CV202, HW #5

Oren Freifeld and Meitar Ronen
The Department of Computer Science, Ben-Gurion University

Release Date: 20/4/2020
Submission Deadline: 31/5/2020, 20:00

Abstract

Please make sure you have carefully read the general instructions in the course website at https://www.cs.bgu.ac.il/~cv202/HW_Instructions. These instructions address, among other things, what to do or not do with figures, Jupyter Notebooks, *etc.*, as well as what compression files are legit or not.

This assignment is dedicated to building an exact sampler for the Ising model, on an 8×8 2D regular lattice, using dynamic programming. At each of three temperatures, ten random samples are displayed and two empirical expectations are computed. The main Computer Exercise is Computer Exercise 7. Those before it are shorter, easier, and are used as building blocks or debugging tools for that Exercise. Computer Exercise 8 boils down to using the implementation from Computer Exercise 7. Finally, in Problem 2 you are asked to analyze the results from Computer Exercise 8.

Note well: results from this HW assignment will be reused in the next one.

Version Log

- 1.01, 21/4/2020. Corrected the title (HW6 → HW5)
- 1.00, 20/4/2020. Initial release.

Contents

1 Dynammic Programming in the Ising Model **1**
1.1 The Clique Functions in the Ising Model 1
1.2 Brute Force on Small Lattices 2
1.3 Dynamic Programming on an 8×8 Lattice 5
1.4 Detailed Notes and How to Debug 6

1 Dynammic Programming in the Ising Model

1.1 The Clique Functions in the Ising Model

Computer Exercise 1 *Implement a function, G, whose input is*

1. *a 1D numpy array, denoted here by* r *and in your code by* `row_s`*;*

2. *a scalar* Temp*,*

and whose output is the scalar

$$\exp\left(\frac{1}{\text{Temp}}\sum_{i=1}^{\text{length}(r)-1} r_i r_{i+1}\right). \tag{1}$$

It should be a single line of code, and you must not use a loop (nor a list/dict comprehension). ♢

Computer Exercise 2 *Implement a function,* F*, whose input is*

1. *two 1D numpy arrays of the same length, denoted here by* r *and* $\tilde{r}$ *and in your code by* `row_s` *and* `row_t`*;*

2. *a scalar* Temp*,*

and whose output is the scalar

$$\exp\left(\frac{1}{\text{Temp}}\sum_{i=1}^{\text{length}(r)} r_i \tilde{r}_i\right). \tag{2}$$

Again, it should be a single line of code, you must not use a loop (nor a list/dict comprehension). ♢

1.2 Brute Force on Small Lattices

Some of the computer exercises in this section require you to use an obscene number (from a programmer's perspective) of nested loops. Thus, you can (but do not have to) exploit Python's itertools module to make this a bit more elegant.

Computer Exercise 3 *In the case of a* 2×2 *lattice,* i.e.

$$S = \{(i,j)\}_{1\le i\le 2, 1\le j\le 2}, \tag{3}$$

compute Z_{Temp} *(for three different values of* Temp *where* $\text{Temp} \in \{1, 1.5, 2\}$*) using brute force (use 4 nested For loops, one for each of the* x_s*'s, the looping is done over the values that* x_s *can take:* $\{-1, 1\}$*). To help you debug: For* $\text{Temp} = 1$*, your result should be* $Z_{\text{Temp}} = 121.23\ldots$*.* ♢

Computer Exercise 4 *In the case of a* 3×3 *lattice,* i.e.

$$S = \{(i,j)\}_{1\le i\le 3, 1\le j\le 3}, \tag{4}$$

compute Z_{Temp} *(for three different values of* Temp *where* $\text{Temp} \in \{1, 1.5, 2\}$*) using brute force (use 9 nested For loops, one for each of the* x_s*'s, the looping is done over the values that* x_s *can take:* $\{-1, 1\}$*). To help you debug: For* $\text{Temp} = 1$*, your result should be* $Z_{\text{Temp}} = 10^5 \cdot 3.65\ldots$*.* ♢

Obviously, 9 For loops is pretty ugly. We will see below we can use fewer loops, even if we remain in a brute-force approach.

