Simulated Annealing for the Ising perceptron
Markov Chains and Algorithmic Applications project

Nicolas Roussel
EPFL

December 18, 2017

Abstract

In this report, we formulate the Ising perceptron in a learning problem and use a Markov Chain Monte-Carlo (MCMC) method to solve it. In particular, the simulated annealing algorithm will be used to avoid local minimums and improve the convergence speed. This report will recall the problem statement and describe several simulated annealing schemes with their respective results.

1 Problem statement

Let $\underline{w}^*$ be a random vector of size N where every component is in {-1, +1} (i.i.d. uniformly distributed). This vector is the goal weights to achieve by our Ising perceptron. We then generate M vectors of size N which we call $\underline{x}_\mu$ for $\mu = 1, 2, .., M$ such that $\underline{x}_\mu \in \mathbb{R}^N$. These are the patterns of the dataset. Finally, from these patterns and weights we generate y_μ where

$$y_\mu = sign(\underline{w}^* \cdot \underline{x}_\mu) \ , \ \mu = 1, 2, .., M$$

This is a vector of size M where each of its components is respectively the class assigned to $\underline{x}_\mu$.

The goal of the problem is to find $\underline{w}^*$ when only given the pattern and classes pairs ($\underline{x}_\mu$, y_μ). Note that we define M as $\alpha = M/N$, and are interested in the different values α can take for large M, N.

2 MCMC solution

The Markov Chain Monte-Carlo solution to this problem is quite straightforward. The core idea is to start from a random weights vector $\underline{w}$ and swap its components until we reach a solution to the following equation:

$$y_\mu = sign(\underline{w} \cdot \underline{x}_\mu) \ , \ \mu = 1, 2, .., M$$

Here is the actual algorithm step by step:

1. Start from a random initial vector $\underline{w} \in \{-1, +1\}^N$
2. Compute the current energy of this weights vector as $E(\underline{w}) = \frac{1}{2} \sum_{\mu=1}^{M} (y_\mu - sign(\underline{w} \cdot \underline{x}_\mu))^2$
3. Select uniformly a component of $\underline{w}$ and swap its sign (from -1 to 1 or 1 to -1). Call this vector $\underline{w}'$.
4. Compute the energy of $\underline{w}'$ as in step 2.
5. Accept $\underline{w}'$ as the current weights vector $\underline{w}$ with probability

$$a_\beta(\underline{w}, \underline{w}') = min(1, exp(-\beta(E(\underline{w}') - E(\underline{w}))))$$

   β is called the temperature parameter (assume that it is a fixed scalar for now).
6. Iterate steps 2 through 5 until $E(\underline{w})$ is sufficiently low.

3 Simulated annealing

In the previous solution the β parameter is fixed. However, by changing this parameter as we iterate we might converge faster or even get a better result depending on how these changes are done. The intuition here is to start with a small temperature and increase it as we go, this will guarantee that when β is very high there will be no more changes accepted as according to the acceptance formula in step 5. There are obviously a lot of different temperature increasing schemes possible, we will from now on refer to them as temperature schedules, or just schedules. For example, we could simply linearly increase β at each iteration.

4 Results

4.1 First part: constant schedules

Constant schedules are exactly the algorithm described in chapter 2, the temperature β remains constant throughout the whole algorithm.

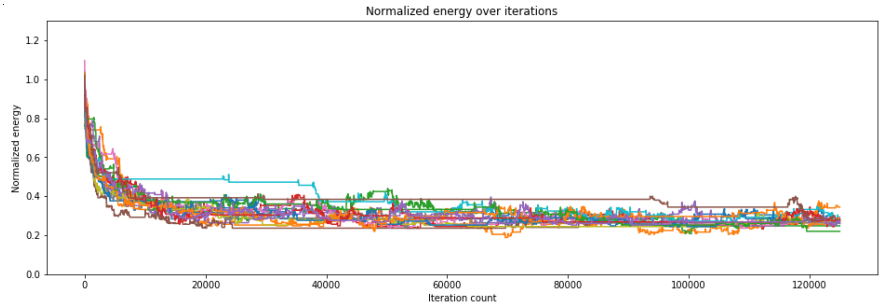

Figure 1: Normalized energy over iterations for $\beta = 0.6$, $\alpha = 2$ and $N = 250$. The colors are 16 independent runs of the algorithm using different random seeds.

