Boltzmann Machine

David Barber
Department of Computer Science
University College London

October 19, 2017

1 Undirected Models

We consider 'undirected' models of the form

$$p(x) = \frac{1}{Z_\theta} e^{E_\theta(x)}$$

where the normalisation term (partition function) is given by, in the case of discrete x,

$$Z_\theta = \sum_x e^{E_\theta(x)}$$

For continuous x, Z the sum over the space would be replaced by an integral. We focus here on discrete x.

Unfortunately, except in very special cases, Z is not computationally tractable and we cannot calculate $p(x)$. For this reason approximation methods are central in training undirected models and typically we will use sampling methods.

1.1 Gradient Based approaches to learning the parameter θ

For notational simplicity, we'll write simply $E(x)$ and Z. The log likeilhood then of a state x is given by

$$\log p(x) = E(x) - \log Z$$

Consider a set of training examples $x^1, \ldots, x^N$ and that we wish to find θ to maximise the (scaled) log likelihood for the whole dataset

$$L(\theta) = \frac{1}{N} \sum_{n=1}^{N} \log p(x^n)$$

The gradient is then (exercise that we did on the whiteboard)

$$\frac{\partial L}{\partial \theta} = \frac{1}{N} \sum_n \frac{\partial E(x^n)}{\partial \theta} - \sum_x p(x) \frac{\partial E(x)}{\partial \theta}$$

One can write this more compactly as

$$\frac{\partial L}{\partial \theta} = \left\langle \frac{\partial E(x)}{\partial \theta} \right\rangle_{\hat{p}} - \left\langle \frac{\partial E(x)}{\partial \theta} \right\rangle_{p} \tag{1}$$

where $\hat{p}$ is the empirical data distribution

$$\hat{p}(x) = \frac{1}{N} \sum_{n=1}^{N} \delta\left(x, x^n\right)$$

and $\langle f(x) \rangle_p$ denotes expectation of $f(x)$ with respect to a distribution $p(x)$. $\delta(x, y)$ is the Kronecker Delta function which is 1 if $x = y$ and zero if $x \neq y$.

In general it will not be possible to find the expecation $\left\langle \frac{\partial E(x)}{\partial \theta} \right\rangle_p$ exactly and approximations are required. One approach is to (approximately) draw samples $\tilde{x}^s$, $s = 1, \ldots, S$ from p and then approximate the expectation using these samples

$$\left\langle \frac{\partial E(x)}{\partial \theta} \right\rangle_p \approx \frac{1}{S} \sum_{s=1}^{S} \frac{\partial E(\tilde{x}^s)}{\partial \theta}$$

We'll describe a simple way to draw these approimate samples efficiently for the Boltzmann Machine model in a later section.

1.2 The curvature of L

Taking the average over the training points we can write the Hessian of L as

$$\frac{\partial^2 L}{\partial \theta_i \partial \theta_j} = \left\langle \frac{\partial^2}{\partial \theta_i \partial \theta_j} E(x) \right\rangle_{\hat{p}} - \left\langle \frac{\partial^2}{\partial \theta_i \partial \theta_j} E(x) \right\rangle_p - \left\langle \frac{\partial E(x)}{\partial \theta_i} \frac{\partial E(x)}{\partial \theta_j} \right\rangle_p + \left\langle \frac{\partial E(x)}{\partial \theta_i} \right\rangle_p \left\langle \frac{\partial E(x)}{\partial \theta_j} \right\rangle_p$$

The final two terms are negative covariance elements and thus Negative Semidefinite (NSD). Indeed, it's easy to show that the final two terms are just the Fisher Information matrix elements of $p(x)$. Note that in this case it makes sense to use the Fisher Information as an approximate curvature matrix since, for a well fitting model, the expectation with respect to the training data and the model should be close to zero, meaning that the Hessian is approximately given by the curvature matrix. In practice, of course, we cannot calculate these expectations exactly, so we would need to approximate them either by sampling from p or from the empirical distribution.

