COMP9444
Neural Networks and Deep Learning

8b. Hopfield Networks and Boltzmann Machines

Outline

- Content Addressable Memory
- Hopfield Network
- Generative Models
- Boltzmann Machine
- Restricted Boltzmann Machine
- Deep Boltzmann Machine
- Greedy Layerwise Pretraining

Content Addressable Memory

Humans have the ability to retrieve something from memory when presented with only part of it.

For example,

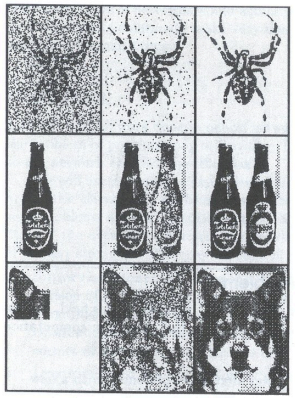
To be or not to be, ...

I came, I saw, ...

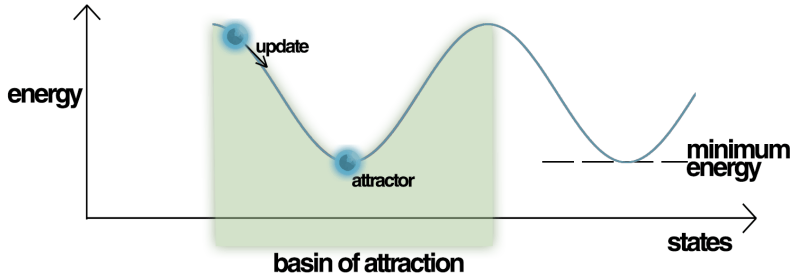
Can we recreate this in computers?

Auto-Associative Memory

We want to store a set of images in a neural network in such a way that, starting with a corrupted or occluded version, we can reconstruct the original image.

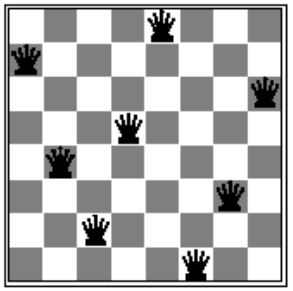


Energy Based Models



We can try to define an energy function $E(x)$ in configuration space, in such a way that the local minima of this energy function correspond to the stored items.

Constraint Satisfaction Problems

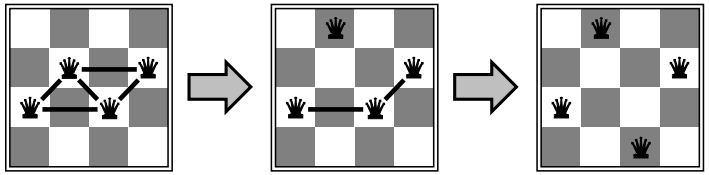


Example: Place n queens on an n-by-n chessboard in such a way that no two queens are attacking each other.

We assume there is exactly one queen on each column, so we just need to assign a row to each queen, in such a way that there are no "conflicts".

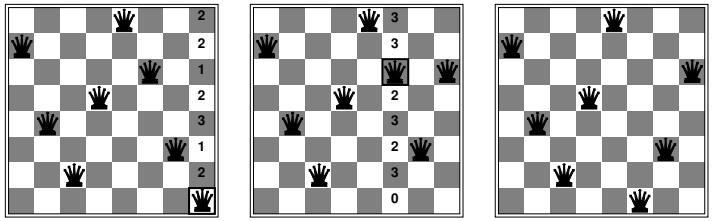
Local Search

Some algorithms for solving Constraint Satisfaction Problems work by "Iterative Improvement" or "Local Search".