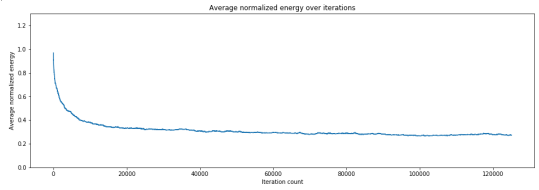

Figure 2: Averaged normalized energy over iterations for $\beta = 0.6$, $\alpha = 2$ and $N = 250$. Averaged over 16 independent runs of the algorithm using different random seeds.

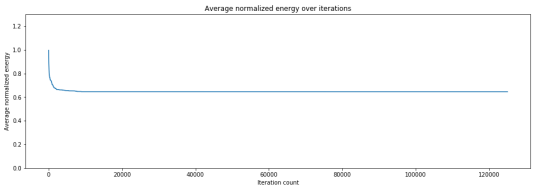

Figure 3: Averaged normalized energy over iterations for $\beta = 4$, $\alpha = 3.5$ and $N = 250$. Averaged over 16 independent runs of the algorithm using different random seeds.

Figure 1, 2 and 3 illustrate how the energy typically evolves over time for a constant schedule (other values of α and β show the same behavior). The main difference which can be observed when changing α and β is how fast the energy plateaus and at which value. Figure 2 and 3 illustrates this. When β is very small, the plots are much noisier and the normalized energy typically stays around 1. This happens due to the fact that the acceptance probability is always 1 or close to 1.

Another metric is the overlap $q(\alpha, \beta) = \frac{1}{N} \sum_{i=1}^{N} (w_i w_i^*)$. This indicates how close we are to the ground truth weights (our goal). For cases where α is small there might be several solutions to the equation in chapter 2, hence the energy can reach 0 but the overlap is not 1.

Figures 4 and 5 respectively show the averaged normalized energies and overlaps as function of α for different values of β. Interestingly enough we can see the performance completely switch order as α grows.

This can be explained by the previously mentioned case where the acceptance probability is very close to 1 - when both α and β are small. Additionally, as said before, when α is small there are multiple weight vectors which can reach a very low energy but might not have a good overlap. This is exactly what is illustrated in both figures.

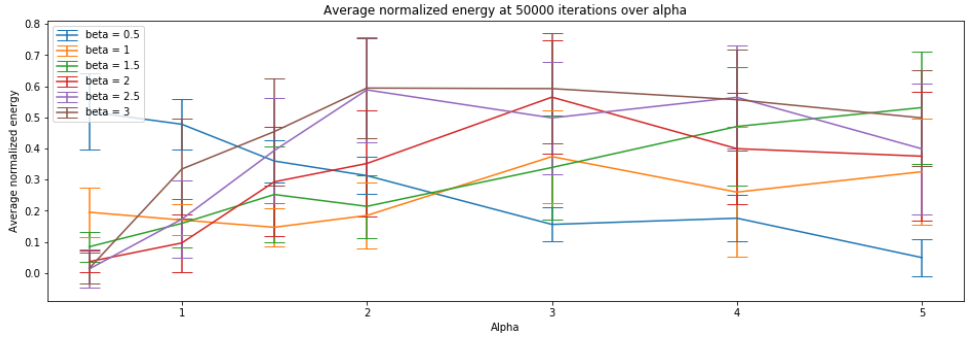

Figure 4: Averaged normalized energy at 50000 iterations over α for $N = 100$. Averaged over 128 independent runs of the algorithm. The horizontal caps represent the standard deviation of the measures.