In general, therefore, L does not have a unique optimum since the Hessian will not be negative semidefinite. However, as we will see below, there are special cases (exponential family models) for which the Hessian is NSD and the objective has a unique optimum.

2 The Boltzmann Machine

The Boltzmann Machine is defined by an undirected model with function

$$E(x) = x^{\mathsf{T}} W x, \qquad x_i \in \{0, 1\}, \qquad i = 1, \ldots, D$$

Here the W matrix is playing the role of the parameter θ. Note that in this case the 2nd derivative wrt W is zero and hence the Hessian is NSD and the log likelihood is concave. (Indeed, for any exponential family modelled represented in canonical form, the second order parameter derivative is zero and hence the log likelihood is concave.)

2.1 Boltzmann-Gibbs distribution

For a state x with energy $E(x)$ and system temperature T, the probability of the system being in state x is

$$p(x) \propto e^{-\frac{E(x)}{kT}}$$

where k is Boltzmann's constant. The Boltzmann Machine (BM) is named in honour of Boltzmann (even though it came many years after his death) since it is of the form of a Boltzmann distribution.

2.2 Hidden Units

We can partition x into a collection of 'visible' units v (variables that we will have observed values for) and 'hidden' units h (variables that we can never observe). In this case

$$x = \begin{pmatrix} v \\ h \end{pmatrix}$$

(a) Ludwig Eduard Boltzmann (1844-1906)

(b) With his wife Henriette von Aigentler

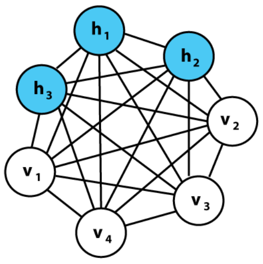

Figure 1: A Boltzmann Machine with both hidden and visible units represented as a Markov network. From Wikipedia.

and we can write W as a partioned matrix

$$W = \left(\begin{array}{c|c} W_{vv} & W_{vh} \\ \hline W_{hv} & W_{hh} \end{array} \right)$$

Note that, without loss of generality, we may assume that W is symmetric. In this case we can write $E(x)$ in terms of the vectors v and h as

$$E(v, h) = v^\mathsf{T} W_{vv} v + 2v^\mathsf{T} W_{vh} h + h^\mathsf{T} W_{hh} h$$

See figure(1) for the Graphical Model representation of the BM in terms of an undirected graph. In general we will have an edge in the graph between and two variables x_i and x_j if $W_{ij} \neq 0$.

2.3 AI and the Boltzmann Machine

The BM has a long history in AI. If one considers the variables as analogous to 'neurons' that are either firing or not, the BM looks like an ensemble of neurons.

The physicist John Hopfield (Neural networks and physical systems with emergent collective computational abilities, Proc. Natl. Acad. Sci. USA, vol. 79 no. 8, pp. 25542558, April 1982.) was perhaps the first to make this connection and this attracted widespread involvement from theoretical physicists (in particular statistical mechanics researchers) to the study of such 'neural systems'.

Others, including AI researchers/psychologists Hinton and Sejnowski made early connections to this work, for example (Geoffrey E. Hinton and Terrence J. Sejnowski, Analyzing Cooperative Computation. In Proceedings of the 5th Annual Congress of the Cognitive Science Society, Rochester, New York, May 1983.).

One can show that, provided there are sufficient number of hidden units, the distribution $p(v)$ is very expressive and can model pretty much any distribution on the binary hypercube $\{0.1\}^D$.

2.4 Gibbs Sampling

Consider a distribution $p(x_1, x_2)$. A simple way to sample from a distribution is to write

$$p(x_1, x_2) = p(x_1|x_2)p(x_2)$$

This means that we can sample a value for x_1, x_2 by first sampling a value for x_2 and then sampling x_1 from $p(x_1|x_2)$. Similarly, then given x_1 we can sample from $p(x_2|x_1)$. This procedure in which we sample from the conditional

$$p(x_i|x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$$

and cycle through the variables x_i is called Gibbs sampling.

