Progetto Computazionale

Emanuele Marconato

Settembre 2020

Abstract

In questo progetto analizzo il modello di Ising 2D, facendone una simulazione con il metodo Metropolis-Hasting per ricavare nel limite di numero di spin $N \to \infty$ l'andamento delle osservabili medie in funzione della tempertura. I valori medi estratti vengono poi confrontate con quelli estratti dalla sorgente https://github.com/coppolachan/magneto, in cui viene utilizzato l'algoritmo di Swedsen-Wang.
In seguito, estraendo dalla simulazione precedente 13 dataset sulle configurazioni di spin a temperatura fissata, alleno delle restricted boltzmann machine (RBM), tramite Contrastive Divergence(CD). Da queste è infine possibile ricavare le correlazioni tra spin, come risultato del corretto training delle RBM.

1 Simulazione del reticolo di Spin

Il modello di Ising è dato da una hamiltoniana di interazione tra primi vicini $H = -J\sum_{\langle i,j\rangle} s_i s_j$, dove i valori di spin ammessi sono $s_i = \pm 1$. A fissata temperatura, è possibile descrivere il modello all'equilibrio con l'ensemble canonico.
Per quanto riguarda la simulazione del reticolo di spin, ho generato un dataset in un range di temperature $T \in [1.8, 3]$, con punto critico per $N \to \infty$ situato in $\beta_c = \log(1+\sqrt{2})$, posto $2J/k_B = 1$. Presa una configurazione random degli spin, il primo passo è stato lasciare evolvere all'equilibrio il reticolo, con il seguente pseudo-codice, tramite Metropolis-Hasting:

- Viene estratto un punto del reticolo e viene valutata l'energia tra primi vicini $E_i = -J\sum_{\langle j\rangle} s_i s_j$;
- Vengono aggiornati gli spin del reticolo con i seguenti condizionali:

  if $E_i < 0$ **then** $\sigma_i \leftarrow -\sigma_i$

  else: definita $p_i = \exp(-\beta E_i)$ per $E_i > 0$ **and if** *random-uniform*$(0 < x < 1) < p_i$ **then** $\sigma_i \leftarrow -\sigma_i$;

Il risultato per $T = 2.6$ è il seguente:

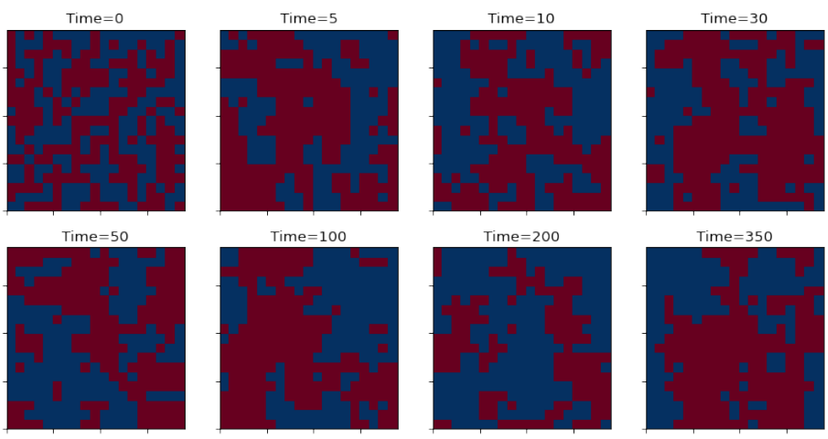

Figure 1: Evoluzione temporale verso l'equilibrio per T= 2.6. Dimensioni del reticolo 20×20.

La seguente simulazione è stata effettuata per un range di temperatura, nel caso di un reticolo di dimensione 20×20.

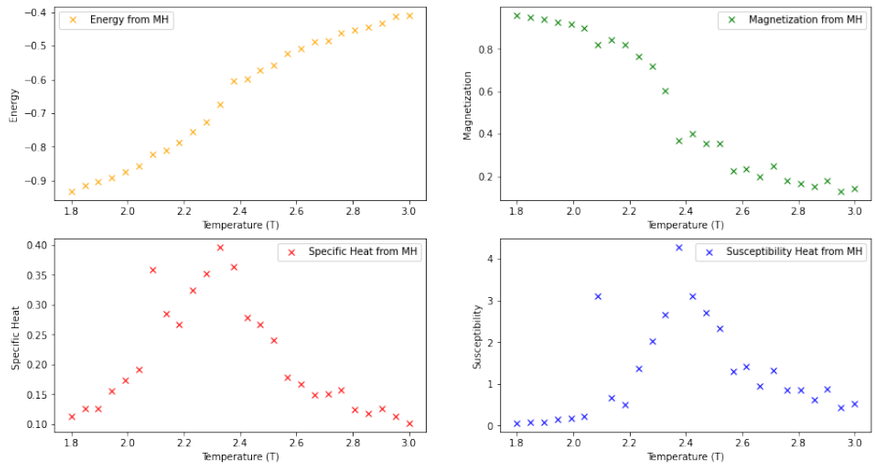

Figure 2: Simulazione per un reticolo 20×20.

