Boltzmann Machine learning

Niels Fennema, Luke Reijnen and Joeran Bosma

January 2020

1 Introduction and application

The dynamics of neurons in the brain are best described by the Hodgkin-Huxley model, describing the ion currents of the cell and (external) stimuli. However, simulating a large number of neurons using these dynamics is infeasible. In this report, the dynamics of the individual neurons are neglected and the system of neurons is described as an Ising model. This statistical description of the system is motivated by the stochastic nature of neurons. The Ising model means that the neurons are reduced to a binary state of either 'firing' or 'not firing' and all neurons are fully connected. When the couplings are symmetric, this reduces to the Boltzmann Machine.

We will apply a Boltzmann Machine to describe the neuronal activity of retinal neurons of a Salamander. The neuronal activity was measured as part of a study published in Nature [1]. As a test of the Boltzmann Machine, we will generate new data and compare the statistics of this newly generated data with the statistics of the measured data. In particular, the firing rates of particular patterns will be compared.

2 Model

In order to mimic the statistics of the Salamander's neurons, the couplings matrix w_{ij} and thresholds h_i of the Boltzmann Machine are adapted such that the likelihood of observing the measured action potentials is maximized. Due to the monotonic increasing property of the logarithm, this is equivalent to maximizing the log-likelihood of observing the data:

$$L(w,\theta) = \frac{1}{P}\sum_{\mu} \log p(s_1^\mu, ..., s_n^\mu) \tag{1}$$

The factor P^{-1} is a normalization constant which does not alter the properties, but enables easier comparison of the log-likelihood for different data set sizes. Analogous to the Ising model, the energy and probability of observing a state are given by:

$$\begin{aligned} -E(s) &= \frac{1}{2}\sum_{ij} w_{ij} s_i s_j + \sum_i \theta_i s_i \\ Z &= \sum_s \exp\left(-E(s)\right) \\ p(s) &= \frac{1}{Z}\exp\left(-E(s)\right) = \frac{1}{Z}\exp\left(\frac{1}{2}\sum_{ij} w_{ij} s_i s_j + \sum_i \theta_i s_i\right) \end{aligned} \tag{2}$$

In this notation $s = \{s_1, s_2, ..., s_N\}$ denotes a 'state' or 'pattern', with s_i denoting the state of a particular neuron ($s_i = 1$ meaning an action potential and $s_i = 0$ meaning no action potential). For the Boltzmann Machine, the couplings are symmetric ($w_{ij} = w_{ji}$), which means that the Boltzmann-Gibbs distribution is the stationary distribution of the Glauber dynamics.

To maximize the log-likelihood w.r.t. the model parameters, one can calculate the partial derivatives:

$$\begin{aligned} \frac{\partial L}{\partial \theta_i} &= \langle s_i \rangle_c - \langle s_i \rangle \\ \frac{\partial L}{\partial w_{ij}} &= \langle s_i s_j \rangle_c - \langle s_i s_j \rangle, \; i \neq j \end{aligned} \tag{3}$$

With $\langle s_i \rangle$ the mean firing rate ('expectation value of the spin') and $\langle s_i s_j \rangle$ the mean spin product. The subscript c denotes the statistics observed in the data and without subscript are the 'free' statistics of the Boltzmann Machine. See (17) and (18) for the formulae of the clamped statistics. The derivation of these partial derivatives can be found in Appendix C.

The difference in these statistics dictate the way the couplings and thresholds should be updated:

$$\begin{aligned} \theta_i(t+1) &= \theta_i(t) + \eta \frac{\partial L}{\partial \theta_i} = \theta_i(t) + \eta \left(\langle s_i \rangle_c - \langle s_i \rangle \right) \\ w_{ij}(t+1) &= w_{ij}(t) + \eta \frac{\partial L}{\partial w_{ij}} = w_{ij}(t) + \eta \left(\langle s_i s_j \rangle_c - \langle s_i s_j \rangle \right), \; i \neq j \end{aligned} \tag{4}$$

In which η is the learning rate. These relations are called the Boltzmann Machine learning rules.

3 Exact learning

Exact calculation of the statistics of the model entails three sums over all states. The normalization constant Z is required to determine $p(s)$, after which $\langle s_i \rangle = \sum_s p(s) s_i$ and $\langle s_i s_j \rangle = \sum_s p(s) s_i s_j$ can be calculated. For small systems (up to about 20 spins) this calculation is possible, but for larger systems this becomes intractable, as the number of states increases exponentially (the number of states is 2^N).

To test the performance of the Boltzmann Machine and its learning rules we have generated a data set for 15 neurons. We generated 1000 bins with either a spike or no spike using Poisson deviates with a mean firing rate $\lambda = 0.1$ to resemble a typical data set. We initialized the coupling matrix with random uniform values between -0.1 and 0.2 and filled the thresholds with uniform random values between -0.1 and 0. We then symmetrized the couplings and set w_{ii} to zero.

After 50 learning steps with $\eta = 0.1$ the relative change in log-likelihood reduced to about $-3 \cdot 10^{-3}$%, which further reduced to about $-6 \cdot 10^{-4}$% after 100 steps. The absolute changes in model parameters after 50 learning steps were all below $3 \cdot 10^{-3}$, and below $2 \cdot 10^{-3}$ after 100 steps. Because the likelihood is difficult to calculate accurately for larger systems, we will use the change in model parameters as convergence criterion, with the above value after 100 steps as threshold:

$$\max(\Delta w_{ij}, \Delta \theta_i) \leq 2 \cdot 10^{-3} \tag{5}$$

The value of $\eta = 0.1$ was determined empirically. A value of 0.2 resulted in oscillating convergence and with $\eta = 0.5$ the system did not converge at all. The training metrics for the toy data set are included in Figure 1.

