Boltzmann Machine

Kevin Jacobs, Zhuoran Liu, Ankur Ankan

Introduction to Boltzmann Machine

Boltzmann machine is a general learning rule applicable to any stochastic network with symmetric connections.
It was firstly introduced by Hinton and Sejnowski in 1983. The name Boltzmann machine is because the probability of the states of the system is given by the Boltzmann-Gibbs distribution of statistical mechanics.

Main learning process

The learning process will adjust the weights and biases in each iteration using these two formulas until convergence.
Every time we need to calculate $\langle S_i S_j \rangle$ and $\langle S_i \rangle$ in both clamped and un-clamped state for each pattern.
To calculate them, we must come the equilibrium and sample many units from its equilibrium distribution at a temperature.

Deterministic Boltzmann machine

In the original form, the Boltzmann learning algorithm is very slow because of the need for extensive stochastic variables, in particular the un-clamped $\langle S_i S_j \rangle$.

There are few applications compared with back-propagation, and the mean filed theory version is mainly considered.

Normalizing Constant

In the learning procedure, probabilities of state vectors are needed in order to compute the gradients. For these probabilities, a normalizing constant is needed. This requires to sum over all possible 2^n states where n is the number of units - which takes a very long time.

In the first part of the assignment, we approximated the normalizing constant as follows. Since all the probabilities should sum up to 1, we know that:

$$1 = \frac{1}{Z} \sum_{\mathbf{s}} \exp(-E(\mathbf{s}))$$

So therefore, $Z = \sum_{\mathbf{s}} \exp(-E(\mathbf{s}))$. The states that contribute the most to Z are the states with low energies.

Sequential Dynamics

With sequential dynamics, a random state is initialized. Then, one of the neurons is picked and is changed according to the Boltzmann-Gibbs distribution. This is repeated many times. The energy of the system will be minimized and therefore states can be sampled which have low energy. Now we can approximate Z be summing over these sampled states. Other states will contribute less to Z and will be neglected.

Mean Field Approximation

Using Mean Field Approximation, Z is approximated by calculated the free energy in the system and then calculate Z directly from it using $Z = \exp(-F)$. This is way faster, because we do not need to take samples from the Boltzmann-Gibbs distribution in order to train the Boltzmann Machine.

Conclusion

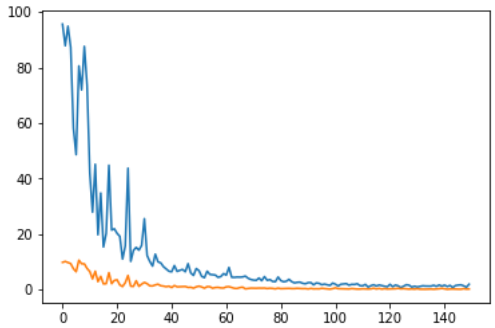
Use the Mean Field Approximation whenever this is possible to avoid the computational burden on computing the normalizing constant. Otherwise, take samples from the Boltzmann-Gibbs distribution and approximate the normalizing constant and if the number of units is extremely small, Z can be computed by bruteforcing all possible states.

Results
First part of the assignment

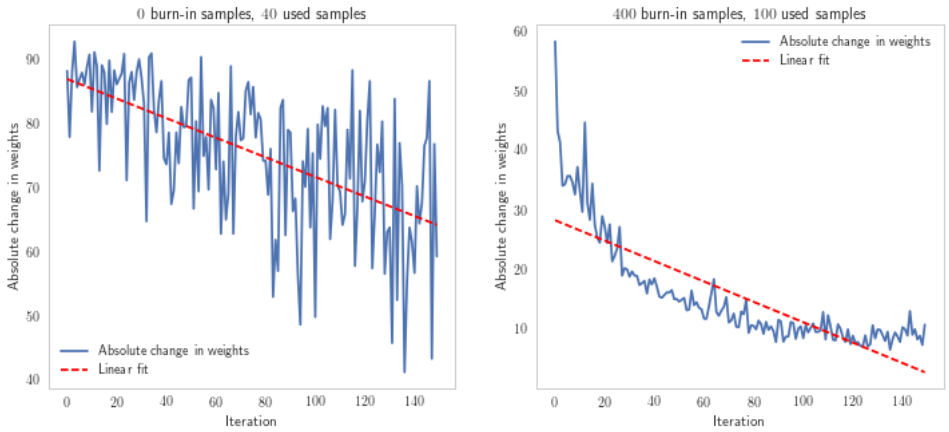
We were asked to generate training samples and use a Boltzmann Machine consisting of 10 neurons. In order to compute the normalizing constant, we used the sampling approach as we described. We use 500 steps of sequential dynamics, but we throw away some burn-in samples. In total, we use 100 burn-in samples and we use 400 samples generated in the low energy state.