1.1 Implementazione in Python

Nel seguente link RBMs vi è il codice per il progetto. All'interno di ising.py è stata implementata la classe Ising, al cui interno vi sono le seguenti funzioni:

- __init__(self, N): member function per l'inizializzazione della classe. self.__N è passata come variabile privata;
- mcmove(self, config, beta): implementa un passo con l'algoritmo di Metropolis-Hasting per ogni spin nel reticolo.
- simulate(self, Spin_Lattice, beta, boolean = False): è una funzione che simula l'evoluzione all'equilibrio per una data $\beta = 1/T$. Viene inoltre inserita della white noise durante il ciclo, invertendo la direzione di al più due spin estratti casualmente.
- obs_simulation(self, eqSteps= 1000, mcSteps= 1000, tstart = 1.8, tstop=3.0, tsteps=15): simula le osservabili nel range [tstart, tstop], per un totale di tsteps valori di temperatura.

I risultati per simulate e obs_simulation sono presentati in Figure 1 e Figure 2.
Per il training delle RBM, è stata invece utilizzata una simulazione con magneto, estraendo 10^5 stati per temperatura nel range $[1.8, 3]$, a dimensione $L = 8$. L'algoritmo scelto per la simulazione, seguendo [1], è di Swendsen-Wang:

```
~/magneto-master$ ./magneto -L=6 -TMin=1.8 -TMax=3
-TSteps=13 -record=main -states=spins  -N1=1000
-N3=5  -alg=sw -N2=100000
```

2 Restricted Boltzmann Machine

Le restricted Boltzmann machines sono un modello generativo *energy-based*, a grafo indiretto. L'energia associata alle unità visibili v e nascoste h si può scrivere:

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{b}^T\mathbf{v} - \mathbf{c}^T\mathbf{h} - \mathbf{v}^T W\mathbf{h}$$

dove b, c, W sono i parametri di learning. L'interazione tra i due layer è descritta dalla matrice W, come riportato in figura 3. La probabilità congiunta di una data configurazione (v, h) è data da:

$$p(\mathbf{v},\mathbf{h}) = \frac{1}{Z_{RBM}} \exp -E(\mathbf{v},\mathbf{h})$$

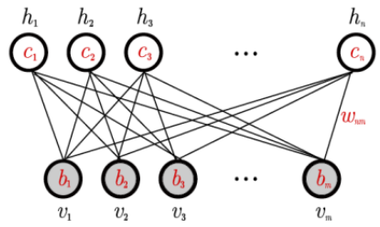

Figure 3: Esempio di una RBM.

Nel caso di entrate a valore booleano per le unità visibili e nascoste, la struttura semplice della RBM permette di stimare le probabilità condizionate tramite:

$$P(h_i = 1|\mathbf{v}) = \sigma(\mathbf{v}^T W_{:,i} + b_i)$$

dove $\sigma(z) = 1/(1 + e^{-z})$ e $W_{:,i}$ identifica la colonna i della matrice di pesi (nel caso della probabilità condizionale per v data h, si ha una forma simile).

2.1 Obiettivo dell'applicazione della RBM

L'obbiettivo della simulazione è addestrare la RBM a imparare la distribuzione target $p(v|\theta)$ sul modello di Ising 2D ad una data temperatura e valutarne le osservabili medie in funzione della temperatura, in particolare ricostruire la matrice delle correlazioni tra spin. Il risultato è stato precedentemente ottenuto, vedi ref T.Giani-2018 (vedi sezione 4).

