Advanced Machine Learning Week 3

Daria Mihalia, Benedetta Felici & Jasper Pieterse

November 2023

Exercise 31.1

a.)

$$S = \sum p(x) \ln \left[\frac{1}{p(x)} \right] = \sum p(x) \ln \left[\frac{1}{\frac{1}{Z(\beta)} \exp(-\beta E(x))} \right]$$

Simplify this expression:

$$\begin{aligned} S &= \sum p(x) \ln \left[Z(\beta) \exp(\beta E(x)) \right] \\ &= \sum p(x) \left[\ln Z(\beta) + \ln \exp(\beta E(x)) \right] \\ &= \sum p(x) \ln Z(\beta) + \sum p(x) \beta E(x) \end{aligned}$$

Since $\ln Z(\beta)$ is constant w.r.t x, we take it out of the sum:

$$S = \ln Z(\beta) \sum p(x) + \beta \sum p(x) E(x)$$

The sum of probabilities $\sum p(x)$ is equal to 1, and $\sum p(x)E(x)$ is the definition of the mean energy $\bar{E}(\beta)$. So we get:

$$S = \ln Z(\beta) + \beta \bar{E}(\beta)$$

b).

$$\begin{aligned}\frac{\partial F}{\partial T} &= \frac{\partial}{\partial T}\left(-kT \ln Z\right) \\ &= -\ln Z \cdot \frac{\partial}{\partial T}(kT) - kT \cdot \frac{\partial}{\partial T}(\ln Z) \\ &= -\ln Z \cdot k - kT \cdot \frac{\partial \beta}{\partial T}\frac{\partial}{\partial \beta}(\ln Z) \\ &= -\ln Z \cdot k + kT\frac{1}{kT^2}\frac{\partial}{\partial \beta}(\ln Z)\end{aligned}$$

The derivative of $\ln Z$ with respect to β can be expressed in terms of the mean energy:

$$\frac{\partial \ln Z}{\partial \beta} = \frac{1}{Z}\frac{\partial Z}{\partial \beta} = \frac{1}{Z}\sum_x -E(x)\exp(-\beta E(x)) = -\bar{E}$$

Substituting this back into our equation for $\frac{\partial F}{\partial T}$, we get:

$$\begin{aligned}\frac{\partial F}{\partial T} &= -\ln Z \cdot k - \frac{\bar{E}}{T} \\ &= -k[\ln Z + \beta \bar{E}]\end{aligned}$$

If we set $k = 1$, we recover our result of part a)

$$\frac{\partial F}{\partial T} = -S$$

Report: Exercices on Combinatoric Optimisation

We want to solve the following combinatorics problem:

$$E = -\frac{1}{2}\mathbf{x}^\top \mathbf{w}\mathbf{x} \tag{1}$$

with $\mathbf{w}$ an $n \times n$ symmetric matrix with zero diagonal and $\mathbf{x} = (x_1, \ldots, x_n)$ a binary vector: $x_i \in \{-1, 1\}$.

In this report, we will analyse and compare two different binary optimization methods for solving this problem: Iterative improvement and simulated annealing. Within simulated annealing, we will distinguish between the exponential schedule and the Aarts and Korst (AK) schedule.

Theory

Binary vs Nonbinary Optimization

For trivial systems where $\mathbf{x}$ is real-valued instead of binary $\|\mathbf{x}\|_2 = 1$, the minimum of E can be found by finding the eigenvector corresponding to minimal eigenvalue λ of $\mathbf{w}$. This procedure involves diagonalizing the symmetric matrix w. The computational complexity of diagonalizing an arbitrary binary $n \times n$ matrix is $\mathcal{O}(n^3)$ for algorithms such as Singular Value Decomposition making it computationally expensive for large matrices.

Solving the problem this way gives you a continuous valued eigenvector [see code]. Solving the problem with binary vectors x where each x_i is either $+1$ or -1 is a much more challenging problem.

For specific choices of $\mathbf{w}$, the problem can be significantly more or less difficult. In a ferromagnetic system where $w_{ij} > 0$, the energy is minimized when all spins are aligned. Since $w_{ij} > 0$ for all i, j, the product $x_i x_j$ contributes positively to the energy E. To minimize E, we therefore need to maximize $x_i x_j$ for each pair i, j. The product $x_i x_j$ takes its maximum value of 1 when x_i and x_j are either both 1 or both -1.

When w_{ij} has arbitrary sign, the system becomes frustrated as there is typically no global solution $\mathbf{x}$ that minimizes each term $w_{ij} x_i x_j$.

Iterative Improvement

To determine the minimum energy state of the system, we implement a Python algorithm for the iterative improvement method. It works very similar to a MH algorithm (see last week), but now the states are only accepted if the difference in energy

$$\Delta E = 2 \cdot x_i \cdot \sum_j w_{ij} \cdot x_j$$

is lower than one. If the energy difference is higher, we always reject. This is a local search algorithm and because the algorithm is only allowed to descend in energy, it gets stuck easily in local minima. The probability to still accept a higher energy state in MH sampling, allows it to better escape local minima.