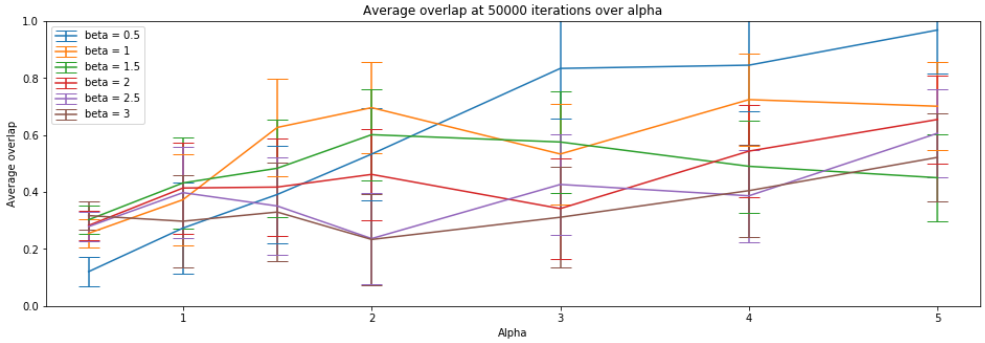

Figure 5: Averaged overlap at 50000 iterations over α for $N = 100$. Averaged over 128 independent runs of the algorithm. The horizontal caps represent the standard deviation of the measures.

4.2 Second part: linear, polynomial, logarithmic and exponential schedules

These are the schedules which were experimented with : linear, sublinear, polynomial, logarithmic and exponential. The Figure 6 illustrates each one of them and the Table 1 has their formal definitions. They were all optimized with a coarse grained grid search for $N = 100$, and $\alpha = 5$. Of course, there might be added value to optimizing these parameters for individual αs and Ns.

There was also an experiment to quantize these schedules. The main difference would be that the temperature would not change at every iteration, but only after a few iterations. There was no signifcant improvement to notice.

Figure 7 and 8 show the average normalized energies and overlaps of each schedule.

Constant	**Linear**	**Sublinear**	**Polynomial**	**Logarithmic**	**Exponential**
β_0	$\beta_0 + \gamma t$	$\beta_0 t^\gamma, \gamma < 1$	$\beta_0 t^\gamma, \gamma > 1$	$\beta_0 \log(t + \gamma), \gamma > 1$	$\beta_0 \gamma^{-t}, 0 < \gamma < 1$
$\beta_0 = 0.6$	$\beta_0 = 0.2$	$\beta_0 = 1.1$	$\beta_0 = 1.1$	$\beta_0 = 1.1$	$\beta_0 = 0.15$
	$\gamma = 0.7$	$\gamma = 0.8$	$\gamma = 1.5$	$\gamma = 01.2$	$\gamma = 0.1$

Table 1: Schedule definitions where t is the fraction of total iterations to complete.

5 Conclusion

There is no schedule which has a clear advantage over all others. However, slow-growing functions do have a better standard deviation which make them more consistent in practice. Differences might be more marked with larger values for N and better tuned parameters - more computation time would be needed.

