Boltzmann Machines

Team 4 **Natali Alfonso, Simon Kern, Vangelis Kostas**
Machine Learning (NWI-NM048C)
Prof. B. Kappen Due date: 31/01/2017

Introduction

A Boltzmann Machine (BM) is a symmetrically connected network of binary neurons that can learn stochastically from a set of binary-encoded data. From a statistical physics lens, these networks can be conceived as entropy-driven, energy-based systems that tend to low-energy configurations. Interestingly, under either parallel or sequential stochastic dynamics, at low temperature, the probability distribution over the states of all neurons in the network converge to the Boltzmann-Gibbs distribution. For random sequential dynamics, this distribution is given by

$$p(s) = \frac{1}{Z}\exp(-E(s)) \tag{1}$$

where states of low energy have a high probability. The energy is defined as

$$E(s) = \frac{1}{2}\sum_{i,j} w_{i,j} s_i s_j \sum_i \theta_i s_i \tag{2}$$

and Z is the normalizing constant, also known as the partition function

$$Z = \sum_s \exp(-E(s)) \tag{3}$$

Boltzmann machines can solve the computational problem of *learning* by means of simple gradient-based algorithms. We can obtained the necessary statistics to compute the gradients from the analytical expression of the stationary probability distribution $p(s)$, i.e. the *mean firing rate* of a neuron

$$m_i = \langle s_i \rangle = \sum_s s_i p(s) \tag{4}$$

and the correlations between neurons

$$\chi_{i,j} = \langle s_i s_j \rangle - \langle s_i \rangle \langle s_j \rangle = \sum_s s_i s_j p(s) - m_i m_j \tag{5}$$

Likewise, we compute the *clamped* statistics from a training set P with pattern $s^\mu = (s_1^\mu, ..., s_n^\mu)$ with $\mu = 1, 2, ..P$,

$$\langle s_i \rangle_c = \frac{1}{P}\sum_\mu s_i^\mu \tag{6}$$

$$\langle s_i s_j \rangle_c = \frac{1}{P}\sum_\mu s_i^\mu s_j^\mu \tag{7}$$

These gradients maximize the log likelihood of the observed data $L(w, \theta) = \frac{1}{P} P(s_1^\mu, ..., s_n^\mu)$ w.r.t w and θ

$$\frac{\partial L}{\partial w} = \langle s_i \rangle_c - \langle s_i \rangle \tag{8}$$

$$\frac{\partial L}{\partial \theta} = \langle s_i s_j \rangle_c - \langle s_i s_j \rangle \tag{9}$$

and are later added to randomly selected weights and thresholds to update the learning process

$$w_{i,j}(\tau + 1) = w_{i,j}(\tau) + \eta \frac{\partial L}{\partial w} \tag{10}$$

$$\theta_i(\tau + 1) = \theta_i(\tau) + \eta \frac{\partial L}{\partial \theta} \tag{11}$$

These computations become *intractable*, $O(2^n)$, for larger networks due to the sum over all possible states in equations (3), (4) and (5). A tractable and more efficient learning method for BMs is *the mean field approximation*. By means of the so-called *mean field equations* and the *linear response correction* to approximate m_i and $\chi_{i,j}$, respectively, we can compute the solution in closed form, in one learning step. The equations are summarized below:

$$m_i = \langle s_i \rangle_c \tag{12}$$

$$\chi_{i,j} = \langle s_i s_j \rangle_c - \langle s_i \rangle_c \langle s_j \rangle_c \tag{13}$$

$$w_{i,j} = \frac{\delta_{i,j}}{1 - m_i^2} - C_{i,j} \tag{14}$$

$$\theta_i = \tanh^{-1}(m_i) - \sum_{j=1}^{n} w_{i,j} m_j \tag{15}$$

where C is the *linear response*, $C = (\chi^{-1})_{i,j}$, with $C_{i,j}$ corrected when $i = j$. The correction is obtained from $\chi_{i,i} = 1 - m_i^2$. These n terms, thus, become an additional constraint on w.