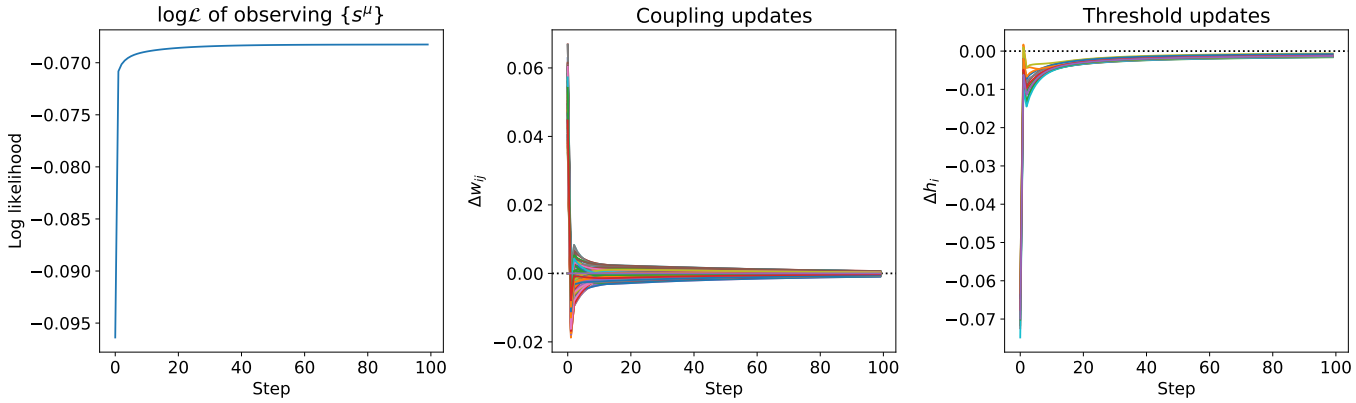


Figure 1: Log-likelihood and parameter updates for couplings w_{ij} and thresholds θ_i for a system of 15 neurons, with $\eta = 0.1$ and using exact calculations of the free statistics.

4 Approximating statistics

Our implementation in Python for a system of 15 neurons required about 30 minutes to perform 100 Boltzmann Machine learning steps. Disregarding memory constraints, overhead and possible optimizations, this would mean that a system of 20 neurons would require 16 hours of calculations. Calculating the statistics of the 160 neurons of the Salamander retina would take about 10^{37} years for just one step, about 10^{27} times the age of the universe.

Although optimizations could speed up these calculations, the sum over $2^{160} \approx 10^{48}$ states is simply intractable. Hence, an approximation of the free model statistics is required. We have employed both a vanilla/flat importance Monte Carlo sampler and a Metropolis-Hastings Monte Carlo Markov Chain to approximate the mean firing rates $\langle s_i \rangle$ and mean spin products $\langle s_i s_j \rangle$ for a given w_{ij} and θ_i.

4.1 Monte Carlo sampler

For large systems the target probability distribution $p(s)$ cannot be evaluated, however, $p^*(s) \equiv Z \cdot p(s) = \exp(-E(s))$ is easily calculated for larger systems as well. As seen in Exercise 29.13 from [2], importance sampling is a way to approximate the expectation value of a function of a stochastic variable:

$$\begin{aligned} \mathbb{E}[f(s)]_p &= \sum_s f(s)p(s) \approx \frac{Z_q}{Z} \cdot \frac{1}{P} \sum_i f(X_i) w^*(X_i), \text{ with } X_i \sim q(s) \\ \text{and } w^*(s) &= \frac{p^*(s)}{q^*(s)} = \frac{Z \cdot p(s)}{Z_q \cdot q(s)} \end{aligned} \tag{6}$$

P is the number of samples. With a flat proposal distribution $q^*(s) = 2^{-N}$ we have $Z_q = 1$ and Z is approximated by:

$$Z \approx \frac{1}{P} \sum_i w^*(X_i) = \frac{2^N}{P} \sum_i p^*(X_i), \text{ with } X_i \sim q(s) \tag{7}$$

Combining (6) and (7) gives:

$$\begin{aligned} \mathbb{E}[f(x)]_p &\approx \frac{P}{2^N \sum_i p^*(X_i)} \frac{1}{P} \sum_i f(X_i) p^*(X_i) 2^N \\ &= \frac{1}{\sum_i p^*(X_i)} \sum_i f(X_i) p^*(X_i), \text{ with } X_i \sim q(s) \end{aligned} \tag{8}$$

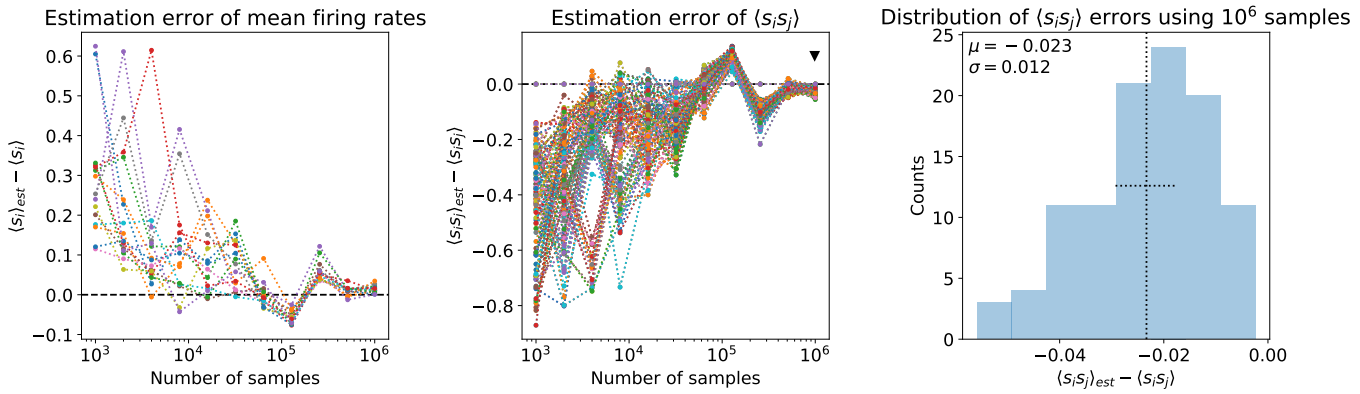


Figure 2: Estimation error of statistics $\langle s_i \rangle$ and $\langle s_i s_j \rangle$ using the vanilla Monte Carlo sampler. The distributions of errors are summarized to their respective mean and standard deviation, as shown in the right panel for the estimation of $\langle s_i s_j \rangle$ using 10^6 samples.

