell script bxor.run (may have to run more than once for convergence).

Speedup Possibility

- Training of a Boltzmann machine is extremely slow.
- A possible speedup is to use the “mean-field” approximation to get the correlation values.
- This approach is due to Peterson and Anderson, 1987.

Mean-Field Theory

- If \( f \) is a function of two variables, then the expectation \( E[f(x, y)] \) can be approximated by \( f(E[x], E[y]) \).
- This idea can be applied to the weight change rule of the Boltzmann machine, which entails computing \( \langle \Delta y \rangle = E[y|x] \approx E[y|x] E[y] \)
Mean-Field Theory

- For the Boltzmann distribution, the probability that node $i$ takes value 1 at temperature $T$ can be shown to be:
  \[ p_1 = \frac{1}{1+\exp(-\sum w_j E[y_j]/T)} \]
- So the expected output of node $i$ is
  \[ E[y_i] = 1 \cdot p_1 + (-1) \cdot (1- p_1) = \tanh(\sum w_j E[y_j]/2T) \]

Mean-Field Theory

- We now have $n$ non-linear equations in $n$ unknowns $E[y_j]$ which can be solved deterministically by using successive approximations (without simulation!), but we still have to anneal.
- We can then use these approximations to update the weights:
  \[ D_{w_{ij}} = h(E+ [y_i] E+ [y_j] - E- [y_i] E- [y_j]) \]
  where + and - designate clamped vs. free as before.

Cauchy Machine (Szu, 1986)

- Same topology as Boltzmann machine
- Learns arbitrary spatial patterns by Hebbian encoding and fast simulated annealing
- Claimed to find min. energy with probability 1.
- Probability of neuron being 1 is
  \[ p_1 = \frac{T}{T+\langle E \rangle^2} \] vs. \[ p_1 = \frac{1}{1+\exp(-\langle E \rangle/T)} \] for Boltzmann.
- Annealing schedule is
  \[ T = T_0/(1+k) \] vs. \[ T = T_0/(1+\log k) \] (typical) for Boltzmann.

Other Hopfield Machines

- BAM (Bi-directional Associative Memory)
  (Bart Kosko, USC)
- Stores input-output patterns
- Can retrieve either direction
  - Given input, find output
  - Given output, find input
- Hopfield net divided into two tiers with behavior activating one tier then the other.