A) Boltzmann Machines

Problem statement

We trained a general BM on a 1000 randomly-generated, binary samples. The network consists of 10 neurons, all fully-connected, trained in two steps: a first step were we compute the *clamped* statistics from the data, followed by a sequential Glauber dynamics to sample from (1) to ultimately compute the *free* statistics (4) and (5). Glauber dynamics is a Markov chain that sequentially updates the states of the network according to a flip operator. At each iteration t, the probability of the network to transition to state S' at $t + 1$ is,

$$T(\mathbf{S'}|\mathbf{S}) = \frac{1}{N} p(S_i', t + 1|\mathbf{S}, t) \quad \text{for the } i^{th} \text{ neuron} \tag{16}$$

$$p(S_i', t + 1|\mathbf{S}, t) = \sigma(S_i', h_i(\mathbf{S(t)})) \tag{17}$$

$$h_i(\mathbf{S}) = \sum_{j \neq i} w_{ij} S_i + \theta_i \tag{18}$$

We accept the new state using a Gibbs sampler, i.e if $T(\mathbf{S'}|\mathbf{S}) > \mathit{rand}(0, 1)$ and we do this for 500 iterations per learning step. The learning step k is set to 200.
The weights and thresholds are adapted with gradient-descent as defined in equations (10) and (11), where η is set to $1e^{-3}$.

Results

We evaluate the performance of the BM by the magnitude of the gradients scaled by the learning step $\eta(\frac{\partial L}{\partial w} + \frac{\partial L}{\partial \theta})$. That is, we keep track of the absolute changes in weights and the bias term along learning iterations. For 500 sampling steps (200 burn-in samples) and 200 learning steps, the sum of the absolute change in weights converges to ~ 1.03.

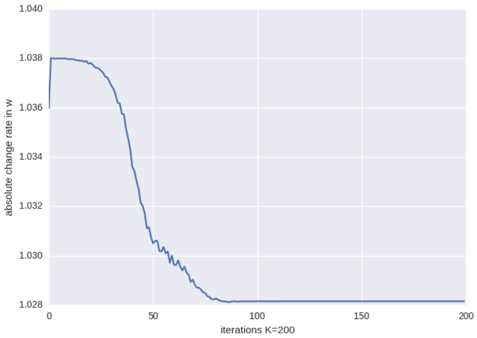


Figure 1: Absolute change in weights at every learning step.

Research questions

Computational speed. What is the time-complexity of a general BM w.r.t the number of neurons?

The results turn out as expected: the complexity is exponential in the number of neurons. Most of the overhead comes from computing the correlations between neurons χ_{ij}.

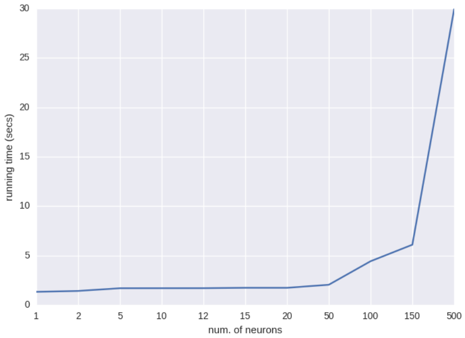


Figure 2: Computational speed of a BM w.r.t the number of visible neurons.

Is there a minimum burn-in period?

Is not a trivial task to determine how many samples will be tossed away (*burn-in*) before a Markov chain gives reasonably highly probable samples. To find out, we experimented with different number of burn-in samples in the sampling phase. Every learning step samples 500 points from the stationary distribution and the burn-in period is set to a different value at every trial-run.

The results are collected in the figure below. A burn-in period of 300-400 samples showed to give the best results. Shorter burn-in periods converged to less optimal solutions with a higher

absolute Δw.
Surprisingly, all trial-runs converged to their final value at the same time, at around 70 iterations. We would have expected to have a delay in convergence for trials with less burn-in, since it would have required more samples to compensate for including the first "bad", misguiding samples.