4.2 Metropolis-Hasting Monte Carlo Markov Chain sampler

From *Bayesian Inference for Perceptron Learning with MCMC*[3] we know that when samples are distributed according to $p(s) = \frac{1}{Z}\exp(-E(s))$, we can employ Metropolis-Hasting MCMC as long as we can evaluate $E(s)$. The samples generated by this method are (asymptotically) distributed according to $p(s)$, so calculating the relevant quantities is simply done by:

$$\mathbb{E}[f(x)] \approx \frac{1}{P}\sum_i f(X_i) \tag{9}$$

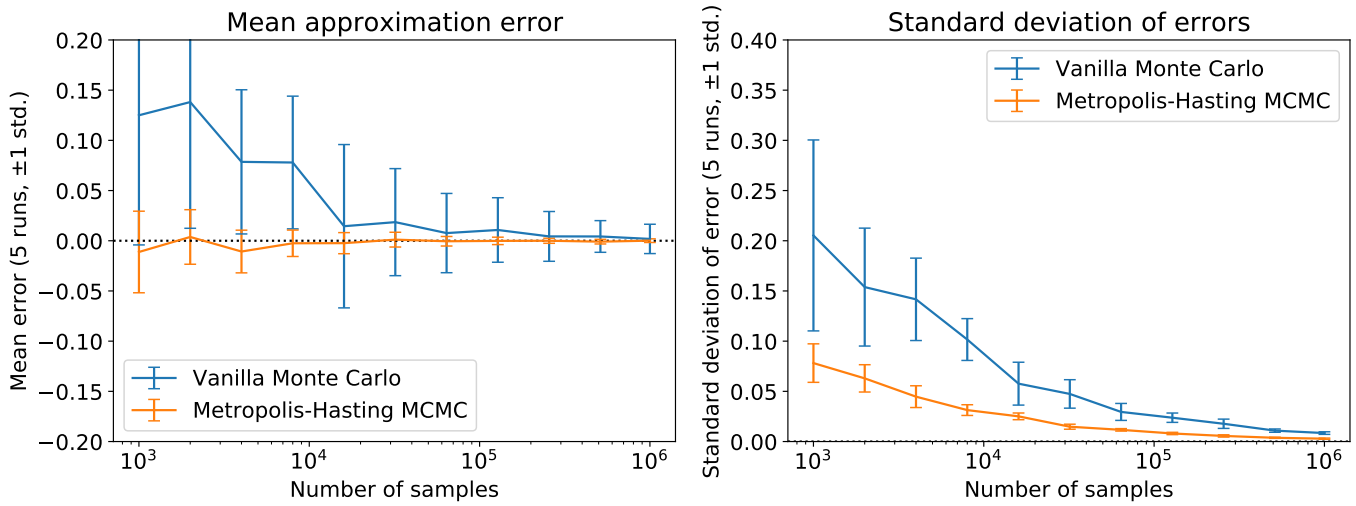
With P the number of samples.

The (state-dependent) proposal distribution can greatly impact the performance of the Metropolis-Hasting MCMC sampler, although asymptotically all distributions would yield the correct result. The proposal distribution is required to be constant in time, though. We have employed a proposal distribution that flips a set amount of spins of the current state and proposes the resulting state. This approach is motivated by the Simulated Annealing algorithm applied to a spin glass system[4], which also consists of a (fully) connected network of spins.

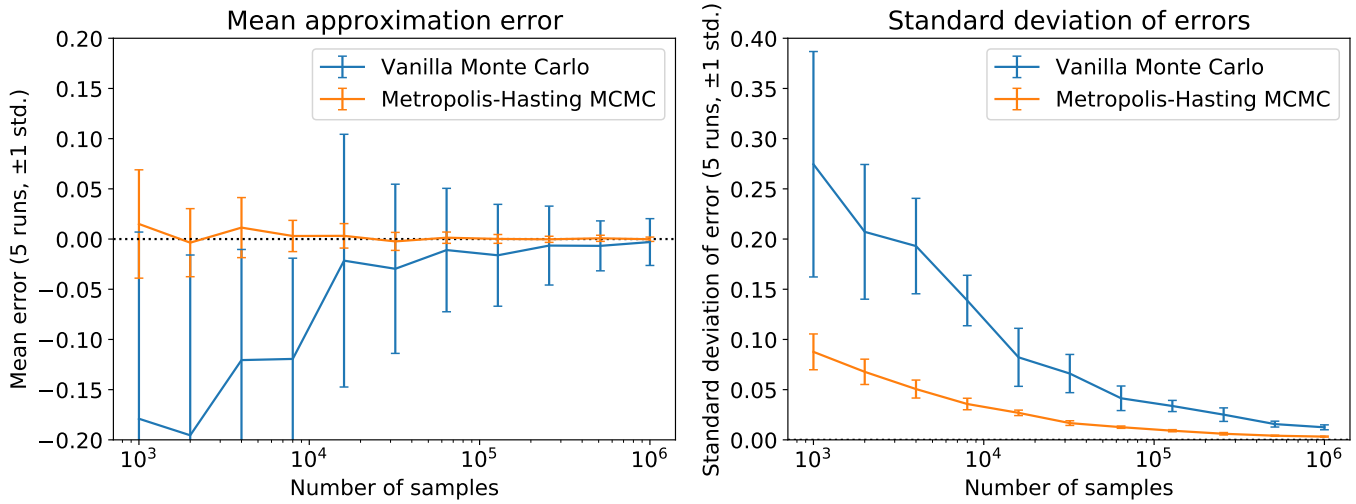
4.3 Performance of approximations

To assess the performance of these approximations we have compared the statistics obtained from these sampling methods with the exactly calculated statistics for a system of 15 neurons. To obtain typical results, we have used the couplings and thresholds obtained after 100 exact learning steps as described in *3 Exact learning* to calculate the energy of a state.