For the BM we can readily derive that the conditional is given by

$$p(x_i = 1|x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = \sigma\left(W_{ii} + 2\sum_{j\neq i} W_{ij}x_j\right)$$

where $\sigma(x) = 1/(1 + e^{-x})$.

This sampling procedure looks very reminiscent of how neurons update, see figure(2) in which, as potential builds up at the soma of a neuron, it becomes increasingly likely to fire.

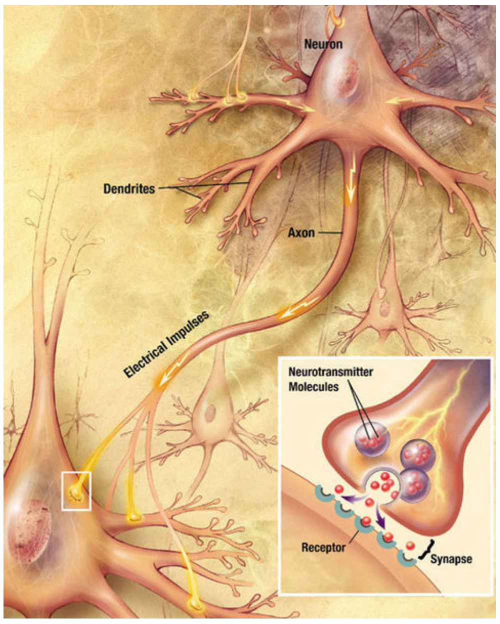

Figure 2: As potential builds up at a neuron, it becomes increasingly likely to fire. As it fires, it sends a signal to neurons that it is connected to and the amount that the firing influences the connected neuron (its 'weight') is mediated through how many neurotransmitters are released. These influences roughly add up linearly at the soma (body) of the neuron which in turn then becomes more likely to fire. `http://mindsparke.com/brain-fitness-training/the-structure-and-function-of-brain-cells/`.

2.5 The Hopfield Net

The Hopfield Net is a limiting deterministic case of Gibbs sampling from the BM in which the sigmoid function $\sigma(x)$ becomes a step function. Each neuron updates according to the rule

$$x_i(t+1) = \begin{cases} 1 \text{ if } W_{ii} + 2\sum_{j\neq i} W_{ij}x_j > 0 \\ 0 \text{ otherwise} \end{cases}$$

The neurons can be updated either synchronously or in sequence (a la Gibbs sampling).

This forms a dynamical system and the aim is to find weights W such that a set of patterns $x^1, \ldots, x^N$ are attractors for this system. That is, if we start in a perturbed form of x^n, under the Hopfield dynamics, it will converge back to x^n.

A classical way to set W is called the Hebb rule

$$W_{ij} = \frac{1}{N}\sum_{n=1}^{N} x_i^n x_j^n$$

This works OK, but has spurious attractors.

A much better approach (for the stochastic synchronous Hopfield net) is to simply set W by maximising the conditional likelihood of patterns $p(x_{t+1} = x^n | x_t = x^n)$, see the BRML textbook, chapter 26.

3 Contrastive Divergence

In the gradient rule (1) we need to find the expectation wrt the model. In general this is difficult and usually is approximated by sampling (for example Gibbs sampling).

Contrastive Divergence is an approximate sampling algorithm in which one starts the sampler at one of the training datapoints and then runs only a small (usually just one) sampling step (rather then running to convergence). The intuition is that, for a well trained model, the model should generate samples that are close to the training data, so by initialising the samples close to the training data, this will be consistent with a well trained model, moving the sampler immediately to a relevant part of the space.

In the experiments below we use a form of CD to approximate the gradient required for training.

4 Demonstration

We look at training a fully visible BM on the MNIST dataset. We look then at a small problem in which we have only 10 training examples (one from each of the 10 digit classes) and later a ‘large’ problem in which all 60,000 MNIST training points are used.