The most computationally-efficient approach, to achieve the main goal of this HW assignment (exact sampling from the Ising model), is to use a raster or boustrophedonic[1] site-visitation schedule and then compute conditional probabilities, site-by-site, in a backward ordering. The number of computations is $O(64 \cdot 2^9) = O(2^{15})$. **However, this is NOT what you are asked to do. Instead, you are requested to do something much simpler (and sub-optimal):** the programming burden becomes substantially easier if the 8×8 lattice is represented as a Markov chain with 8 sites (where each of them encodes an entire row from the original 8×8 lattice) and 8 corresponding variables, $y_1, \ldots, y_8$. Although computational complexity becomes $O(8 \cdot 2^{16}) = O(2^{19})$, the programming complexity is vastly reduced – so we will go with that. Each site variable, y_s (where $s = 1, \ldots, 8$) has 2^8 states, $y_s \in \{0, 1, \ldots, 255\}$, corresponding to the 2^8 possible configurations of row s in the Ising lattice. The correspondence is thus $y_s \leftrightarrow (x_{(s,1)}, \ldots, x_{(s,8)})$. A convenient mapping between y_s and $(x_{(s,i)})_{i=1}^8$ is to use the 8-bit binary representation of y_s, together with the conversion of 0's to -1's.

Example 1 *Suppose $y_s = 136$. The 8-bit binary representation of 136 is '10001000'. Therefore, the corresponding row (*i.e.*, $(x_{(s,i)})_{i=1}^8$) is [1 -1 -1 -1 1 -1 -1 -1].* ◊

The following helper function computes the $y_s \to (x_{(s,i)})_{i=1}^8$ mapping.

```
import numpy as np
def y2row(y,width=8):
    """
    y: an integer in (0,...,(2**width)-1)
    """
    if not 0<=y<=(2**width)-1:
        raise ValueError(y)
    my_str=np.binary_repr(y,width=width)
    # my_list = map(int,my_str) # Python 2
    my_list = list(map(int,my_str)) # Python 3
    my_array = np.asarray(my_list)
    my_array[my_array==0]=-1
    row=my_array
    return row
```

Remark 1 *Note that in this HW assignment, the y's merely stand for groups of x's, used as a mechanism for simplifying the programming. These y's should not be confused with "observations". In this assignment we use only the prior, $p(x)$, and there are no noisy observations;* i.e.*, this HW assignment does not involve a posterior distribution. However, if we let z denote hypothetical observations, the sampling mechanism from this HW assignment can be easily adjusted to sampling from the posterior, $p(x|z)$, in the case that each data point is connected in the graph to a single site in the original graph of $p(x)$;* i.e.*, if the likelihood, $p(z|x)$, factorizes as $p(z|x) = \prod_{s \in S} p(z_s|x_s)$. As we saw in class, the only thing*

[1] Namely, from right to left and from left to right in alternate lines. The word came from Greek and refers to how an ox turns when plowing a field.

that will have to change is the functional form of the clique functions. But again, to clarify, this is not something you are asked to do in this HW assignment. ◊

Computer Exercise 5 *In the case of a* 2×2 *lattice, converted to a chain of length 2, compute* Z_{Temp} *(for three different values of* Temp *where* $\text{Temp} \in \{1, 1.5, 2\}$*) using brute force (use 2 nested For loops, one for each of the* y_s*'s, the looping is done over the values that* y_s *can take:* $\{0, 1, 2, 3\}$*). In effect,*

$$p(y) = p(y_1, y_2) = \frac{1}{Z_{\text{Temp}}} G(y_1)G(y_2)F(y_1, y_2) \tag{5}$$

and

$$Z_{\text{Temp}} = \sum_{y} G(y_1)G(y_2)F(y_1, y_2) = \sum_{y_1}\sum_{y_2} G(y_1)G(y_2)F(y_1, y_2) \tag{6}$$

where the singleton function

$$G(y_s) = \exp\left(\frac{1}{\text{Temp}} x_{(s,1)}x_{(s,2)}\right), \qquad s = 1, 2 \tag{7}$$

representing, for each of the two rows, s*, the intra-row pair clique from the original graph, and a pair clique function*

$$F(y_1, y_2) = \exp\left(\frac{1}{\text{Temp}} \sum_{i=1}^{2} x_{(1,i)}x_{(2,i)}\right) \tag{8}$$

representing the product of the two inter-row pair cliques from the original graph.