Figure 2 shows the estimation errors of the mean firing rates, $\langle s_i \rangle_{est} - \langle s_i \rangle$, and of the mean spin products, $\langle s_i s_j \rangle_{est} - \langle s_i s_j \rangle$, using the vanilla/flat importance Monte Carlo sampler. For a system of 15 neurons, the mean firing rate errors contain $N = 15$ values and the mean spin product errors have $(N^2 - N)/2 = 105$ unique values, which are displayed on Figure 2 left and center panel, respectively. The distribution of the mean spin product errors when using 10^6 samples is highlighted in the right panel of Figure 2 and shows how the distribution is summarized to its mean and width (with one standard deviation as measure).
The same statistics were estimated using the Metropolis-Hasting Monte Carlo Markov Chain sampler using the same number of samples (including the 5% of samples which were burned). The estimation



(a) Bias and standard deviation of mean firing rate errors, $\langle s_i \rangle_{est} - \langle s_i \rangle$



(b) Bias and standard deviation of mean spin product errors, $\langle s_i s_j \rangle_{est} - \langle s_i s_j \rangle$

Figure 3: Summarizing the approximation errors as in Figure 2 right panel was done eleven times. For each number of samples, the mean of these runs is shown. The error bars show the standard deviation between these eleven trails.

errors of these statistics have also been summarized to their means and standard deviations. For both the vanilla Monte Carlo sampler and Metropolis-Hasting MCMC, we performed the estimations eleven times to also capture the variance in these summarizing quantities. The biases and widths of the error distributions are shown in Figure 3a and 3b, with the error bars showing their variability between trails. These results clearly indicate that the Metropolis-Hasting MCMC method results in much better approximations of the statistics, for the mean of the error distributions are closer to zero, and the errors deviate less from this near-zero mean. Furthermore, these results clearly indicate that the vanilla Monte Carlo method overestimates the mean firing rates, while the mean spin products are underestimated. This effect is not visible for the Metropolis-Hasting MCMC method.

A similar analysis was performed to determine the optimal number of spins to flip for each new proposal state. An odd number of spin flips resulted in both lower bias and standard deviation of the approximation errors compared to an even number of spin flips. This can be explained by realising that depending on the initial state, an even number of spin flips cannot reach the ground state with all spins pointing

down (neurons not firing). Furthermore, increasing the number of odd (or even) spin flips resulted in an increase in both bias and standard deviation of the errors. Metropolis-Hasting MCMC with one spin flip for new proposal state thus has the best performance, as also shown in Figure A1.

To meet the convergence criterion set in *3 Exact learning*, the magnitude of the approximation errors should be at most $2 \cdot 10^{-3}/\eta$. With a value of $\eta = 0.1$, this means the approximations should be within $\pm 2 \cdot 10^{-2}$ of their true values. For the system of 15 neurons, the Metropolis-Hasting MCMC sampler using 10^6 samples yielded a deviation of more than 10^{-2} from the true value in less than 1% of the cases.

4.3.1 Larger system sizes

For larger systems the approximation error could increase, but this is hard to assess due to the exact statistics becoming intractable to compute. Figure A3 shows the performance of the Metropolis-Hasting MCMC sampler for system sizes up to 18 neurons, for couplings and thresholds obtained in a similar method as in *3 Exact learning*. This figure shows that the bias remains approximately zero and the width of the error distribution (precision) seems to increase linearly, as opposed to the exponential growth of the error of the Monte Carlo sampler (not shown).

For an unbiased estimator, the distribution of the estimation errors is equal to the distribution of the estimations across various trails, with the exception of the means of the distributions. Figure A4 shows the standard deviation of the estimation between various trails. The left panel of Figure A4 shows the mean firing rate estimates using 10^6 samples and the center panel shows the estimates using 10^7 samples. The error bars denote plus and minus one standard deviation from the mean estimate. The right panel of Figure A4 depicts the distribution of standard deviations of the $\langle s_i s_j \rangle$ estimates.

Based on Figure A4, the Metropolis-Hasting MCMC approximation using 10^7 should yield estimates which are accurate enough to meet the convergence criterion with $\eta = 0.1$. However, the convergence criterion is easier met by lowering the learning rate η. The convergence criterion is linear in η, whereas the estimation error is proportional to $1/\sqrt{\text{n.o. samples}}$. [1]

5 Mean field and linear response correction estimates

When the Boltzmann Machine only consists of one visible (fully connected) layer and no hidden layers, the weights and thresholds of the system can be estimated in closed-form. The learning rules, (3), maximize the log-likelihood of observing the data. At this maximum, these partial derivatives are zero, and the statistics coincide:

$$\begin{aligned} \langle s_i \rangle &= \langle s_i \rangle_c = m_i \\ \langle s_i s_j \rangle &= \langle s_i s_j \rangle_c, \; i \neq j \end{aligned} \tag{10}$$

[1] For large sample sizes M, the samples obtained from the Metropolis-Hasting MCMC are distributed $\sim p(s)$ and are independent when shuffled. When each contribution to an estimation is independent, the Fisher information is the sum of all individual contributions, i.e. $I_F \propto M$. For an unbiased estimator, $\sigma^2_{est} \geq 1/I_F$, so at best $\sigma_{est} \propto M^{-1/2}$. Figure A2 shows these variances and fits $\sigma = a \cdot M^{-1/2}$.