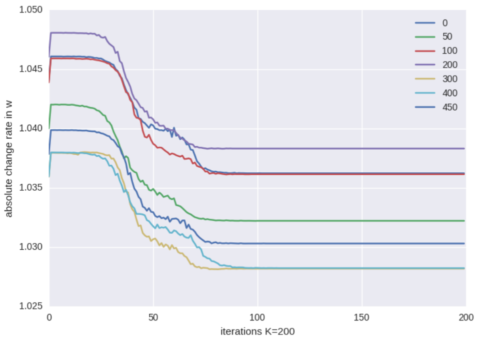


Figure 3: Absolute change in weights for all trials using different burn-in periods

What is the optimal learning rate? Does *momentum* aid the learning process?

We set the learning rate to a decreasing number of values $[1e^{-1}, 1e^{-8}]$ during the learning process of the exact same randomly generated dataset. We plot the Δw of the last learning iteration for every trial-run. The results are showed in the figure below.

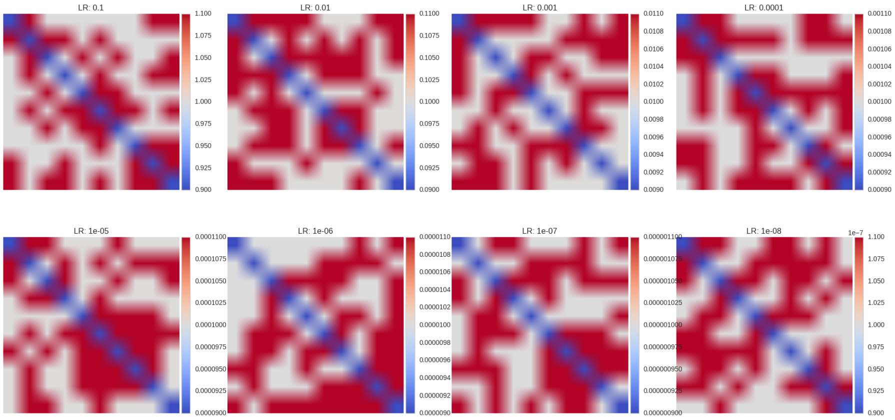


Figure 4: Absolute change for every w_{ij} using decreasing values of learning step sizes.

Interestingly, it seems that regardless the size of the learning step, all BMs converged to a local minima on the first iterations. The absolute changes in the weights remain steady for the most part. It might be that the energy (error function) landscape is that of a multi-modal one, showing many "valleys" with similar depth. Moreover, the symmetric nature of these gradients hint that the classifier learned some features from the input (BMs are acyclic graph, hence the symmetry), given that initialization of weights was randomized.
It had to be noted that the magnitude of these changes are ever lower when the learning rate is decreased. The weights rarely change with a step size of $1e^{-8}$, reaching a low-energy configuration of the model. When the learning rate is set to $1e{-}6$ and lower, the magnitude of the

gradients resemble among the trial-runs most likely disclosing the true internal structure of the input data.

We now set the learning rate to default $1e^{-3}$ again and added momentum to the change in weights. We applied different magnitudes of momentum to test if the learning process can be improved. Indeed, 20% of momentum improved largely the results in comparison to adding no momentum whatsoever. However, a higher amount of momentum worsen the performance (from 60% momentum and up). See the figure below for an overview of the results.

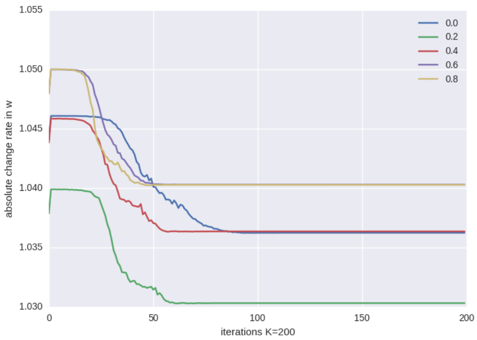


Figure 5: Absolute change in weights when different levels of momentum is added to the gradients.

Does different weight initializations affect the speed of convergence and/or the performance?