In your implementation, use the provided helper function ***y2row*** *(in the second argument, pass width=2 since here there are only two* x_s*'s in each row) and your implemented* G *and* F *from the previous exercises.*

To help you debug: For $\text{Temp} = 1$*, your result should be* $Z_{\text{Temp}} = 121.23\ldots$ *(which is of course the same result from before).* ◊

Computer Exercise 6 *In the case of a* 3×3 *lattice, converted to a chain of length 3, compute* Z_{Temp} *(for three different values of* Temp *where* $\text{Temp} \in \{1, 1.5, 2\}$*) using brute force (use 3 nested For loops, one for each of the* y_s*'s, the looping is done over the values that* y_s *can take:* $\{0, 1, \ldots, 7\}$*). In effect,*

$$p(y) = p(y_1, y_2, y_3) = \frac{1}{Z_{\text{Temp}}} G(y_1)G(y_2)G(y_3)F(y_1, y_2)F(y_2, y_3) \tag{9}$$

and

$$\begin{aligned} Z_{\text{Temp}} &= \sum_{y} G(y_1)G(y_2)G(y_3)F(y_1, y_2)F(y_2, y_3) \\ &= \sum_{y_1}\sum_{y_2}\sum_{y_3} G(y_1)G(y_2)G(y_3)F(y_1, y_2)F(y_2, y_3) \end{aligned} \tag{10}$$

where the singleton function

$$G(y_s) = \exp\left(\frac{1}{\text{Temp}} \sum_{i=1}^{2} x_{(s,i)}x_{(s,i+1)}\right), \qquad s = 1, 2, 3 \tag{11}$$

representing, for each of the three rows, s, the product of the two intra-row pair cliques from the original graph, and pair clique functions

$$F(y_s, y_{s+1}) = \exp\left(\frac{1}{\text{Temp}} \sum_{i=1}^{3} x_{(s,i)} x_{(s+1,i)}\right) \qquad s = 1, 2 \tag{12}$$

representing, for each of the two pairs of rows, $(s, s+1)$, the product of the three inter-row pair cliques from the original graph.

In your implementation, use the provided helper function ***y2row*** *(in the second argument, pass width=3 since here there are three x_s's in each row) and your implemented G and F from the previous exercises.*

To help you debug: For $\text{Temp} = 1$*, your result should be* $Z_{\text{Temp}} = 10^5 \cdot 3.65\ldots$ *(which is of course the same result from before).* ♢

1.3 Dynamic Programming on an 8×8 Lattice

The following Computer Exercise is the main task in this HW assignment.

Computer Exercise 7 *Using dynamic programming and an 8×8 lattice, build an exact sampler for each of the three temperatures,* $\text{Temp} \in \{1, 1.5, 2\}$*.* ***The details for how to do it appear at the notes at the end of this document.*** *Use the samplers to generate ten independent samples (to clarify, each such sample is an entire 8×8 image) at each of the three temperatures, and display these as thirty images all in a single figure (three rows, one for each temperature, and ten columns of 8×8 black-and-white images; the subplot command should be useful here). When displaying your images using plt.imshow (in Python), you should use the Interpolation="None" option (*i.e.*, passing the string "None", not to be confused with Interpolation=None, which passes the Python built-in None).*
If your result for $\text{Temp} = 2$ *looks completely random and does not make sense, perhaps you should first solve Problem 1.* ♢

Remark 2 *In the problem above, when* $\text{Temp} = 1$*, some of you may be puzzled to see that most of the samples (where each sample is an entire 8-by-8 image) are "all blue" (or "all black", depending on the colormap you are using) and you will be wondering how come this is not symmetric. In effect, you will feel that it makes sense that most of the images are unicolor, but you would have expected that about half of them will be "all blue" ("-1") and about half of them will be "all red" ("+1"). Well, this is not how imshow works by default. Here is what is going on: if you hover with the mouse over the pixels in these unicolor images, you will see (at the bottom of the figure) that, in fact, some of them are all "-1" and some of them are all "+1", even though both cases are "all blue". This is because by default, imshow(img) scales the display such that the blue stands for img.min(), and red stands for img.max(). If var(img)=0 (*i.e.*, all pixels share the same value), then img.min()=img.max() so only a single color (blue) is used. This is regardless whether all the pixels are -1 or all the pixels are +1. One way to overcome this visualization issue is using the vmin and vmax optional parameters: imshow(...,vmin=-1, vmax=+1). Read imshow's documentation about it. That said, later in our next HW assignment, when you show images corrupted by real-valued noise, you should avoid this [-1,1] range because it will trim the values which may be outside that range.* ♢