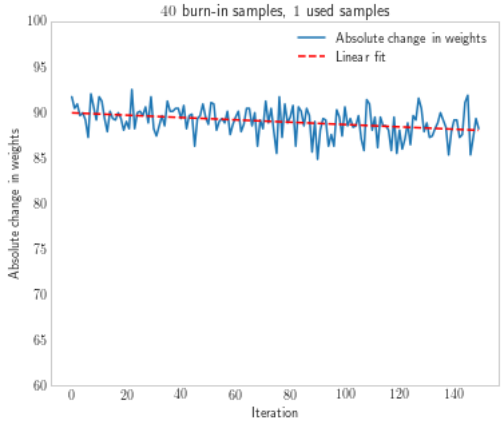
Convergence

Convergence for the first part of the assignment is visible in the next plot. The blue line is the absolute change in the weights and the yellow line shows the absolute change in the biases.



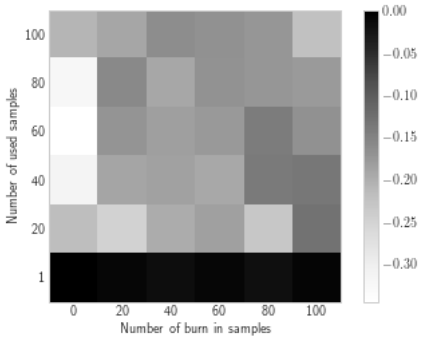
We tried to capture the convergence of the absolute change in weights using a linear model on the change in weights and by plotting the slopes for different configurations. We use input consisting of 10 components, 150 iterations of the learning algorithm and we take different numbers of burn-in samples and of samples that we actually use. The learning rate is fixed to 0.05. A linear model for the convergence is not realistic, but we hope to see some behaviour. The following plot shows two different configurations and a linear fit on the change in weights (and also shows that a linear model is not the best model since both pictures have an equivalent slope).





For the final report, we will look for other measures of convergence and then figure out which configurations work best.

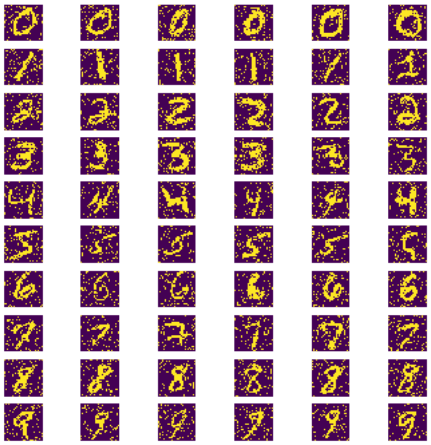
It gives however very low slopes for either fast or slow converging change of weights. If we now plot the slope of the linear fit per configuration, we get the following result:



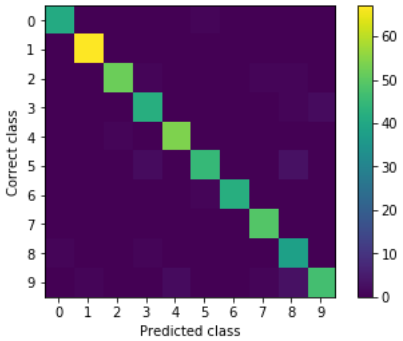
The only thing that can be seen, is that there is very little change in weights when the number of samples sampled from the Boltzmann-Gibbs distribution is extremely low (say 1). If we plot such a case, we can clearly see that this is the case:

Second part of the assignment

For the second part of the assignment, we use MNIST data consisting of 28×28 images. Some samples with 10% noise are shown in the following image:



The images are converted to binary images. We indeed found similar results as reported in the assignment description. The next image shows the confusion matrix. We got 23 errors out of 500 test images, so an overall accuracy of 95%.