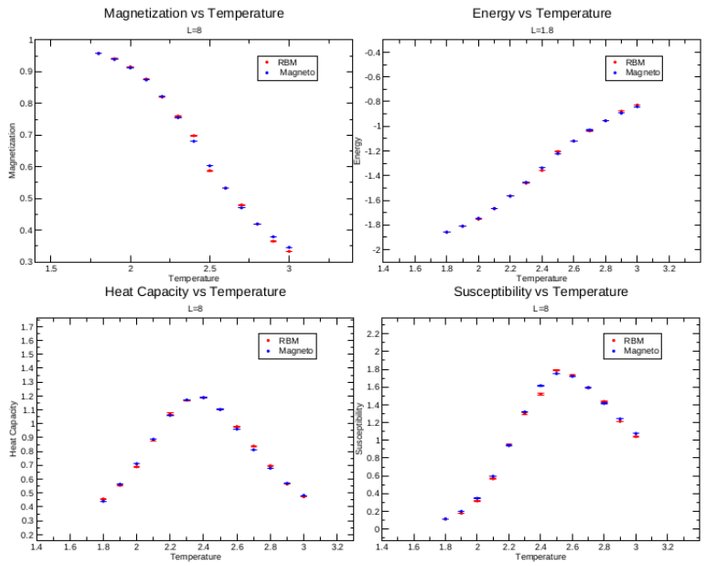

Figure 4: Osservabili per il modello di Ising 2D in funzione della temperatura. Immagine presa da [1].

3 Training della RBM

Partendo da un'inizializzazione random dei pesi $\theta = \{W, c, b\}$, si va a creare un dataset delle possibili configurazioni a fissata temperatura, estraendo le configurazioni con probabilità legata all'ensemble $p = e^{-\beta E\{s_i, s_j\}}/Z_{can}$.
Per ogni configurazione estratta dal dataset $S = \{\mathbf{s}_1, \mathbf{s}_2, ...\}$, $\mathbf{v} \leftarrow \mathbf{s}$ ottenendo la likelihood:

$$\mathcal{L}(\theta|\mathbf{v}) = \prod_i p_{RBM}(v_i|\theta)$$

$$\theta_{n+1} = \theta_n + \epsilon \partial_\theta \log \mathcal{L}$$

Il gradiente per la funzione di verosimilianza può essere scomposto per ogni sample in

$$\nabla_\theta \log p(v|\theta) = \nabla_\theta \log \tilde{p}(v|\theta) - \nabla_\theta \log Z(\theta)$$

Nel caso in esame, per i parametri W, b, c si ottiene rispettivamente:

$$\begin{aligned}
\frac{1}{|S|}\sum_{v\in S}\frac{\partial \log p(v|\theta)}{\partial W_{ij}} &= \mathbb{E}_{data}[p(h_i = 1|v,\theta)v_j] - \mathbb{E}_{model}[p(h_i = 1|v',\theta)v'_j] \\
\frac{1}{|S|}\sum_{v\in S}\frac{\partial \log p(v|\theta)}{\partial b_j} &= \mathbb{E}_{data}[v_j] - \mathbb{E}_{model}[v'_j] \\
\frac{1}{|S|}\sum_{v\in S}\frac{\partial \log p(v|\theta)}{\partial c_i} &= \mathbb{E}_{data}[p(h_i = 1|v,\theta)] - \mathbb{E}_{model}[p(h_i = 1|v',\theta)]
\end{aligned} \tag{1}$$

A questo punto si può utilizzare l'algoritmo di Contrastive Divergence per l'ottimizzazione col gradient-ascent, andando a stimare i termini $\mathbb{E}_{model}$ in base alla ripetizione di GIbbs sampling k_CD . L'ultimo passo durante il training è monitorare le epoche con una stima della likelihood

$$\mathcal{L}(v|\theta) = \frac{1}{Z_{RBM}}\prod_{i=1}^{M} p^*(v_i|\theta) \tag{2}$$

Stimare la funzione di partizione per la RBM non è banale, ne si può dare una stima per Importance Sampling estraendo un campione $v^{(i)} \propto p_0(v)$, dove $p_0(v)$ è una prior sulla distribuzione attesa p_{RBM}. Si ottiene

$$Z_{RBM}/Z_0 = \frac{1}{m}\sum_{i=1}^{m}\frac{p^*_{RBM}(v_i)}{p^*_0(v_i)}$$

implementato con il metodo di Annealing Importance Sampling (AIS).
In questo caso la prior è stata scelta per $\beta = 0$, con funzione di partizione $Z_0 = 2^N$ e $p^*_0(v) = 1$. Viene poi stimato il valore di $\log Z_{RBM}$ tramite AIS per tsteps=1000, fino a raggiungere $\beta = 1$ (di sampling per la RBM). Per ulteriori dettagli, si veda Appendice A. Un altro stimatore utilizzato è il reconstruction error

$$\epsilon = \frac{1}{N_{batch}}\sum_{\text{batches}b}\frac{1}{|S^{(b)}|}\sum_{v\in S^{(b)}}|v^{(0)} - v^{(1)}|^2$$

In Figure 5 vengono plottati rispettivamente i valori di $\log \mathcal{L}$ ed ϵ per una RBM allenata su un dataset di 10^5 configurazioni a $T = 1.8$.