Problem 1 *A student, who used Python 2 (as opposed to Python 3), encountered a weird problem when working on Computer Exercise 7. While the resulting samples for* $\text{Temp} = 1$ *and* $\text{Temp} = 1.5$ *look as one might expect, for* $\text{Temp} = 2$ *the samples that the student obtained looked completely random with no structure whatsoever. Explain what programming bug the student probably had, and from which distribution (instead of the Ising model) over binary images that student really sampled from.* ♢

Computer Exercise 8 *Using the three samplers you implemented above, at each of the three temperatures, draw 10,000 samples,* $x(1), \ldots, x(10000)$ *(each such sample is an* 8×8 *binary image) and compute two empirical expectations:*

$$\widehat{E}_{\text{Temp}}(X_{(1,1)}X_{(2,2)}) \triangleq \frac{1}{10000}\sum_{n=1}^{10000} x_{(1,1)}(n)x_{(2,2)}(n) \tag{13}$$

$$\widehat{E}_{\text{Temp}}(X_{(1,1)}X_{(8,8)}) \triangleq \frac{1}{10000}\sum_{n=1}^{10000} x_{(1,1)}(n)x_{(8,8)}(n) \tag{14}$$

where $\text{Temp} = 1, 1.5,$ *and* 2 *and where* $x_{(i,j)}(n)$ *is the value at the* (i,j)*-th lattice site of sample* n*.* ♢

Problem 2 *Using the results of the Computer Exercise above, explain the relative values of* $\widehat{E}$*, in terms of the spatial distance of the lattice sites and in terms of the temperature.* ♢

1.4 Detailed Notes and How to Debug

1. You should do the sampling in terms of the y_s's, not the x_s's, since it is considerably simpler.

2. In this representation, where we use an 8-length Markov chain (with each of its variables taking values in $\{0, 1, \ldots, 255\}$) to represent a binary MRF defined over an 8×8 lattice, there are two kinds of clique functions:

   (a) Singletons of the form

   $$G(y_s) = \exp\left(\frac{1}{\text{Temp}}\sum_{i=1}^{7} x_{(s,i)}x_{(s,i+1)}\right), \qquad s = 1, 2, \ldots, 8 \tag{15}$$

   representing, for each of the eight rows, s, the product of the seven intra-row pair cliques from the original graph;

   (b) pair cliques of the form

   $$F(y_s, y_{s+1}) = \exp\left(\frac{1}{\text{Temp}}\sum_{i=1}^{8} x_{(s,i)}x_{(s+1,i)}\right) \qquad s = 1, 2, \ldots, 7 \tag{16}$$

   representing, for each of the seven pairs of rows, $(s, s+1)$, the product of the eight inter-row pair cliques from the original graph.

Thus,

$$p(y_1, \ldots, y_8) \propto \left(\prod_{s=1}^{8} G(y_s) \right) \left(\prod_{s=1}^{7} F(y_s, y_{s+1}) \right) . \tag{17}$$

By now, you should already have these two types of functions implemented.

The dynamic programming iterations, adopting the site-visitation schedule $1, 2, \ldots, 8$, is then

$$T_1(y_2) = \sum_{y_1=0}^{255} G(y_1) F(y_1, y_2) \qquad y_2 \in \{0, 1, \ldots, 255\} \tag{18}$$

$$T_k(y_{k+1}) = \sum_{y_k=0}^{255} T_{k-1}(y_k) G(y_k) F(y_k, y_{k+1}) \qquad 2 \le k \le 7 \quad y_{k+1} \in \{0, 1, \ldots, 255\} \tag{19}$$

$$\mathbb{R} \ni T_8 = \sum_{y_8=0}^{255} T_7(y_8) G(y_8) \quad (= Z_{\text{Temp}}) \tag{20}$$