Simulated Annealing

Simulated annealing involves starting at a high temperature $\beta = \frac{1}{T}$ in which the phase space can be explored broadly and then decreasing the temperature through some annealing scheme such that the sampler gradually 'freezes' in some local minima which hopefully is (close to) the global minima. The temperature will determine the degree to which the gradient descent search can be perturbed – the higher the temperature, the more energy and hence the higher perturbation. Thus, this allows to escape a local minima, making simulated annealing a very powerful algorithm.

The energy is updated differently, not by always accepting if $\langle \Delta E \rangle = 0$ but instead computing an acceptance rate in Metropolis-Hasting style but now with the temperature modulating the acceptance rates:

$$a = e^{-\Delta E \beta} \tag{2}$$

This acceptance rate can be used to update the states using the annealing schedule. A simple choice for such schedule is an exponential schedule $\beta_{k+1} = f\beta_k$. An alternative choice, proposed in *Aarts and Korst, 1988*, is $\beta_{k+1} = \beta_k + \frac{\Delta\beta}{\sqrt{V_k E}}$ with $V_k E$ the variance of the energy in chain k.

Implementation: Iterative Method

Methods

We implemented the iterative method in Python. The code for this method and all other results can be found here. For the iterative method, we take the neighborhood to be single spin flips and perform $N_{timesteps} = 10.000$ iterations of the algorithm. We use multiple restarts K using different initial states and take the lowest energy found from these restarts. This is because different initial states give rise to different descents through the energy landscape. We then use multiple runs $N_{runs} = 20$ of the algorithm to assess whether the results are reproducible.

To generate results for high K values, we implement the algorithm as efficient as possible. We used the Numba package to compile the code in machine code to speed up the process. Furthermore we only calculate the local field change of flipping the spin for efficient updates. We also experimented with an early stopping mechanism, but found that it was hard to determine where the threshold for it should be. The system tends to get stuck in local minima for a high amount of iterations before it escapes local minima.

We randomly generated 500×500 w symmetric matrix for both a ferro-magnetic and frustrated problem. The matrices were generated sparsely with $p = 0.5$, meaning that half of their values were set to 0. We evaluated the iterative improvement method on these two problems and the provided $n-500$ frustrated system matrix. We compared the performance of the iterative improvement method for these problems in terms of number of restarts K needed to obtain reproducible results. As a measure of performance, we evaluate the energy of the final solution averaged over the N runs and its standard deviation. We measured the average CPU time is for a single run of the algorithm (Over the K loop, not over the N runs).

Results

Ferromagnet For the ferromagnet problem we obtained these results:

K	CPU Time (sec)	E
20	0.12	-462 ± 0
100	0.21	-462 ± 0
200	0.31	-462 ± 0
500	0.88	-462 ± 0
1000	1.72	-462 ± 0
2000	3.45	-462 ± 0
4000	7.75	-462 ± 0

Random Frustrated For the randomly generated frustrated problem we obtained these results for $N = 10.000$ iterations of the algorithm:

K	CPU Time (sec)	E
20	0.03	-214 ± 3
100	0.16	-217 ± 0
200	0.34	-217 ± 0
500	0.89	-217 ± 0
1000	1.68	-217 ± 0
2000	3.42	-217 ± 0
4000	6.79	-217 ± 0

From the results for the ferromagnet and the random frustrated system, we can observe that for the increasing amount of restarts K, the CPU time also increases. More importantly, we see that there is no variance on the energy value E, thus it remains exact (except for the outlier in K = 20 for the random frustrated system).

W500 Frustrated For the frustrated problem $w500$ we obtained the following results for $N = 10.000$ iterations of the algorithm:

K	CPU Time (sec)	E
20	0.27	-6191 ± 70
100	1.34	-6279 ± 38
200	2.69	-6300 ± 51
500	6.63	-6340 ± 49
1000	13.13	-6358 ± 42
2000	26.29	-6388 ± 38
4000	52.60	-6408 ± 38

These are similar to the assignment, a little bit worse. Here, we can notice that for the W500 Frustrated system, the CPU time is significantly longer, still increasing with increasing number of K restarts. Also, the energy is decreasing with increasing restarts and the variance decreases, as expected – getting closer to exact values with more restarts.

To obtain similar CPU times as in the assignments, we tweak our amount of iterations to be $N = 3.500$:

K	CPU Time (sec)	E
20	0.11	-5873 ± 60
100	0.55	-6020 ± 97
200	1.00	-6032 ± 57
500	2.54	-6076 ± 46
1000	5.01	-6112 ± 38
2000	9.89	-6132 ± 38
4000	19.88	-6162 ± 52

Showing our implementation is not as efficient as in the assignment (might be because of MatLab vs Python; we tried optimizing the algorithm already). We also observe that although devreasing the number of iterations takes almost three times less time for computation, the variance of the energy is higher and the energy value for most restarts will still be higher than the previous run of this algorithm with N = 10000.