To test how weight initialization affect the results, we conducted four different experiments where we apply a different weight initialization technique on each. Every experiment consisted of 10 trial-runs from which the mean and the third of a standard deviation was plotted ($\frac{1}{3}\sigma$ was plotted for visualization reasons). The following techniques were applied:

- **Orthogonal Weights:** A Singular Value Decomposition (SVD) was applied to a randomly generated, $rand(0,1)$, weight matrix M: $\text{SVD}(M) = U\Sigma V^*$, where the unitary (conjugate) matrix U was used to initialize the weights.
- **Gaussian Sampling:** Sample random weights w_{ij} according to a zero-mean Gaussian distribution with $\sigma = 0.1$.
- **Xavier initialization:** Sample random weights w_{ij} according to a Uniform distribution between $(-r, r)$, where $r = 4\sqrt{\frac{6}{in+out}}$. The terms *in* and *out* stand for the number of input and output connections of a neuron, respectively.
- **Random:** Randomly generated w_{ij} between $(0,1)$.

The results are gathered in the figures below.

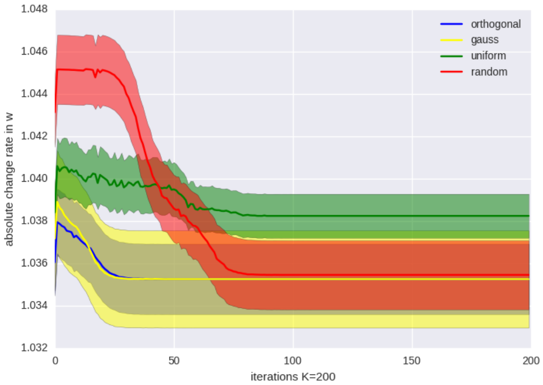


Figure 6: μ and $\frac{1}{3}\sigma$ of all 4 experiments. Each of them employs a different weight initialization method. The figure shows the absolute change in weights of all experiments.

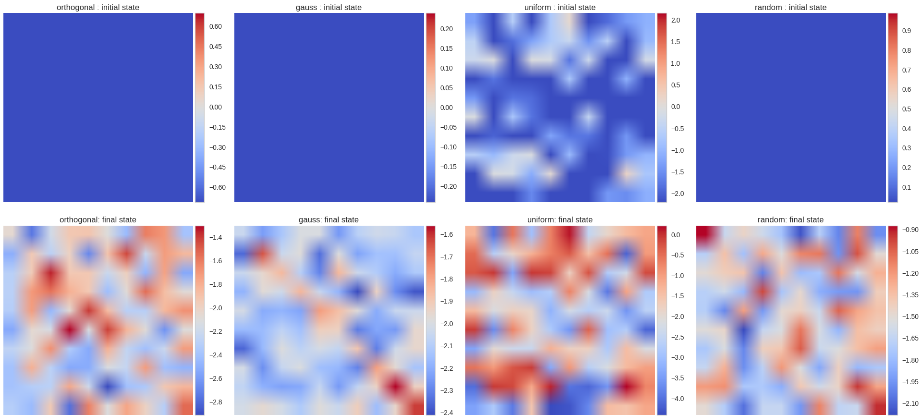


Figure 7: Initial (first row) and final configurations of the weights for an arbitrary trial picked from each experiment.

It seems that a good initialization method is to initialize the weights to either an orthogonal matrix or from a Gaussian distribution. The Orthogonal method showed the best performance in average across all trials, even though a Gaussian initialization set the grounds for a better performance in some trials.
By looking at the weight matrix, both Orthogonal and Gaussian techniques converge to large valued weights. This indicates that, with the same set up, the BM was able to over-fit the input data better than when the weights are initialized randomly or with Xavier method. That is, it learned from the data "too well". Moreover, the symmetry of the Gaussian-initialized weights leads to think that this techniques might be right choice.

B) Mean Field Approximation to Boltzmann Machines

Problem statement

For this section we trained 10 different BMs for all 10 different classes in the MNIST dataset. The MNIST dataset consist of 60000 training and 10000 testing picture samples of handwritten digits from 0 to 9. We used 500 samples from the testing set to validate during the training

phase.The pictures are binary 28x28 sized pictures with 10% of noise added. We used fully-connected networks of 784 input neurons each.
The mean field approximation is applied to BMs to train them using equations (12)-(15).