Working backwards:

$$p_8(y_8) = \frac{T_7(y_8) G(y_8)}{Z_{\text{Temp}}} \qquad y_8 \in \{0, 1, \ldots, 255\} \tag{21}$$

$$p_{k|k+1}(y_k | y_{k+1}) = \frac{T_{k-1}(y_k) G(y_k) F(y_k, y_{k+1})}{T_k(y_{k+1})} \qquad k = 7, 6, \ldots, 2 \quad y_k, y_{k+1} \in \{0, 1, \ldots, 255\} \tag{22}$$

$$p_{1|2}(y_1 | y_2) = \frac{G(y_1) F(y_1, y_2)}{T_1(y_2)} \tag{23}$$

3. On a computer:
   (a) p_8 is a 1D array of length $2^8 = 256$;
   (b) for each $k \in \{1 \ldots, 7\}$, $p_{k|k+1}$ is a $2^8 \times 2^8 = 256 \times 256$ array (so you should have 7 such 2D arrays: $p_{7|8}$; $p_{6|7}$; $p_{5|6}$; $p_{4|5}$; $p_{3|4}$; $p_{2|3}$; $p_{1|2}$).

4. Sampling is fast; computing the conditional probabilities is slow. So you might want to save the conditional probabilities (each of which is a 256×256 array as mentioned above) after they are computed (or, at least save the "T functions").

5. Debug your dynamic-programming implementation on 2×2 and 3×3 lattices by computing Z_{Temp}, the normalizing constant, and the associated T functions. Here are some results for small lattices (with Temp = 1):

```
2x2 lattice results:
T1: [ 21.18917525   8.20463255   8.20463255  21.18917525]
T2 = Z: 121.232931344
```

```
3x3 lattice results:
T1: [ 155.37102759   46.44297052   31.70116107   46.44297052   46.44297052
   31.70116107   46.44297052  155.37102759]
T2: [ 23416.16435187   4634.76802124   3916.10003703   4634.76802124
   4634.76802124   3916.10003703   4634.76802124  23416.16435187]
T3 = Z: 365645.749136
```

In other words, before you start the sampling (which is done in the "backward pass", in effect, in ordering 8,7,6,...,1), you should debug on a small lattice (2×2 or 3×3), making sure you get the right results for the "forward pass". In effect, computing the T's, the last of which is $Z_{\rm Temp}$.

To be more concrete:

- For a 2×2 lattice:
  Each y_s (for $s \in \{1, 2\}$), *i.e.*, a row, is a 1D array of length $2^2 = 4$, whose entries take integral values between 0 and $3 = 2^2 - 1$, inclusive. T_1 is a function of y_2. Since y_2 has 4 different values, it follows that T_1 should be represented as a 1D array of length 4. Particularly, since $T_1(y_2) = \sum_{y_1=0}^{3} G(y_1)F(y_1, y_2)$, it follows that the 4 entries of the T_1 array are:

$$T_1(y_2 = 0) = \sum_{y_1=0}^{3} G(y_1)F(y_1, 0) \tag{24}$$

$$T_1(y_2 = 1) = \sum_{y_1=0}^{3} G(y_1)F(y_1, 1) \tag{25}$$

$$T_1(y_2 = 2) = \sum_{y_1=0}^{3} G(y_1)F(y_1, 2) \tag{26}$$

$$T_1(y_2 = 3) = \sum_{y_1=0}^{3} G(y_1)F(y_1, 3)\,. \tag{27}$$

  T_2 is a scalar because it is the last T (since we have only two rows). In fact, $T_2 = Z_{\rm Temp}$, the normalizing constant:

$$T_2 = Z_{\rm Temp} = \sum_{y_2=0}^{3} T_1(y_2)G(y_2)\,. \tag{28}$$

  Now you can compare it with the brute-force result.