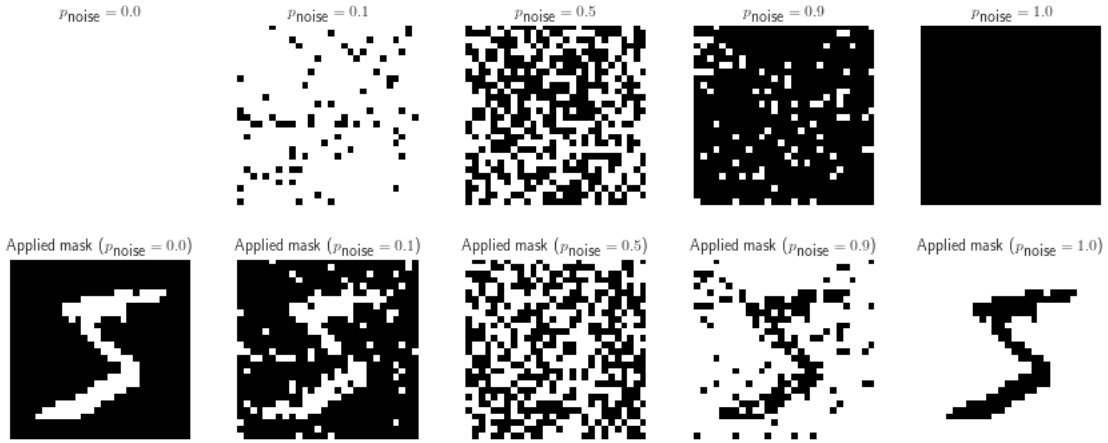


We wondered how different levels of noise would influence the accuracy. We ran 10 rounds for cross validation and we computed the accuracy for different noise levels. First, I will explain how we defined the noise and then I will talk about the results.

Definition of noise

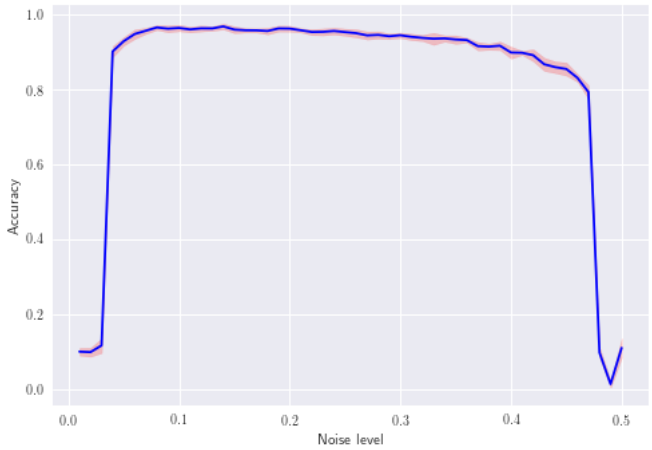
For each pixel, a noise mask is created. Every pixel in the mask has value -1 with probability p_{noise} and value 1 with probability $1 - p_{\text{noise}}$. Then, element-wise multiplication is applied on the original image with the mask.

For $p_{\text{noise}} = 0$, there is no distortion of the original images. For $p_{\text{noise}} = 1$, every bit is flipped and the structures in the original images remain. For $p_{\text{noise}} = 0.5$, maximum noise is obtained and there is no structure in the images.



Noise level versus accuracy

In the following plot, the results are shown. It can be seen that $p_{\text{noise}} \approx 0.10$ is optimal. Furthermore, there are some interesting spots in the plot.



Why is there a low accuracy for $0.00 \leq p_{\text{noise}} \leq 0.03$***?***

Because when there is a low noise level, there are pixels without any change. Take for example the left-most pixel in each image. This pixel is off in all the images. The learning algorithm cannot cope with constant pixels, so at least some noise should be added.

Then, there will be multiple zeroes on the diagonal of the C matrix used in the learning procedure. In the learning procedure, this matrix is inverted. In some cases it will not be possible to invert this matrix because of these zeroes. That is one explanation for the low accuracy between 0.00 and 0.03.

Dip at 0.48 *and accuracy at* 0.50

We could not find an explanation for the dip for $p_{\text{noise}} \approx 0.48$. The accuracy 0.10 for $p_{\text{noise}} = 0.50$ is explained by the fact that since the image is completely random. Since the image is completely random, the classifier acts the same as a random guess. By chance, guessing the correct number is indeed $\frac{1}{10}$.

Conclusion

Use the Mean Field Approximation whenever this is possible to avoid the computational burden on computing the normalizing constant. Otherwise, take samples from the Boltzmann-Gibbs distribution and approximate the normalizing constant and if the number of units is extremely small, Z can be computed by bruteforcing all possible states. For the MNIST dataset, a noise level of about 10% works best. This is the only parameter in the Boltzmann Machine model as described as in the assignment using the Mean Field approximation for the normalizing constant.