Substituting these quantities in the definition of the correlations between neurons gives the relation below which entirely depends on the data. Furthermore, the mean field approximation[5] provides equations for the mean firing rates of the neurons which have to hold self-consistently:

$$\begin{aligned} \chi_{ij} &= \langle s_i s_j \rangle_c - \langle s_i \rangle_c \langle s_j \rangle_c = \langle s_i s_j \rangle_c - m_i m_j, \; i \neq j \\ m_i &= \tanh\left(\sum_{j=1}^{N} w_{ij} m_j + \theta_i \right) \end{aligned} \tag{11}$$

This gives a relation for the thresholds:

$$\theta_i = \tanh^{-1}(m_i) - \sum_{j=1}^{N} w_{ij} m_j \tag{12}$$

The next part of the derivation uses the linear response correction. As derived in (21), the mean firing rate is related to the normalization constant Z as stated below. Taking another partial derivative to θ_l shows that this is equal to the correlation between neurons, of which the derivation entails the same expansion of Z and derivatives as in (21). Renaming the subscripts k and l to i and j gives:

$$\begin{aligned} \frac{\partial \log Z}{\partial \theta_i} &= \langle s_i \rangle \\ \frac{\partial^2 \log Z}{\partial \theta_i \partial \theta_j} &= \langle s_i s_j \rangle - \langle s_i \rangle \langle s_j \rangle = \chi_{ij} \end{aligned} \tag{13}$$

From this follows that, in the mean field and linear response correction estimates, the mean firing rate and correlations are related as:

$$\chi_{ij} = \frac{\partial \langle s_i \rangle}{\partial \theta_j} \approx \frac{\partial m_i}{\partial \theta_j} \tag{14}$$

Inverting the matrix of correlations, χ , gives:

$$\begin{aligned} \left(\chi^{-1}\right)_{ij} = \frac{\partial \theta_i}{\partial m_j} &= \frac{\partial \tanh^{-1}(m_i)}{\partial m_j} - \frac{\partial}{\partial m_j}\left(\sum_{k=1}^{N} w_{ik} m_k \right) \\ &= \frac{\delta_{ij}}{1 - m_i^2} - w_{ij} \end{aligned} \tag{15}$$

Where we substituted the relation for θ_i from (12) and used that $(\operatorname{arctanh} x)' = \frac{1}{1-x^2}$ for $|x| < 1$. We now have closed-form equations for the couplings and thresholds, just as equations (36) – (39) from [5]:

$$\begin{aligned} m_i &= \langle s_i \rangle_c \\ C_{ij} &= \langle s_i s_j \rangle_c - \langle s_i \rangle_c \langle s_j \rangle_c \\ w_{ij} &= \frac{\delta_{ij}}{1 - m_i^2} - \left(C^{-1}\right)_{ij} \\ \theta_i &= \tanh^{-1}(m_i) - \sum_{j=1}^{n} w_{ij} m_j \end{aligned} \tag{16}$$

6 Salamander retina neurons

We have applied the Boltzmann Machine learning to a data set containing action potentials of Salamander retina ganglion cells[6]. This data set contains recordings of 160 neurons during 297 repeats of a 19 seconds natural movie (almost 100 hours in total). The data is already preprocessed and contains binary bins of 20 ms denoting whether the neuron elicited at least one spike. As a result, the data set contains 283.041 bins for each neuron.

We have calculated the clamped mean firing rates and mean spin products, $\langle s_i \rangle_c$ and $\langle s_i s_j \rangle_c$, as defined in (17) and (18). We used a learning rate of $\eta = 0.01$ with 10^6 samples for the first 1000 Boltzmann Machine learning steps, followed by an additional 1000 steps with $\eta = 0.001$ and 10^7 samples to refine the couplings and thresholds. These smaller values of η were chosen because a learning rate of 0.1 with 10^7 samples did not converge to the clamped statistics very well. After 100 learning steps the parameter updates did not fall within the convergence criterion and the mean absolute percentage error (MAPE) of the mean spin products was 15%. In contrast, a learning rate of 0.01 with 10^6 samples converged much better, yielding a MAPE of 1.2% after 100 steps.

In order to perform these calculations fast enough, we rewrote our Metropolis-Hasting MCMC sampler in C++ and returned the relevant statistics to Python using Pybind11. This method enabled us to calculate the free $\langle s_i \rangle$ and $\langle s_i s_j \rangle$ statistics for one million samples in about 6 seconds (using four independent Markov chains of 250.000 samples). This scaled linearly to about one minute to calculate the statistics using 10^7 samples, divided into forty independent Markov chains of 250.000 samples for RAM efficiency. For all Markov chains we have used a burn-in of 5%, so the first 12.500 samples of each chain were not included when calculating the statistics. If more CPU cores are available, the number of concurrent Markov chains can be increased to speed up the estimations.

Using the refined couplings and thresholds, new data can be generated using the Metropolis-Hasting MCMC sampler. With this new data, predictions can be done. We have predicted the firing pattern rates of a subset of 10 of the Salamander retina neurons. These neurons produce a firing pattern of for example '0101000000' if neurons 2 and 4 elicit an action potential in a certain bin, but none of the other neurons. The bin size is 20 ms, so the firing rates are obtained as #occurrences/#samples $\cdot$ 50. Plotting the pattern rates predicted by the Boltzmann Machine versus the pattern rates observed in the Salamander retina recordings yielded the red dots in Figure 4.

Calculating the same pattern rates using the couplings and thresholds calculated with the mean field and linear response correction estimates gives the blue dots in Figure 4. As can be seen in the figure, many of the patterns are underestimated and overall performance is worse. However, calculating the couplings and thresholds using the closed-form is significantly faster than employing the Boltzmann Machine learning rules.

7 Discussion

Predictions made by the statistical model with couplings and thresholds inferred using the Boltzmann learning rules are significantly more accurate than predictions of the same statistical model with couplings and thresholds estimated by the mean field theory and linear response correction approximations. However, the computational difference between the two methods is of several orders of magnitude. Depending on the application, the most suitable solution may vary.