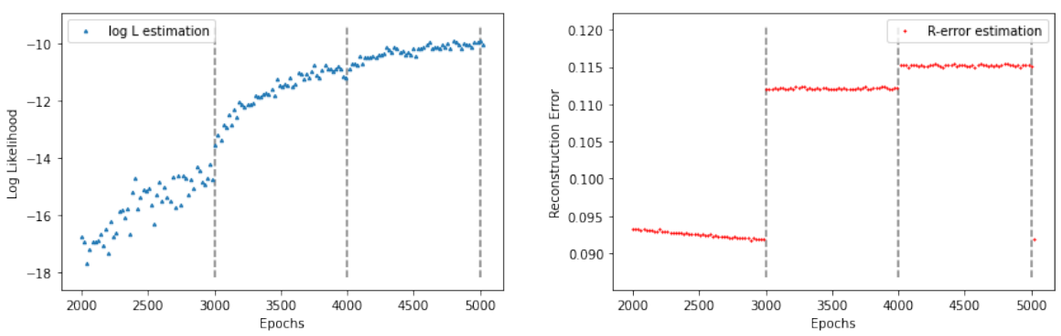

Figure 5: Plot della $\log \mathcal{L}$ e del reconstruction error ϵ per $T = 1.8$. I valori sono stati registrati dopo 2000 epoche di training iniziale. Fino a 3000 epoche sono stati impostati learning rate lr=0.1 e k_CD=1; da 3000 a 4000 con lr=0.01 e k_CD=5; da 4000 in poi lr=0.001 e k_CD=10.

3.1 Implementazione in Python

All'interno del modulo main_module.py sono state implementate la classe RBM ed alcune funzioni per il training delle RBM. Di seguito riporto le principali:

- class txt_Ising_dataset(Dataset): classe che si appoggia all'oggetto torch.utils.data.Dataset ed esegue l'importazione e lettura delle configurazioni di spin da file;
- class RBM(): classe dotata della member function __init__(self, nv, nh) che inizializza i pesi θ passando le variabili private self.__nv e self.__nh;

  sample_h(self, x, beta =1): funzione che restituisce $p(h|v,\theta)$ e h dalla Bernulliana associata;

  train(self, lr, delta_W, delta_b, delta_c): funzione che riceve in input $\Delta\theta$ calcolati tramite (1);
- def train_rbm(rbm, Spin_Config, temp, nb_epoch, k_CD, lr, L, print_log = 1000, reg=False, gamma=0.9): funzione principale di training che riceve in input la RBM[1] e i dati Spin_Config, implementando il training per nb_epochs, con l'algoritmo CD per Gibbs sampling k_CD volte (per l'algoritmo di CD si veda l'Appendice B). Il valore intero print_log viene utilizzato per computare, ogni print_log epoche, la log-likelihood;
- def log_lh(rbm, spins,L, nsamples= 10000): funzione che prende in input la RBM e gli stati spins, eseguendo una stima della log-likelihood per al massimo nsamples dati, estratti casualmente.

Il training è stato implementato con il seguente codice (in formato notebook .ipynb):

Listing 1: Training Example

```
#Before import all necessary libraries
import main_module as mod

#importing the dataset spins0.txt
spins = mod.txt_Ising_dataset("spins180.txt", size= 100000, L=8)
rbm = mod.RBM(64,64)

mod.train_rbm(rbm,spins.imgs, temp=1.8, nb_epoch=3000, k_CD=1,lr=0.1, print_log=20)
mod.train_rbm(rbm,spins.imgs, temp=1.8, nb_epoch=1000, k_CD=5,lr=0.01, print_log=20)
mod.train_rbm(rbm,spins.imgs, temp=1.8, nb_epoch=1000, k_CD=10,lr=0.001, print_log=20)
```

[1] In questo caso rbm viene passata come *by reference* in C++, andando a modificare i pesi θ nello stesso indirizzo di memoria in cui è salvata.