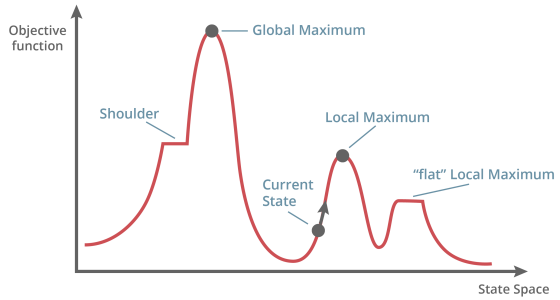
These algorithms assign all variables randomly in the beginning (thus violating several constraints), and then change one variable at a time, trying to reduce the number of violations at each step.

Hill-Climbing by Min-Conflicts



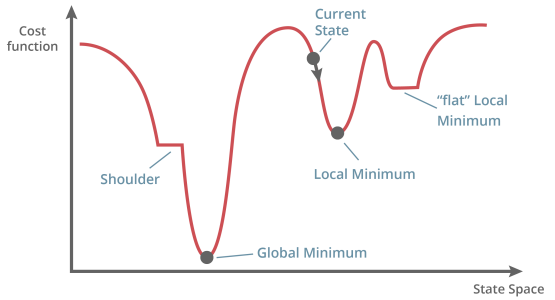
- Variable selection: randomly select any conflicted variable
- Value selection by min-conflicts heuristic
  - choose value that violates the fewest constraints

Hill-Climbing



The term Hill-climbing suggests climbing up to regions of greater "fitness".

Inverted View



When we are minimizing violated constraints, it makes sense to think of starting at the top of a ridge and climbing down into the valleys.

Energy Function for Images

Consider the set of all black-and-white images with d pixels, where each configuration x is an image $x = \{x_j\}_{1 \le j \le d}$, with

$$x_j = \begin{cases} -1, & \text{if pixel } j \text{ is black,} \\ +1, & \text{if pixel } j \text{ is white.} \end{cases}$$

We want to construct an energy function of the form

$$E(x) = -(\frac{1}{2}\sum_{i,j} x_i w_{ij} x_j + \sum_i b_i x_i)$$

such that the stored images $\{x^{(k)}\}_{1 \le k \le p}$ are local minima for $E(x)$.

The idea is to make w_{ij} positive if the two pixels x_i and x_j tend to have the same color, and make w_{ij} negative if pixels x_i and x_j tend to have opposite colors (when averaged across the set of stored images).

Hopfield Network

Consider a state space where each configuration (state) consists of a vector $x = \{x_j\}_{1 \le j \le d}$, with each $x_j =$ either $+1$ or -1

We can define an energy function as

$$E(x) = -(\frac{1}{2}\sum_{i,j} x_i w_{ij} x_j + \sum_i b_i x_i)$$

We normally assume $w_{ii} = 0$ for all i, and $w_{ij} = w_{ji}$ for all i, j.

These look very much like the weights and biases of a neural network.

But, it differs from the feedforward networks we are used to.

- The components (neurons) x_i do not vary continuously, but instead take only the discrete values -1 and $+1$
- neurons are iteratively updated, either synchronously or asynchronously, based on the current values of the neighboring neurons

Hopfield Network

$$E(x) = -(\frac{1}{2}\sum_{i,j} x_i w_{ij} x_j + \sum_i b_i x_i)$$

Start with an initial state x and then repeatedly try to "flip" neuron activations one at a time, in order to reach a lower-energy state. If we choose to modify neuron x_i, its new value should be

$$x_i \leftarrow \begin{cases} +1, & \text{if } \sum_j w_{ij} x_j + b_i > 0, \\ x_i, & \text{if } \sum_j w_{ij} x_j + b_i = 0, \\ -1, & \text{if } \sum_j w_{ij} x_j + b_i < 0. \end{cases}$$

This ensures that the energy $E(x)$ will never increase. It will eventually reach a local minimum.

Hopfield Network

Suppose we want to store p items $\{x^{(k)}\}_{1 \le k \le p}$ into a network with d neurons. We can set $b_i = 0$ and

$$w_{ij} = \frac{1}{d}\sum_{k=1}^{p} x_i^{(k)} x_j^{(k)}$$

In other words, $w_{ij} = (-1 + 2c)p/d$, where c is the fraction of training items for which $x_i^{(k)} = x_j^{(k)}$.