- For a 3x3 lattice: Each y_s (for $s \in \{1, 2, 3\}$), *i.e.*, a row, is a 1D array of length $2^3 = 8$, whose entries take integral values between 0 and $7 = 2^3 - 1$, inclusive. T_1 is a function of y_2. Since y_2 has $2^3 = 8$ different values, it follows that T_1 should be represented as a 1D array of length 8. Particularly, since $T_1(y_2) = \sum_{y_1=0}^{7} G(y_1)F(y_1, y_2)$,

it follows that the 8 entries of the T_1 array are:

$$T_1(y_2 = 0) = \sum_{y_1=0}^{7} G(y_1)F(y_1, 0) \tag{29}$$

$$T_1(y_2 = 1) = \sum_{y_1=0}^{7} G(y_1)F(y_1, 1) \tag{30}$$

$$\vdots \tag{31}$$

$$T_1(y_2 = 7) = \sum_{y_1=0}^{7} G(y_1)F(y_1, 7). \tag{32}$$

Now, T_2 is a function of y_3. Since y_3 has $2^3 = 8$ different values, it follows that T_2 should be represented as a 1D array of length 8. Particularly, since $T_2(y_3) = \sum_{y_2=0}^{7} T_1(y_2)G(y_2)F(y_2, y_3)$, it follows that the 8 entries of the T_2 array are:

$$T_2(y_3 = 0) = \sum_{y_2=0}^{7} T_1(y_2)G(y_2)F(y_2, 0) \tag{33}$$

$$T_2(y_3 = 1) = \sum_{y_2=0}^{7} T_1(y_2)G(y_2)F(y_2, 1) \tag{34}$$

$$\vdots \tag{35}$$

$$T_2(y_3 = 7) = \sum_{y_2=0}^{7} T_1(y_2)G(y_2)F(y_2, 7)\,. \tag{36}$$

T_3, since it is the last T (since we have only 3 rows) is a scalar. In fact, $T_3 = Z_{\text{Temp}}$, the normalizing constant.

$$T_3 = Z_{\text{Temp}} = \sum_{y_3=0}^{7} T_2(y_3)G(y_3)\,. \tag{37}$$

Now you can compare it with the brute-force result.

- For an 8×8 lattice: Each y_s (for $s \in \{1, 2, 3, 4, 5, 6, 7, 8\}$), *i.e.*, a row, is a 1D array of length 8, whose entries take integral values between 0 and $255 = 2^8 - 1$, inclusive. So $T_1, T_2, ..., T_7$ are arrays of length $2^8 = 256$, while $T_8 = Z_{\text{Temp}}$ since it is the last one.

  Once you are done with this forward pass, you use these T's to compute the p's.

  p_8 is function of y_8 (in fact it is the pmf over the 256 states y_8 can take). So p_8 should be represented as a 1D array of length $2^8 = 256$.

  $p_{7|8}(y_7, y_8)$ is the probability mass function of y_7 given y_8. So for every value of y_8 – and we have 256 of these – we should have 256 values to cover the probability of all possible 256 states y_7 may assume. So we represent $p_{7|8}$ using a 256 by 256 array.

Similarly for $p_{6|7}(y_6, y_7)$, $p_{5|6}(y_5, y_6)$, $\ldots$, $p_{1|2}(y_1, y_2)$.
Finally, to produce the entire sample, we first sample y_8 from p_8 (one way to sample from distribution of a discrete RV is using np.random.choice). Then, we pick the row (or column, depends how you built your $p_{7|8}$) that corresponds to that y_8 we drew, and use it to sample $y_{7|8}$. And so on till we end with sampling $y_{1|2}$.

Remark 3 *As usual, when writing p, we could have also absorbed singletons into the pairwise functions,* e.g.*:*

$$\begin{aligned} p(y_1, \ldots, y_8) \propto & \underbrace{G(y_1)F(y_1, y_2)}_{H_{1,2}(y_1,y_2)} \underbrace{G(y_2)F(y_2, y_3)}_{H_{2,3}(y_2,y_3)} \underbrace{G(y_3)F(y_3, y_4)}_{H_{3,4}(y_3,y_4)} \\ & \times \underbrace{G(y_4)F(y_4, y_5)}_{H_{4,5}(y_4,y_5)} \underbrace{G(y_5)F(y_5, y_6)}_{H_{5,6}(y_5,y_6)} \underbrace{G(y_6)F(y_6, y_7)}_{H_{6,7}(y_6,y_7)} \\ & \times \underbrace{G(y_7)G(y_8)F(y_7, y_8)}_{H_{7,8}(y_7,y_8)} = \prod_{s=1}^{7} H_{s,s+1}(y_s, y_{s+1}) \,. \end{aligned} \tag{38}$$

$\Diamond$