CSC321: Neural Networks

Lecture 20
Learning Boltzmann Machines

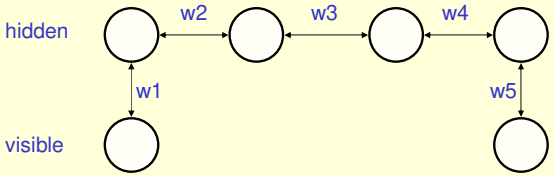
Geoffrey Hinton

The goal of learning

- Maximize the product of the probabilities that the Boltzmann machine assigns to the vectors in the training set.
  - This is equivalent to maximizing the sum of the log probabilities of the training vectors.
  - It is also equivalent to maximizing the probabilities that we will observe those vectors on the visible units if we take random samples after the whole network has reached thermal equilibrium with no external input.

Why the learning could be difficult

Consider a chain of units with visible units at the ends



If the training set is (1,0) and (0,1) we want the product of all the weights to be negative.

So to know how to change w1 or w5 we must know w3.

A very surprising fact

- Everything that one weight needs to know about the other weights and the data is contained in the difference of two correlations.

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle s_i s_j \rangle_{\mathbf{v}} - \langle s_i s_j \rangle_{free}$$

Derivative of log probability of one training vector

Expected value of product of states at thermal equilibrium when the training vector is clamped on the visible units

Expected value of product of states at thermal equilibrium when nothing is clamped

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 (where do we start?)
  - 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 is the derivative so simple?

- The probability of a global configuration at thermal equilibrium is an exponential function of its energy.
  - So settling to equilibrium makes the log probability a linear function of the energy
- The energy is a linear function of the weights and states

$$\frac{\partial E}{\partial w_{ij}} = s_i s_j$$

- The process of settling to thermal equilibrium propagates information about the weights.

Why do we need the negative phase?

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

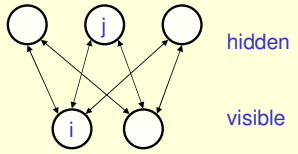
The positive phase finds hidden configurations that work well with v and lowers their energies.

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

Restricted Boltzmann Machines

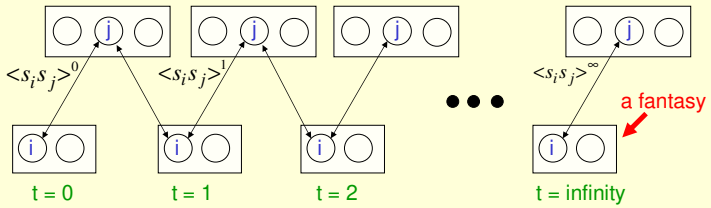
- We restrict the connectivity to make inference and learning easier.
  - Only one layer of hidden units.
  - No connections between hidden units.
- In an RBM it only takes one step to reach thermal equilibrium when the visible units are clamped.
  - So we can quickly get the exact value of :

$$< s_i s_j >_{\mathbf{v}}$$



$$p(s_j = 1) = \frac{1}{1 + e^{-\left(b_j + \sum_{i \in vis} s_i w_{ij}\right)}}$$

A picture of the Boltzmann machine learning algorithm for an RBM

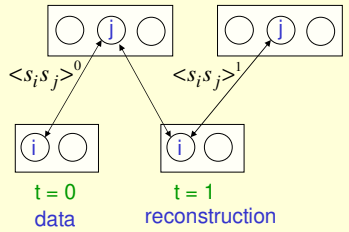


Start with a training vector on the visible units.

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

$$\Delta w_{ij} = \varepsilon \left(<s_i s_j>^0 - <s_i s_j>^\infty \right)$$

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.

$$\Delta w_{ij} = \varepsilon \left(<s_i s_j>^0 - <s_i s_j>^1 \right)$$

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

Why does the shortcut work?

- If we start at the data, the Markov chain wanders away from them data and towards things that it likes more. We can see what direction it is wandering in after only a few steps. It's a big waste of time to let it go all the way to equilibrium.
  - All we need to do is lower the probability of the "confabulations" it produces and raise the probability of the data. Then it will stop wandering away.
    - The learning cancels out once the confabulations and the data have the same distribution.
- We need to worry about regions of the data-space that the model likes but which are very far from any data.
  - These regions cause the normalization term to be big and we cannot sense them if we use the shortcut.