This is known as Hebbian learning, by analogy with a process in the brain where the connection strength between two neurons increases when they fire simultaneusly or in rapid succession.

One consequence of this choice for b_i and w_{ij} is that, if x is a stable attractor, then the negative image $(-x)$ is also a stable attractor.

Hopfield Network

Once the items are stored, then for any item $x = x^{(l)}$ we have

$$\sum_{j=1}^{d} w_{ij} x_j^{(l)} = \frac{1}{d}\sum_{j=1}^{d}\sum_{k=1}^{p} x_i^{(k)} x_j^{(k)} x_j^{(l)} = x_i^{(l)} + \frac{1}{d}\sum_j \sum_{k \ne l} x_i^{(k)} x_j^{(k)} x_j^{(l)}$$

The last term on the right is called the crosstalk term, representing interference from the other stored items. If, for all i, the crosstalk term is smaller than 1 in absolute value (or it has the same sign as $x_i^{(l)}$) then x_i will not change and $x^{(l)}$ will be a stable attractor.

Hopfield Network

- The number of patterns p that can be reliably stored in a Hopfield network is proportional to the number of neurons d in the network.
- A careful mathematical analysis shows that if $p/d < 0.138$, we can expect the patterns to be stored and retrieved successfully.
- If we try to store more patterns than these, additional, "spurious" stable states may emerge.

Generative Models

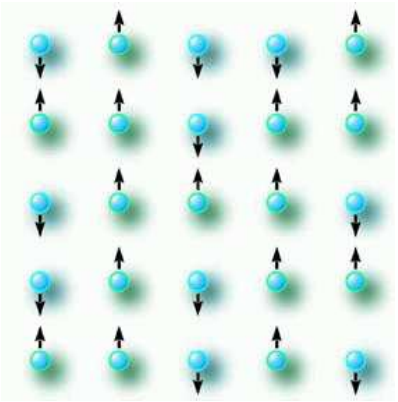
The Hopfield Network is used to store specific items and retrieve them.

What if, instead, we want to generate new items, which are somehow "similar" to the stored items, but not quite the same.

This is known as a generative model.

The first attempt to do this using neural networks was the Boltzmann Machine.

Ising Model of Ferromagnetism



Boltzmann Machine (20.1)

The Boltzmann Machine uses exactly the same energy function as the Hopfield network:

$$E(x) = -(\sum_{i<j} x_i w_{ij} x_j + \sum_i b_i x_i)$$

The Boltzmann Machine is very similar to the Hopfield Network, except that

- components (neurons) x_i take on the values $0, 1$ instead of $-1, +1$
- used to generate new states rather than retrieving stored states
- update is not deterministic but stochastic, using the sigmoid

Boltzmann Distribution

The Boltzmann Distribution is a probability distribution over a state space, given by

$$p(x) = \frac{e^{-E(x)/T}}{Z}$$

- $E(x)$ is an energy function
- T is a temperature parameter
- Z is the partition function which ensures that $\sum_x p(x) = 1$

In most cases, it is too complicated to compute the partition function directly. But, we can sample from the distribution by an iterative process using the relative probability of neighboring states.

Gibbs Sampling (16.3)

Consider a state x for which a particular component x_i is equal to 1. Suppose we change x_i to 0 but leave all other components fixed, to produce a new state x'. Let $\Delta E = E(x') - E(x)$ be the difference in energy between the two states. Then

$$p(x') = p(x)\, e^{-\Delta E/T}$$

Therefore, if all other components stay fixed, the probability of x_i taking the value 1 or 0 must be

$$p(x_i = 1) = \frac{p(x)}{p(x) + p(x')} = \frac{1}{1 + e^{-\Delta E/T}}$$

$$p(x_i = 0) = 1 - p(x_i = 1) = \frac{1}{1 + e^{+\Delta E/T}}$$

Boltzmann Machine

The Boltzmann Machine operates similarly to a Hopfield Network, except that there is some randomness in the neuron updates.