Sparsity also could explain why the random frustrated problem is so much faster than the $w500$ problem.

Implementation: Simulated Annealing

Methods

We optimized the algorithm using Numba and only computing the state energy once at the start of the computation and then only adding the energy change if the state was accepted. To properly track computation time, a warm-up was performed such that Numba and weights were intialized before sampling time was tracked.

The algorithm works by starting at an initial temperature β_1 which is determined by finding the maximum energy difference ΔE_{max} for sampling the initial state for 10.000 iterations at high temperature $\beta = 0.1$. The initial temperature is then given as $\beta_1 = \frac{1}{\Delta E}$. Every L iterations, we update the temperature $\beta_i \to \beta_{i+1}$ through an iterative update scheme.

For different parameters of the annealing schemes, we computed the lowest energy and the CPU time over 20 separate MCMC runs and then averaged these results to obtain the mean computation time and mean ground energy and their standard deviation.

It is unclear from the implementation in the assignment what kind of measure of convergence was used for the exponential schedule. We tried both the $\mathrm{Var}(E_L) < 0$ or using a fixed *accepted* sample size, but neither give similar CPU times as in the assignments. We resorted to having a fixed sample size, because that gave us our best results. For the AK schedule, we did not use a fixed amount of samples but instead sampled until $\mathrm{Var}(E_L) < 0$.

For the exponential schedule, we found worse results for $L = 1000$, while in the assignment the results are similar to the rest, only the computation time is tripled. It is not entirely clear to us what the cause of this discrepancy is, but is must have to do with the previously mentioned ambiguity in convergence measure.

Results

AK Schedule For the AK schedule, we found the following results using the variatonal bound:

$\Delta\beta$	L	CPU (sec)	E
0.1	500	1.28	-6436 ± 73
0.01	500	10.32	-6566 ± 26
0.001	500	15.14	-5790 ± 184
0.0001	1000	14.90	-5854 ± 167

Which appears to be somewhat similar to the results in the assignment. We observe that with increasing values of L, number of iterations or moves made at each temperature level, the CPU time increases which is to be expected. Also, the more iterations, the lower the energy E is, but also variance thereof increases.

Exponential Schedule For the exponential schedule, we found the following results:

For 1.000.000 total samples, no energy variance convergence

$\Delta\beta$	L	CPU Time (sec)	E
1.01	500.00	0.74	-6516 ± 68
1.001	500.00	0.74	-5827 ± 138
1.002	500.00	0.73	-6557 ± 38
1.0002	1000.00	0.74	-5877 ± 131

For 5.000.000 total samples, no energy variance convergence

$\Delta\beta$	L	CPU Time (sec)	E
1.01	500.00	5.54	-6523 ± 49
1.001	500.00	5.59	-6565 ± 25
1.002	500.00	6.05	-6563 ± 37
1.0002	1000.00	5.98	-5880 ± 154

In both these cases, the CPU time remains constant as we did not vary the number of samples - except for the last example, which doubles L, and, in the last table, leads to much higher energy but also significantly higher variance.

For $L = 1000$ we observe worse results then for $L = 500$. This is different from the results in the assignment. We again hypothesise that this difference is due to the difference in convergence criterion used. When using a fixed sample size N, the run using $L = 1000$ gets effectively half of the temperature updates that $L = 500$ gets. It appears that faster temperature updating is beneficial for the algorithm, rather then exploring phase space longer at fixed temperature.

Using no fixed sample size but instead the energy variance convergence measure we found the results:

$\Delta\beta$	L	CPU Time (sec)	E
1.01	500.00	0.38	-6078 ± 146
1.001	500.00	0.41	-6027 ± 119
1.002	500.00	0.37	-6018 ± 118
1.0002	1000.00	1.22	-6224 ± 97

So energy variance convergence appears to give worse results and we don't see the same CPU time scaling as in the assignment (although there is a tripling of computation time when going from $L = 500$ to $L = 1000$). It remains a mystery what convergence criterion was used in the assignment, but these results suggest it was not the energy variance.

Conclusion

This report compared two different methods for solving combinatoric optimization problems: iterative improvement and simulated annealing. We found that iterative improvement works well for simpler problems, like the ferromagnetic system, but it struggles with more complicated ones, like the frustrated system.

This is because it often gets stuck in local minima. We tried to fix this by using multiple restarts, which helped but also made the process take longer.

On the other hand, simulated annealing was more effective, especially for the complicated problems. The gradual temperature decrease helps to avoid getting stuck in local minima, allowing it to find better solutions. Sadly, we could not figure out the convergence method used, so our results differed slightly from the assignment but the overall gist remained the same.