4.1 Reconstructing Missing Pixels

For each of the training images, we randomly remove a percentage of the pixels to form an incomplete image. We then solve the problem

$$\max_{x_{miss}} p(x_{vis}, x_{miss})$$

For the BM this can be phrased as a Binary Quadratic Programme. Rather than using a BQP solver we simply instantiate the missing pixels at random and then examine the two states $x_i = 0$ and $x_i = 1$ for a missing pixel i. The state with larger (unnormalised) probability is determined the winner and another variable x_j selected. This process of axis aligned optimisation is repeated to convergence. Some example corruptions and the completed images are presented in figure(3).

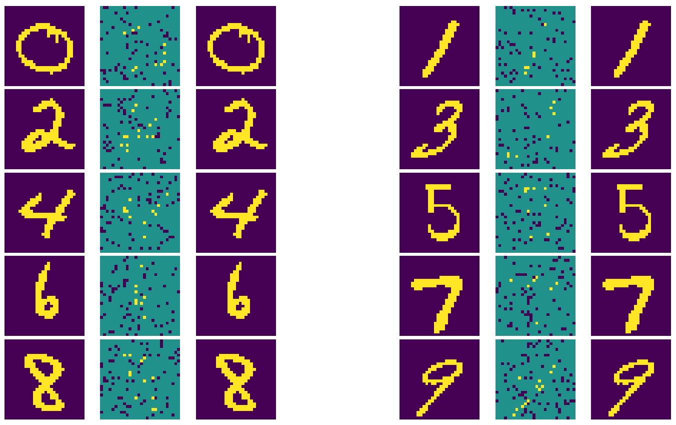

Figure 3: 'Small' MNIST problem. Completing images using the gradient mixture approach. On the left in each panel is the original image, in the centre is the image with missing pixels in light blue ($p_{miss} = 0.9$), on the right is the recovered image. The images are perfectly completed. Gradient based training (with momentum) was used with 300 iterations.

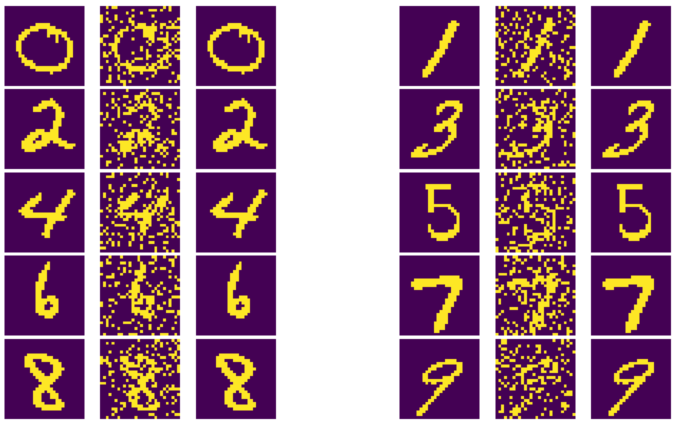

Figure 4: 'Small' MNIST problem. Denoising corrupted images using the mixture of gradient approach. On the left in each panel is the original image, in the centre is the corrupted image ($p_{flip} = 0.2$), on the right is the recovered image. The images are perfectly recovered. Gradient based training (with momentum) was used with 300 iterations.

4.2 Cleaning up Corrupted Pixels

We assume that independently each pixel is potentially flipped with probability γ according to the distribution

$$p(y|x) = \prod_i p(y_i|x_i)$$

where $p(y_i \neq x_i|x_i) = p_{flip}$. Then the decoding problem is, given an observed noisy image y to find

$$\arg\max_x \log p(y|x)p(x)$$

where $p(x)$ is our learned BM. This is also a Binary Quadratic Programme. However, we use a simple approximate solver that, starting from setting $x = y$, simply check each of the two states of a selected pixel i and set x_i to the value with highest $p(y|x)p(x)$, cycling around the pixels until convergence. Some example reconstructions are given in figure(4).