In both cases, we repeatedly choose one neuron x_i and decide whether or not to "flip" the value of x_i.

For the Hopfield Network, x_i will change from 1 to 0 only if $\Delta E < 0$, and will change from 0 to 1 only if $\Delta E > 0$, i.e. we never move to a higher energy state. For the Boltzmann machine, we instead choose $x_i = 1$ with probability

$$p = \frac{1}{1 + e^{-\Delta E/T}}$$

In other words, there is some probability of moving to a higher energy state (or remaining in a higher energy state when a lower one is available).

Boltzmann Machine

$$p(x_i \to 1) = \frac{1}{1 + e^{-\Delta E/T}}$$

- if this process is repeated for many iterations, we will eventually obtain a sample from the Boltzmann distribution
- when $T \to \infty$, the value of 0 or 1 is always chosen with equal probability, thus producing a uniform distribution on the state space
- as $T \to 0$, the behaviour becomes similar to that of the Hopfield Network (never allowing the energy to increase)
- the Temperature T may be held fixed, or it may start high and be gradually reduced (known as Simulated Annealing)

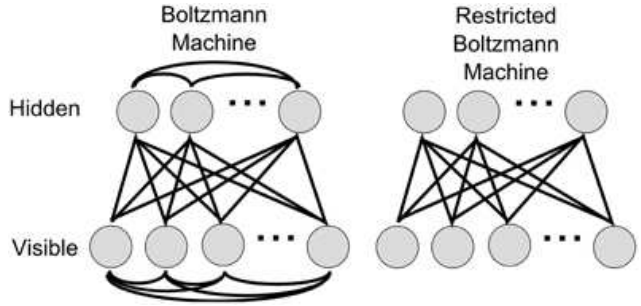
Boltzmann Machine Limitations

The Boltzmann Machine is limited in that the probability of each unit must be a linearly separable function of the surrounding units. It becomes more powerful if we make a division between "visible" units v and "hidden" units h.

The visible and hidden units roughly correspond to input and hidden units in a feedforward network. The aim is that the hidden units should learn some hidden features or "latent variables" which help the system to model the distribution of the inputs.

Restricted Boltzmann Machine (16.7)

If we allow visible-to-visible and hidden-to-hidden connections, the network takes too long to train. So we normally restrict the model by allowing only visible-to-hidden connections.



This is known as a Restricted Boltzmann Machine.

Restricted Boltzmann Machine

- inputs are binary vectors
- two-layer bi-directional neural network
  - visible layer v
  - hidden layer h
- no vis-to-vis or hidden-to-hidden connections
- all visible units connected to all hidden units

$$E(v,h) = -(\sum_i b_i v_i + \sum_j c_j h_j + \sum_{i,j} v_i w_{ij} h_j)$$

- trained to maximize the expected log probability of the data

Conditional Distributions (20.2)

Because the input and hidden units are decoupled, we can calculate the conditional distribution of h given v, and vice-versa.

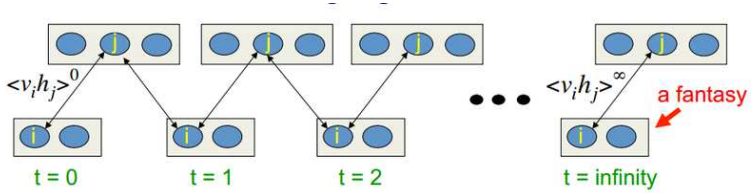
$$p(h|v) = \frac{p(v,h)}{p(v)} = \frac{1}{p(v)}\frac{1}{Z}\exp(\sum_i b_i v_i + \sum_j c_j h_j + \sum_{i,j} v_i w_{ij} h_j)$$
$$= \frac{1}{Z'}\exp(\sum_j c_j h_j + \sum_{i,j} v_i w_{ij} h_j)$$

It follows that

$$p(h|v) = \prod_j p(h_j|v) = \prod_j \sigma\big((2h-1)\odot(c+W^{\mathrm{T}}v)\big)_j$$
$$p(v|h) = \prod_i p(v_i|h) = \prod_i \sigma\big((2v-1)\odot(b+Wh)\big)_i$$

where $\odot$ is component-wise multiplication and $\sigma(s) = 1/(1+\exp(-s))$ is the sigmoid function.