Results

The performance on the validation set (500 samples) is 93.4% and 94.2% on the test set (4500 samples). That is, 33 samples were classified incorrectly in the validation set. A slight improvement from what was expected (vs. 45 incorrect samples claimed in the exercise).
The results are collected in the plots below.

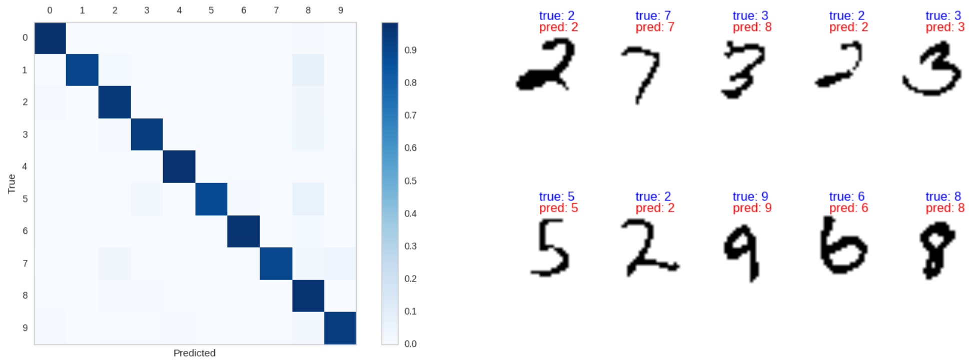


Figure 8: Confusion matrix for all MNIST classes and a few randomly picked predictions.

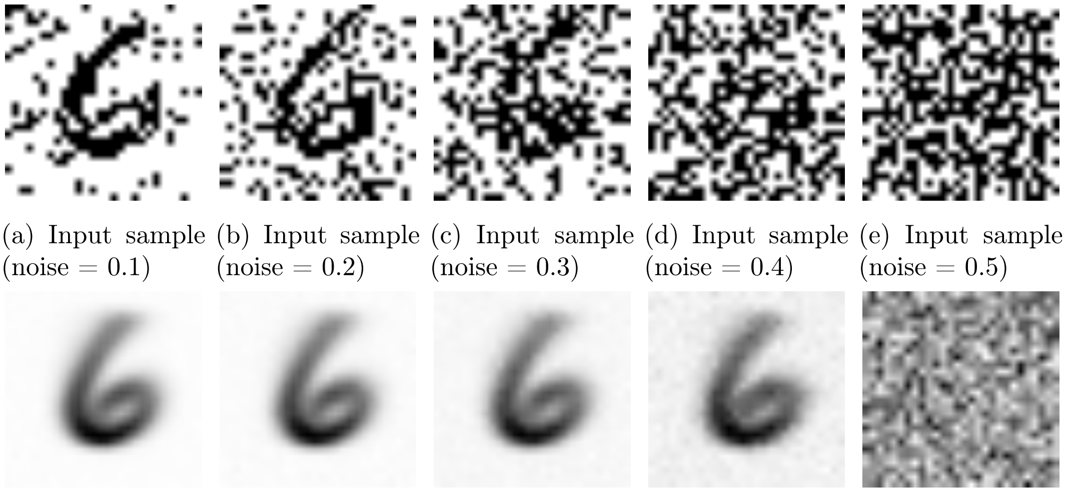
Research questions

Does it converge faster than sequential Glauber dynamics?

We run a fully-connected BM with sequential Glauber dynamics (as described in the previous section) parallely to 10 BMs with Mean Field Approximation (MFA) and compare their training times. For a fair comparison, we used the same number of input neurons and the same number of training samples as the MNIST dataset.
The differences are astonishing: BMs with Glauber dynamics take 111.57 seconds to learn through 200 iterations, meanwhile the MFA to BMs take in total 0,9515 seconds to approximate all classes. This is accomplished in one step, and with no parameter tuning.