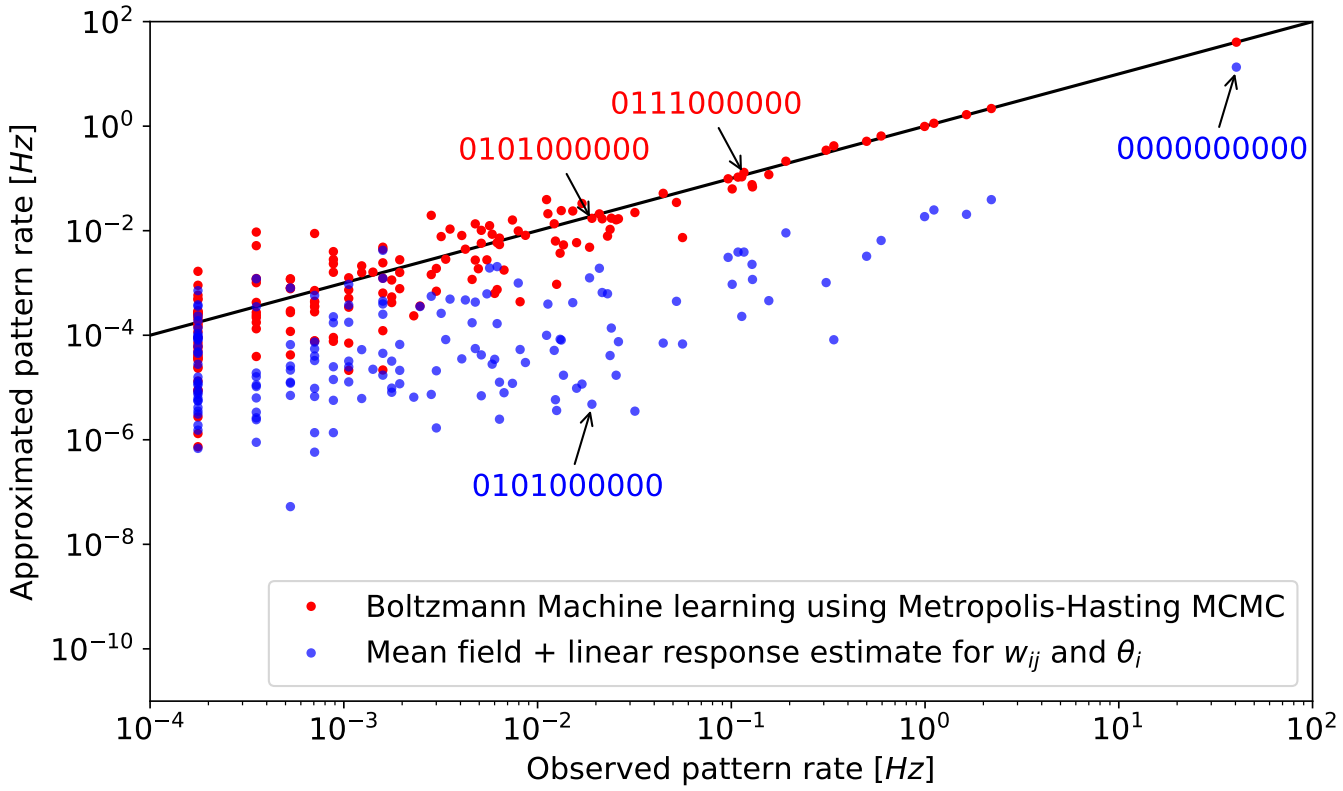


Figure 4: Estimations of observing certain patterns in a subset of 10 neurons of the Salamander retina data.

References

[1] Elad Schneidman, Michael J. Berry, Ronen Segev, and William Bialek. Weak pairwise correlations imply strongly correlated network states in a neural population. *Nature*, 440:1007–1012, April 2006.

[2] David MacKay. *Information Theory, Inference and Learning Algorithms*. March 2005.

[3] Joeran Bosma, Niels Fennema, and Luke Reijnen. Bayesian Inference for Perceptron Learning with MCMC. January 2020.

[4] Luke Reijnen, Niels Fennema, and Joeran Bosma. Iterative Improvement and Simulated Annealing for Energy Analysis of Spin Glass Ising Models. February 2020.

[5] Bert Kappen. Machine learning handouts. `http://www.snn.ru.nl/~bertk/machinelearning/handouts_ml.pdf`, December 2015. Accessed: 2020-01-18.

[6] O. Marre, G. Tkacik, D. Amodei, E. Schneidman, W. Bialek, and M. Berry. Multi-electrode array recording from salamander retinal ganglion cells. *IST Austria*, February 2017.

[7] Joeran Bosma. Code for Boltzmann Machine learning exercise. `https://drive.google.com/file/d/1srUk1JeYfvex55Xy2ublxfft8iZ1cZL2/view`, January 2020. Accessed: 2020-01-18.

Appendices

A Additional figures

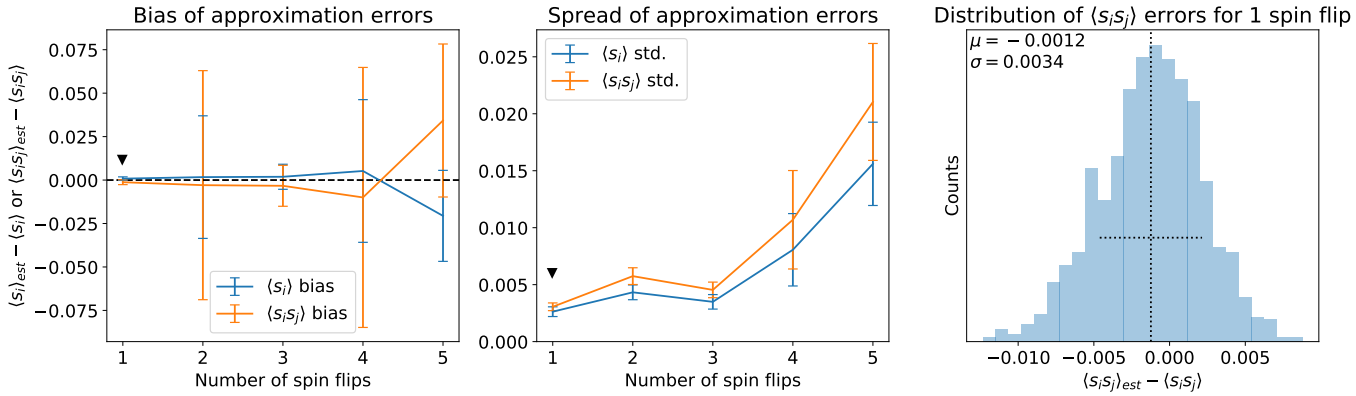


Figure A1: Accuracy and precision of statistics approximations using 10^6 samples, for different number of spin flips for the new proposal states of the Metropolis-Hasting MCMC. The right panel shows the distribution of $\langle s_i s_j \rangle$ estimation errors.