5 The Restricted BM (Harmonium)

The RBM (orginally called the Harmonium in the work of Smolensky) is defined by

$$E(v, h) = v^{\mathsf{T}} W_{vh} h + a^{\mathsf{T}} v + b^{\mathsf{T}} h$$

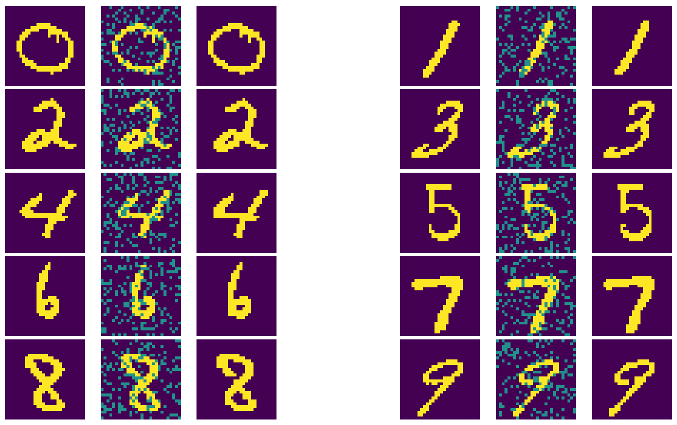

Figure 5: ‘Large’ MNIST problem. Training on the full 60,000 MNIST digits. Completing images using the KL mixture approach. On the left in each panel is the original image, in the centre is the image with missing pixels in light blue ($p_{miss} = 0.2$), on the right is the recovered image. Gradient based training (with momentum) was used with 50,000 iterations.

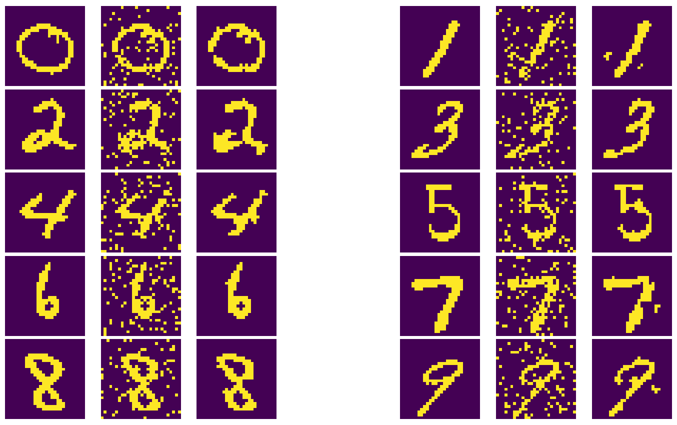

Figure 6: ‘Large’ MNIST problem.Training on the full 60,000 MNIST digits and Denoising corrupted image. On the left in each panel is the original image, in the centre is the corrupted image ($p_{flip} = 0.1$), on the right is the recovered image. Gradient based training (with momentum) was used with 50,000 iterations.

As a Markov Network this means that $p(v, h)$ can be written as a bipartite graph in which there are connections only between the v and h variables (and no connections within the h variables and no connections with the v variables).

THe RBM has the property that GIbbs sampling can be efficiently applied since

$$p(v|h) = \prod_{i=1}^{V} p(v_i|h)$$

where

$$p(v_i = 1|h) = \sigma\left(a_i + \sum_{j=1}^{H} [W_{vh}]_{i,j} h_j\right)$$

This means that we can draw, in parallel, a set of values for the whole vector v from $p(v|h)$.

Similarly, it is easy to show that

$$p(h|v) = \prod_{i=1}^{H} p(h_i|v)$$

where

$$p(h_i = 1|v) = \sigma\left(b_i + \sum_{j=1}^{V}[W_{vh}]_{i,j}v_j\right)$$

and we can then draw samples h in parallel given v. Thus, from a random initialisation v^0, we can then draw successive vectors by Gibns (alternating between drawing from h and drawing from v) $v^0, h^1, v^2, h^3, \ldots$.

Thanks to the existence of an efficient Gibbs sampling procedure, the RBM has found many applications when hidden units are required. It is straightforward to show that the gradient requires expectations with respect to the distributions $p(h|v)$ and marginal $p(v)$, both of which can be efficiently approximated by Gibbs sampling in the RBM. In general, however, it remains an active research topic to find truly fast and effective training procedures for BMs both with and without hidden units.