How does the noise level affect the performance?

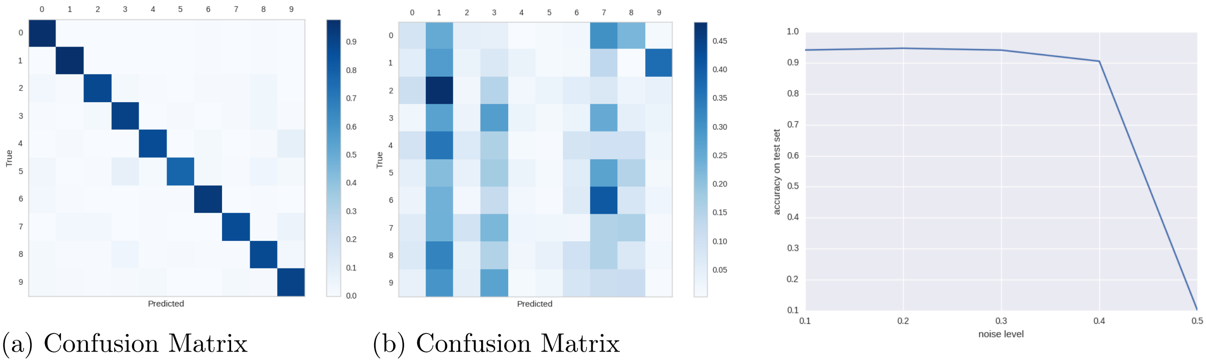
We tested the performance of BM classifiers by adding an increasing level of noise to the input data. Since the input samples were binarized, the highest level of noise is reached by randomly flipping 50% of the pixels in an image.
Results show that the classifiers predict with a higher accuracy when the noise level is set to 10%. Higher levels up to 40% perform similarly good as the confusion matrix below discloses (Figure 10a). However, at 50% noise level, digits are not longer legible and as expected, the classifier predictions are not better than guessing (accuracy of 10%). It is surprising, however, that with a 40% noise level all BMs are able to classify digits quite accurately as opposed to what is revealed to the naked eye. At a closer look to, for instance, the mean firing rates (Figures 9f-j), reveals that the classifiers are still able to learn the necessary features from the data.



(a) Input sample (noise = 0.1) (b) Input sample (noise = 0.2) (c) Input sample (noise = 0.3) (d) Input sample (noise = 0.4) (e) Input sample (noise = 0.5)

(f) Mean firing rate (noise = 0.1) (g) Mean firing rate (noise = 0.2) (h) Mean firing rate (noise = 0.3) (i) Mean firing rate (noise = 0.4) (j) Mean firing rate (noise = 0.5)

Figure 9: First row: An input samples with different noise levels added. Second row: The mean firing rates of the classifier associated to the input sample (number 6) when trained with input data at different noise levels.



(a) Confusion Matrix (Noise = 0.4)

(b) Confusion Matrix (Noise = 0.5)

(c) Noise vs. Accuracy

Figure 10: BM classifiers performance on input data at different noise levels.

Is there a way to improve the current results?

We apply data augmentation in an attempt to improve the overall performance. We added extra samples to the set, by choosing 10% of the dataset randomly and applying different degrees of rotation. Results from a trial-run are collected in the figure below.
Generally, adding rotated samples to the set improved at most by 0.1% in accuracy. Rotations between 70° to 120° showed the worst results. By looking at the input samples one can guess that handwritten numbers 6 and 2 can be easily confused at this rotation angle, if one is rotated. Numbers 6 and 9 face the same issues.

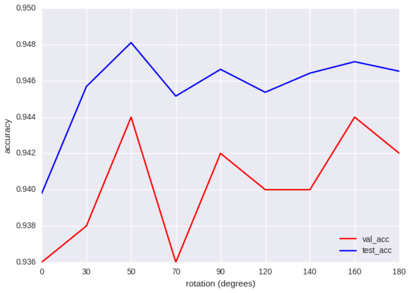


Figure 11: Test and Validation accuracy of sets of data augmented using different image rotations.