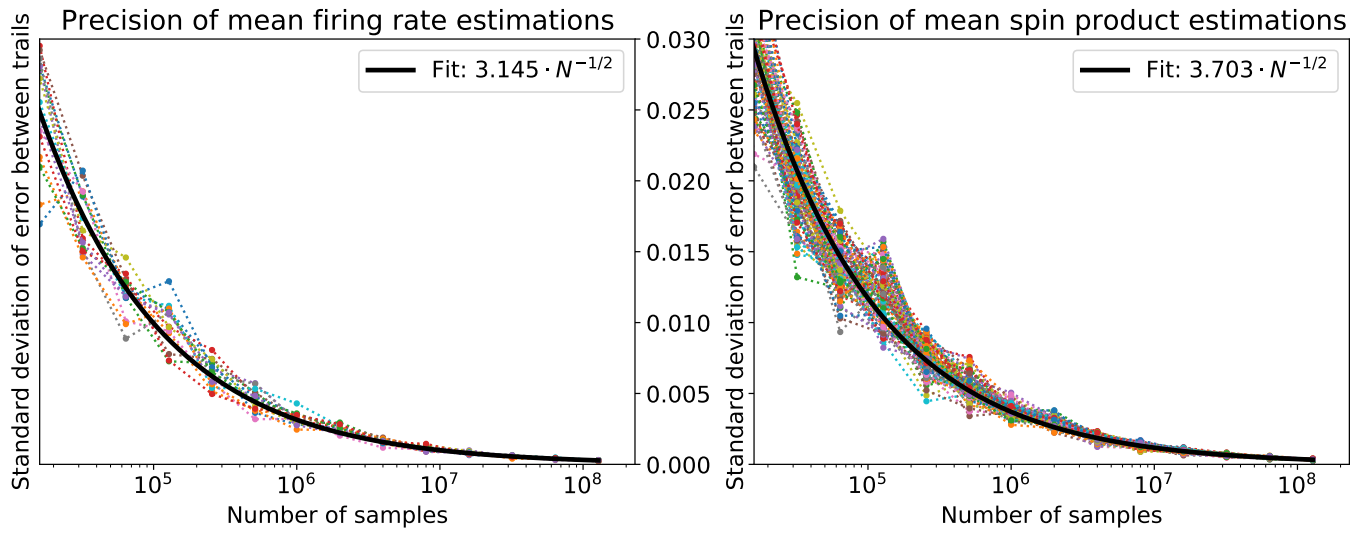


Figure A2: Precision of the mean firing rate estimations, $\langle s_i \rangle_{est}$, and mean spin product estimations, $\langle s_i s_j \rangle_{est}$. The precision is measured by the standard deviation of the estimation between the 30 trails. The precisions are averaged and fitted to the function $\sigma = a \cdot N^{-1/2}$, which is shown in black. The Pearson correlation coefficient between the predictions of this fit and the means of the precisions is $R = 0.9988$ and $R = 0.9980$ for the mean firing rate estimations and mean spin product estimations, respectively. The number of samples range from 16 thousand to 128 million, with a factor 2 between each step.

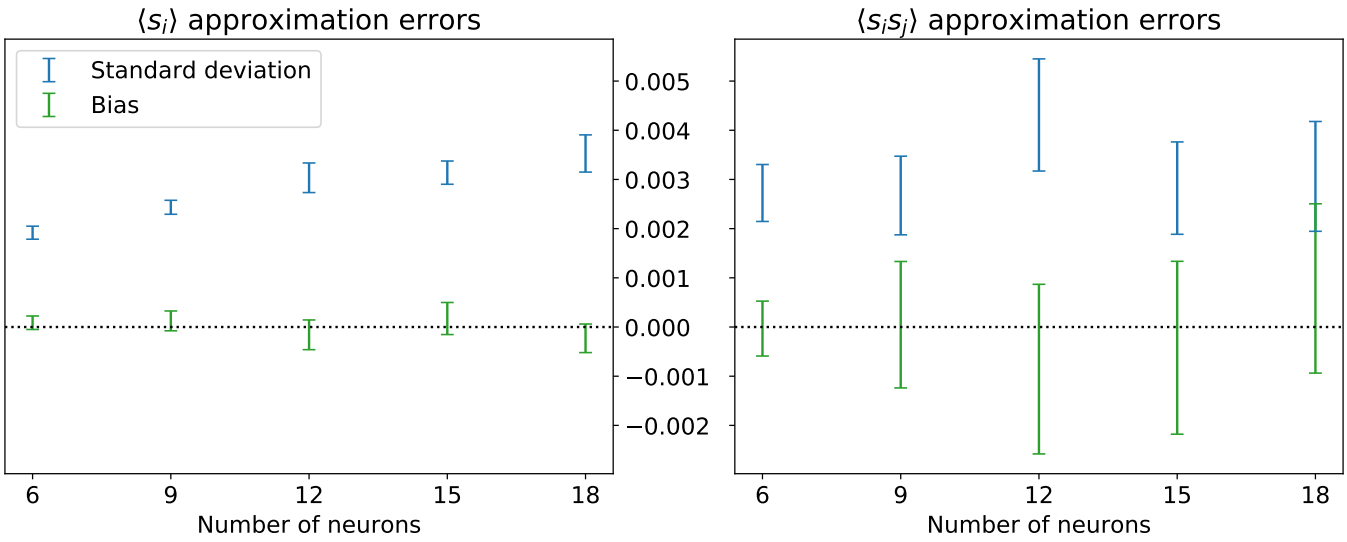


Figure A3: Performance of the approximations using 10^6 samples for different system sizes.