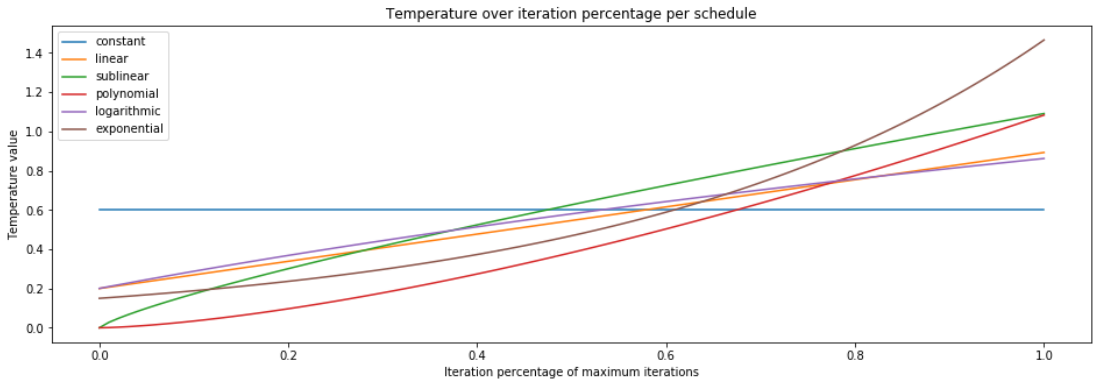

Figure 6: Temperature schedules

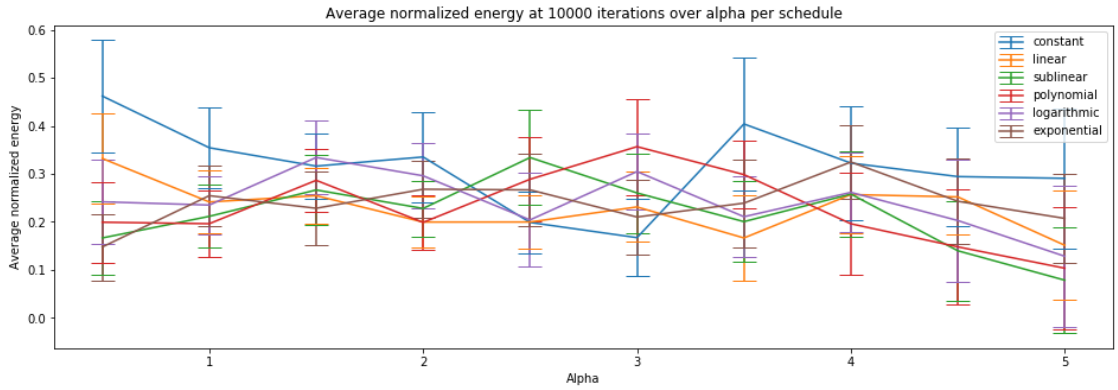

Figure 7: Average normalized energy at 10000 iterations over α per schedule ($N = 100$). Averaged over 128 independent runs of the algorithm. The horizontal caps represent the standard deviation of the measures.

/home/nicolas/Downloads/simulated-annealing-ising.pdf

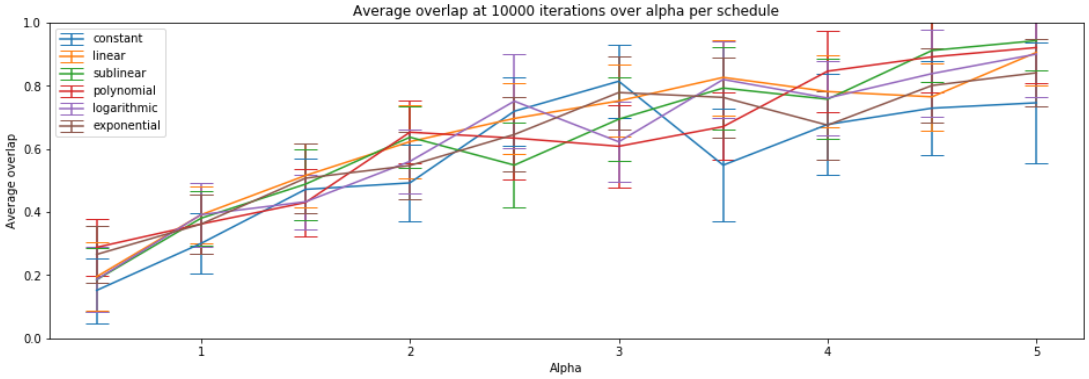

Figure 8: Average ovlerap at 10000 iterations over α per schedule ($N = 100$). Averaged over 128 independent runs of the algorithm. The horizontal caps represent the standard deviation of the measures.

6 Going further

There are a few more steps one can take to improve the performance of the final schedules found here. First of all, some finer tuning of the schedule parameters might give better results. Adding a constant offset to the temperature for certain schedules would avoid acceptance probabilities close to 1 in early stages of the algorithm. Investigating into other schemes might also be an interesting path to take, there might be parts of the schedule which can be more "aggressive" in their increases. Finally, using a Gibbs sampler instead of a Metropolis approach to this simulated annealing problem could also give some additional insights.