4 Validazione e Risultati Ottenuti

4.1 Classi RBM

All'interno del modulo rbm_classes.py sono state implementate le classi poi utilizzate per la simulazione. La classe originale RBM è stata estesa per ereditarietà alla classe Rbm. Al suo interno sono state implementate le funzioni:

- __init__(self, nv, nh, T, W = None, b = None, c = None) member function del constructor per la classe. In questo caso l'inizializzazione permette l'inserimento dei pesi self.W self.b self.c.
- AIS_sample(self, n_samples=10000, print_means=False) funzione di sampling tramite AIS, che restituisce n_samples stati e una lista contenente le osservabili medie;
- Gibbs_sample(self, k_CD = 1, eqsteps=1000, n_samples=10000, print_means=False) funzione di sampling tramite Gibbs Sampling, con k_CD passi. La funzione effettua eqsteps all'equilibrio e restituisce n_samples stati e una lista contenente le osservabili medie;
- Metropolis_sample(self, eqsteps= 1000, n_samples=10000, print_means=False) funzione di sampling tramite Metropolis-Hasting Sampling. La funzione effettua eqsteps all'equilibrio e restituisce n_samples stati e una lista contenente le osservabili medie.

Sono state poi raccolte le Rbm all'interno della classe mother_rbm:

- __init__(self,nv,nh,**kwargs) metodo di inizializzazione, con in input le dimensioni delle Rbm, passate come variabili private. Il constructor prevede il passaggio anche di keywords arguments per l'inserimento vero e proprio delle Rbm all'interno della classe: viene creato un dizionario con chiavi riferite ai parametri delle Rbm inserite;
- add(self, **kwargs) funzione che permette di aggiungere chiavi al dizionario di Rbm, tramite il passaggio di kwargs;
- simulate_AIS/Gibbs funzioni che implementano la simulazione delle osservabili medie tramite AIS o Gibbs. Restituiscono un vettore contenente le osservabili medie.

4.2 Simulazione e Validazione

All'interno di notebook-RBM.ipynb ho riportato i risultati ottenuti. Sono state raccolte le RBM allenate all'interno della classe dizionario mother_rbm, con etichetta rbm10T per oggetto.
Sono state simulate le osservabili medie in funzione della temperatura, estratte dalle RBM allenate e poi messe in grafico con riferimento al valore estratto da magneto. Sono state inoltre inserite le osservabili simulate con il metodo della sezione 1 (MH).

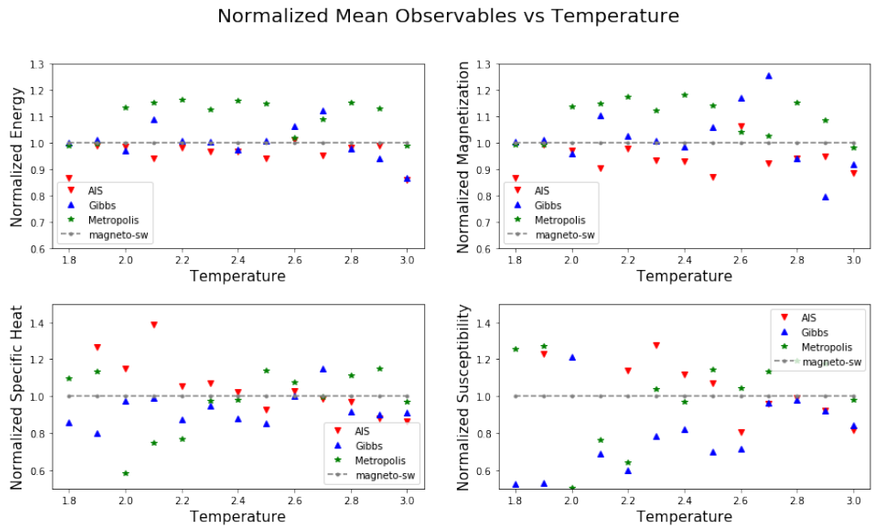

Figure 6: Simulazione delle osservabili medie dalle RBM allenate per AI sampling e GIBBS sampling. Nelle immagini in figura le osservabili sono state normalizzate ai valori calcolati dalla simulazione con magneto.