Alternating Gibbs Sampling



With the Restricted Boltzmann Machine, we can sample from the Boltzmann distribution as follows:

choose v_0 randomly
then sample h_0 from $p(h|v_0)$
then sample v_1 from $p(v|h_0)$
then sample h_1 from $p(h|v_1)$
etc.

Contrastive Divergence (18.2)

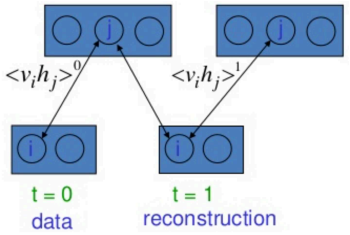
RBM can be trained by Contrastive Divergence

- select one or more positive samples $\{v^{(k)}\}$ from the training data
- for each $v^{(k)}$, sample a hidden vector $h^{(k)}$ from $p(h \mid v^{(k)})$
- generate "fake" samples $\{\tilde{v}^{(k)}\}$ by alternating Gibbs sampling
- for each $\hat{v}^{(k)}$, sample a hidden vector $\tilde{h}^{(k)}$ from $p(h \mid \tilde{v}^{(k)})$
- Update $\{b_i\}, \{c_j\}, \{w_{ij}\}$ to increase $\log p(v^{(k)}, h^{(k)}) - \log p(\tilde{v}^{(k)}, \tilde{h}^{(k)})$

$$b_i \leftarrow b_i + \eta(v_i - \tilde{v}_i)$$
$$c_j \leftarrow c_j + \eta(h_j - \tilde{h}_j)$$
$$w_{ij} \leftarrow w_{ij} + \eta(v_i h_j - \tilde{v}_i \tilde{h}_j)$$

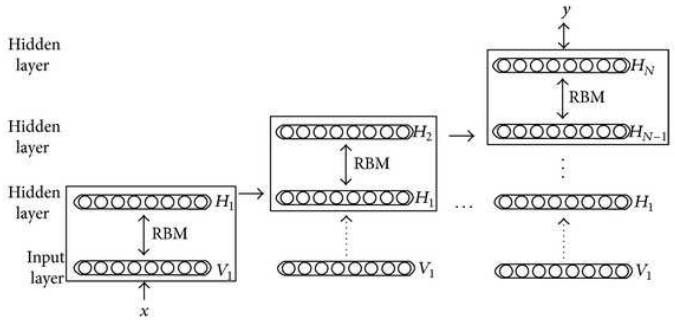
Quick Contrastive Divergence

It was noticed in the early 2000's that the process can be sped up by taking just one additional sample instead of running for many iterations.



- v_0, h_0 are used as positive sample, and v_1, h_1 as negative sample
- this can be compared to the Negative Sampling that was used with word2vec – it is not guaranteed to approximate the true gradient, but it works well in practice

Deep Boltzmann Machine (20.4)



The same approach can be applied iteratively to a multi-layer network. The weights from the input to the first hidden layer are trained first. Keeping those fixed, the weights from the first to the second hidden layer are trained, and so on.

Greedy Layerwise Pretraining

One application for the deep Bolzmann machine is greedy unsupervised layerwise pretraining.

Each pair of layers in succession is trained as an RBM.

The resulting values are then used as the initial weights and biases for a feedforward neural network, which is then trained by backpropagation for some other task, such as classification.

For the sigmoid or tanh activation function, this kind of pre-training leads to a much better result than training directly by backpropagation from random initial weights.