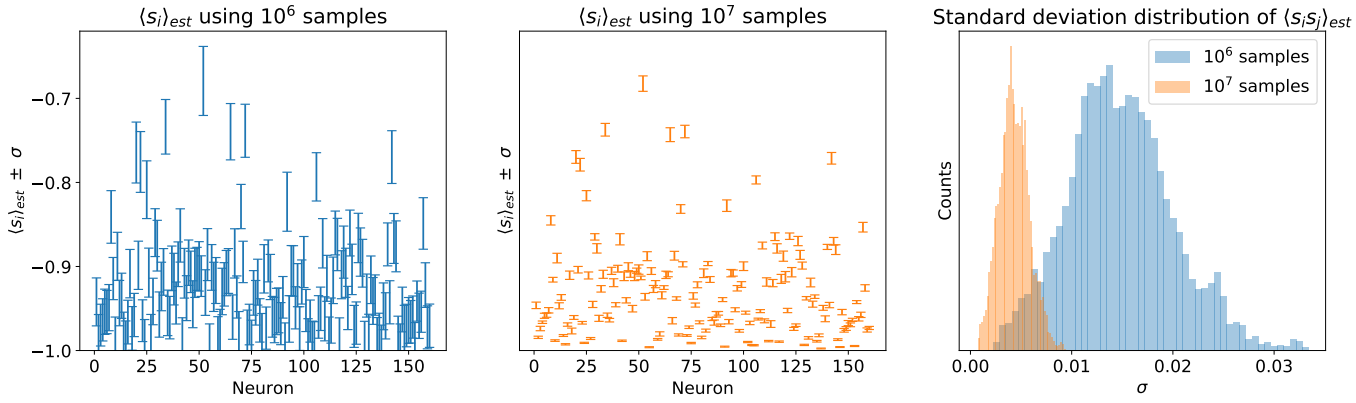


Figure A4: Mean firing rate estimates $\langle s_i \rangle_{est}$ and mean spin product estimates $\langle s_i s_j \rangle_{est}$. The left and center panel depict $\mu_{trails} \pm \sigma_{trails}$ for the Metropolis-Hasting MCMC sampler using 10^6 and 10^7 samples, respectively. The right panel shows the distribution of the standard deviation of the mean spin product estimates between trails. For these estimates, the couplings and thresholds obtained from the Boltzmann Machine learning on the Salamander retina data were used.

B Clamped statistics

The clamped mean firing rates and clamped mean spin products are calculated as:

$$\langle s_i \rangle_c = \frac{1}{P} \sum_{\mu} s_i^{\mu} \tag{17}$$

$$\langle s_i s_j \rangle_c = \frac{1}{P} \sum_{\mu} s_i^{\mu} s_j^{\mu} \tag{18}$$

Where μ runs from 1 to P, with P the total number of patterns in the data set.

C Derivation of learning rules

We start with the log-likelihood of observing the data:

$$\begin{aligned} L(w,\theta) &= \frac{1}{P}\sum_{\mu}\log p(s_1^{\mu},...,s_n^{\mu}) = \frac{1}{P}\sum_{\mu}\log p(s^{\mu}) = \frac{1}{P}\sum_{\mu}\log\left(\frac{1}{Z}\exp(-E(s^{\mu})\right) \\ &= -\frac{1}{P}\sum_{\mu}\log(Z) - \frac{1}{P}\sum_{\mu}E(s^{\mu}) = -\log(Z) - \frac{1}{P}\sum_{\mu}E(s^{\mu}) \end{aligned} \tag{19}$$

In which we used that the normalization constant Z is independent of the data. The partial derivative with respect to a threshold θ_k is:

$$\frac{\partial L(w,\theta)}{\partial\theta_k} = -\frac{\partial\log(Z)}{\partial\theta_k} - \frac{1}{P}\sum_{\mu}\frac{\partial E(s^{\mu})}{\partial\theta_k} \tag{20}$$

All parameters w_{ij} and θ_i are independent, so $\frac{\partial}{\partial\theta_k}(w_{ij}) = 0$ and $\frac{\partial}{\partial\theta_k}(\theta_i) = \delta_{ik}$:

$$\begin{aligned} \frac{\partial\log(Z)}{\partial\theta_k} &= \frac{1}{Z}\frac{\partial Z}{\partial\theta_k} = \frac{1}{Z}\frac{\partial}{\partial\theta_k}\left(\sum_s\exp(-E(s)\right) \\ &= \frac{1}{Z}\sum_s\exp(-E(s))\frac{\partial}{\partial\theta_k}\left(\frac{1}{2}\sum_{ij}w_{ij}s_is_j + \sum_i\theta_is_i\right) \\ &= \sum_s p(s)s_k \equiv \langle s_k\rangle \end{aligned} \tag{21}$$

In which we used $p(s) = \exp(-E(s))/Z$ and the definition of $\langle s_k\rangle$. Similarly:

$$-\frac{1}{P}\sum_{\mu}\frac{\partial E(s^{\mu})}{\partial\theta_k} = \frac{1}{P}\sum_{\mu}\frac{\partial\left(\frac{1}{2}\sum_{ij}w_{ij}s_i^{\mu}s_j^{\mu} + \sum_i\theta_is_i^{\mu}\right)}{\partial\theta_k} = \frac{1}{P}\sum_{\mu}s_k^{\mu} \equiv \langle s_k\rangle_c \tag{22}$$

Combining the results of (21) and (22) with (20), we obtain:

$$\frac{\partial L(w,\theta)}{\partial\theta_k} = \langle s_k\rangle_c - \langle s_k\rangle \tag{23}$$

The derivation of the second learning rule can be done completely analogous. The only difference is that

$$\frac{\partial}{\partial w_{kl}}\left(\frac{1}{2}\sum_{ij}w_{ij}s_is_j + \sum_i\theta_is_i\right) = s_ks_l$$

should be used in the derivations. This results in:

$$\frac{\partial L(w,\theta)}{\partial w_{kl}} = \langle s_ks_l\rangle_c - \langle s_ks_l\rangle,\ k\neq l \tag{24}$$

The relation is not valid for $k = l$, because the coupling of the neurons with itself is zero.

D Code

The code was written in Python and C++. The Jupyter Notebook with all Python code is freely available at [7] and can be run within Google Colaboratory (either in Playground mode or by copying to a personal Drive).