E' stata inoltre valutata la temperatura critica T_C dall'interpolazione sulla magnetizzazione. Per ogni simulazione effettuata sono stati valutati i coefficienti della funzione interpolatrice

$$m(T|A,B) = \frac{1}{1+e^{AT+B}}$$

Per questo tipo di regressione, non sono stati utilizzate le deviazioni standard per ogni punto di magnetizzazione osservata, poichè la distribuzione del sampling dalle RBM in pochi casi risulta avere una forma gaussiana. E‘ quindi stato utilizzato il Mean-Squared-Error

$$\text{mse} = \frac{1}{13}\sum_{i=1}^{13}[m(T_i|A,B) - m_i^{(data)}]^2$$

Dall'interpolazione per i parametri è stata calcolata la temperatura critica per ogni simulazione effettuata

```
Expected Tc from AIS: 2.567 pm 0.401
Expected Tc from GIBBS: 2.635 pm 0.380
Expected Tc from MH: 2.696 pm 0.142
Expected Tc from theory: 2.269
```

E riportata in grafico l'interpolazione

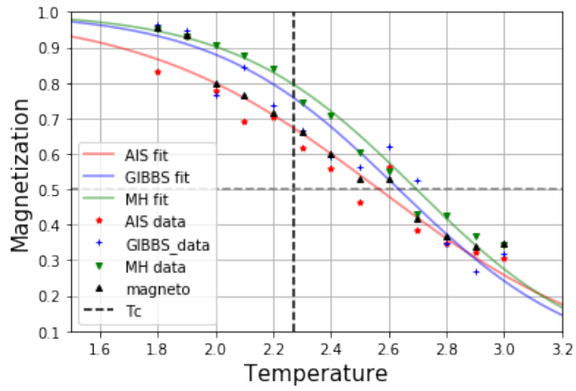

Figure 7: Fit della magnetizzazione per ogni tecnica di simulazione.

E' stata inoltre valutata la matrice di correlazione data da

$$H_{j_1,j_2} = \frac{1}{8}\sum_i \log\frac{(1+e^{c_i+W_{i,j_1}+W_{i,j_2}})(1+e^{c_i})}{(1+e^{c_i+W_{i,j_1}})(1+e^{c_i+W_{i,j_2}})}$$

Il risultato ottenuto seppur non ottimale, si avvicina a quello atteso.

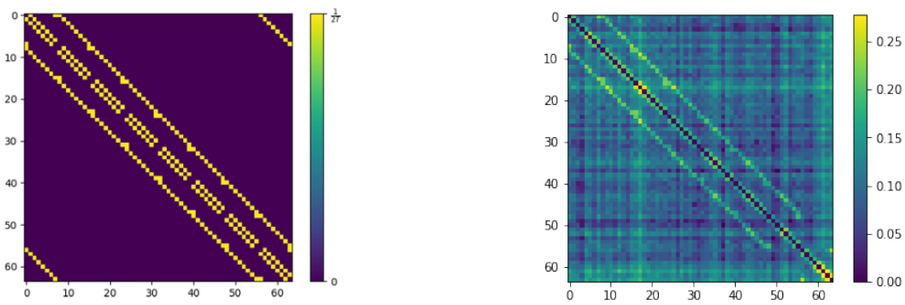

Figure 8: (Sinistra) Matrice delle correlazioni attesa per il modello di Ising a temperatura T. (Destra) Matrice delle correlazioni ottenuta per T=1.8. La scala graduata è stata impostata con massimo pari a 1/2T.

4.3 Discussione dei risultati ottenuti

In questo progetto sono state condotte differenti simulazioni del modello di Ising 2D a dimensione fissata. Con il metodo di Metropolis-Hasting sono stati simulati dei reticoli di dimensione L=8 ed L=20, mentre con i metodi di Annealed Importance Sampling e Gibbs Sampling sono stati simulate le osservabili per un reticolo L= 8 a temperatura fissata. La temperatura critica ottenuta è stata poi confrontata con quella prevista dalla soluzione di Onsager per un reticolo a dimensione infinita. L'andamento delle osservabili medie rispecchia meglio quello atteso all'aumentare delle dimensioni del reticolo, vedi Figure 2.
Tuttavia per allenare le Restricted Boltzmann Machine la scelta delle dimensione del reticolo e della dimensione del dataset sono cruciali: nel caso in esame per L=8 e un dataset di 10^5 configurazioni di spin a temperatura fissata, il tempo medio necessario per completare un epoca è di circa 5 secondi. Ogni 20 epoche è stata stimata la massima verosimilianza: in questo caso il tempo di processamento è risultato di circa 10 secondi. Pur utilizzando gli iperparametri tabulati in [1], le macchine non risultano completamente allenate. In Figure 8 si possono notare alcune linee orizzontali e verticali più luminose rispetto al fondo: questo è probabilmente un sintomo di overfitting. Per ovviare al problema, un approccio possibile sarebbe introdurre una regolarizzazione durante il training, modificando la funzione di massima verosimilianza con una prior Bayesiana. Ad esempio:

$$\log \mathcal{L}(\theta|v) \propto \log \mathcal{L}(v|\theta) - \frac{1}{2}\gamma||\theta||^2$$

dove γ è un iperparametro di regolarizzazione e $||\cdot||^2$ è la norma di Frobenius per i parametri della RBM. E' possibile accendere la regolarizzazione da `train_rbm` impostando `reg=True` e il valore di γ (idealmente bisognerebbe effettuare un grid-search per il γ ideale). In tal caso il numero di epoche di iterazione sarà maggiore rispetto a quelle tabulate in [1].
Il criterio basato sull'early stopping (come utilizzato in [1]) è quindi molto dipendente dal contesto e dall'iniziallizazione dei pesi. Nel caso della presente simulazione la curva di learning ottenuta (Figure 5) è parzialmente soddisfacente ed è possibile che per ulteriori epoche di training $\log \mathcal{L}$ possa ancora crescere. Come verifica, è stato poi valutato il training della RBM a T=1.8 per transfer learning (circa 1500 epoche), ottenendo il seguente risultato per la matrice di correlazione tra spin:

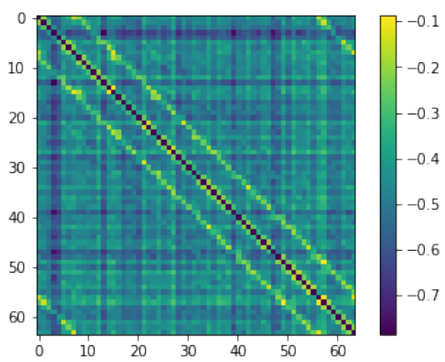

Figure 9: Matrice di Correlazione per la RBM a T=1.8, ri-allenata per transfer learning sui pesi della configurazione a T=1.9.

In questo caso si può notare che, nonostante il fondo sia maggiore, le striscie oblique attese sono più visibili, vedi Figure 8.
Il parziale training delle RBM ha ricadute anche sull'andamento delle osservabili medie estratte per AIS e GIBBS: mentre Energia e Magnetizzazione tendono a situarsi attorno al valore atteso dalla simulazione eseguita con `magneto`, con uno scarto attorno al 10%, il Calore Specifico e la Suscettività (proporzionali ai momenti secondi rispettivamente per Energia e Magnetizzazione) fluttuano maggiormente e sono stati scartati i valori oltre il 50%. In questo caso influiscono maggiormente il numero limitato di dati estratti per AIS, GIBBS e MH (circa 10^3 contro i circa 10^6 in [1]) e la dimensione del reticolo.

4.3.1 Curiosità

Il risultato ottenuto, seppur parziale, mostra l'efficacia delle RBM per apprendere i pattern di correlazione all'interno del reticolo di Ising. Seguendo [1] la simulazione è dunque da considerarsi una verifica per lo studio di modelli meno conosciuti, in cui si spera che una tale architettura di machine learning possa funzionare altrettanto bene.
Il metodo di AIS non è stato utilizzato dagli autori in [1]. Ho pensato comunque di utilizzarlo viste le buone performance sulla stima della funzione di partizione. Il metodo Metropolis-Hasting per le RBM non è stato infine utilizzato ma è possibile ottenere dei sample dalla classe `Rbm`.

A Calcolo della Log Likelihood

Partendo dall'espressione esplicita per $\mathcal{L}(v,h|\theta)$ è stata valutata la likelihood per le unità visibili tramite

$$\mathcal{L}(v|\theta) = \frac{1}{Z_{RBM}} \sum_h \prod p^*(v,h|\theta) \tag{3}$$

dove $p^*(v,h) = e^{-E(v,h)}$. Da [1] si ottiene

$$p(v|\theta) = \frac{1}{Z_{RBM}} \prod_j (e^{b_j v_j}) \prod_i (1 + e^{c_i + \sum_j W_{ij} v_j}) \tag{4}$$

Identificato $p^*_\beta(v) = \exp(-\beta E(v))$, la funzione di partizione è data da

$$Z_{RBM} = Z_1 = \int \mathrm{d}v\, p^*_{\beta=1}(v) = Z_0 \int \mathrm{d}v\, p_0(v) \frac{p^*_1(v}{p^*_0(v)}$$

dove nell'ultimo passaggio è stata riscritto l'integrale tramite la distribuzione di probabilità per $p_{\beta=0}$. In questo caso è possibile stimare la funzione di partizione per Importance Sampling, estraendo le configurazioni di spin per $\beta = 0$, ottenendo

$$Z_1 = Z_0 \frac{1}{M} \sum_{i=1}^{M} \frac{p^*_1(v^{(i)})}{p^*_0(v^{(i)})} = 2^{L^2} \frac{1}{M} \sum_{i=1}^{M} p^*_1(v^{(i)})$$

dove L^2 è il numero totale di spin nel reticolo e $p^*_0 = 1$. Tuttavia, specialmente per le configurazioni a temperatura minore di T_c, l'overlap tra le distribuzioni p_0 e p_1 può risultare basso e portare ad uno stimatore distorto per Z_1. Per ovviare a questo problema, è stato utilizzato il metodo di Annealed Importance Sampling, introducendo distribuzioni di probabilità intermedie $p_0, p_{\beta_1}, ..., p_{\beta_n}$ vicine tra loro, tali per cui $0 = \beta_0 < \beta_1 < ... < \beta_n = 1$. In questo caso risulta

$$\frac{Z_1}{Z_0} = \prod_{j=0}^{n-1} \frac{Z_{\beta_{j+1}}}{Z_{\beta_j}}$$

Estraendo 10^4 configurazioni dalla distribuzione a $\beta = 0$, sono stati implementati 10^3 passi di $\beta_j \in [0.001, 1]$. Per ogni distribuzione p_{β_j} sono stati estratte le configurazioni per Gibbs sampling dalla distribuzione di Bernoulli

$$P(h_i = 1|\mathbf{v}) = \sigma(\beta_j \mathbf{v}^T W_{:,i} + \beta_j c_i)$$
$$P(v_i = 1|\mathbf{h}) = \sigma(\beta_j W_{i,:}\, \mathbf{v} + \beta_j b_i)$$

Viene poi estratto un batch di 50 stati utilizzati per valutare

$$\log \frac{Z_{\beta_{j+1}}}{Z_{\beta_j}} = \log \frac{1}{M} \sum_{i=1}^{M} \frac{p^*_{\beta_{j+1}}}{p^*_j}$$

Ottenuta una stima per Z_{RBM}, è stata infine calcolata $\mathcal{L}(v|\theta)$ per un batch di $N = 10^4$ elementi del dataset:

$$\frac{1}{N} \log \mathcal{L}(v|\theta) = \frac{1}{N} \sum_{i=1}^{N} \log p^*(v^{(i)}|\theta) - \log Z_1$$

B Contrastive Divergence

Il metodo di Contrastive Divergence è il più utilizzato per allenare la Boltzmann Machines. Riporto da Goodfellow et al. - Deep Learning la procedura che ho utilizzato.

Algorithm 18.2 The contrastive divergence algorithm, using gradient ascent as the optimization procedure

Set ϵ, the step size, to a small positive number.
Set k, the number of Gibbs steps, high enough to allow a Markov chain sampling from $p(\mathbf{x};\boldsymbol{\theta})$ to mix when initialized from p_{data}. Perhaps 1–20 to train an RBM on a small image patch.
while not converged **do**
  Sample a minibatch of m examples $\{\mathbf{x}^{(1)},\dots,\mathbf{x}^{(m)}\}$ from the training set
  $\mathbf{g} \leftarrow \frac{1}{m}\sum_{i=1}^{m}\nabla_{\boldsymbol{\theta}}\log\tilde{p}(\mathbf{x}^{(i)};\boldsymbol{\theta})$.
  for $i = 1$ to m **do**
    $\tilde{\mathbf{x}}^{(i)} \leftarrow \mathbf{x}^{(i)}$.
  end for
  for $i = 1$ to k **do**
    for $j = 1$ to m **do**
      $\tilde{\mathbf{x}}^{(j)} \leftarrow \text{gibbs\_update}(\tilde{\mathbf{x}}^{(j)})$.
    end for
  end for
  $\mathbf{g} \leftarrow \mathbf{g} - \frac{1}{m}\sum_{i=1}^{m}\nabla_{\boldsymbol{\theta}}\log\tilde{p}(\tilde{\mathbf{x}}^{(i)};\boldsymbol{\theta})$.
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \epsilon\mathbf{g}$.
end while

References

[1] Guido Cossu, Luigi Del Debbio, Tommaso Giani, Ava Khamseh, and Michael Wilson. Machine learning determination of dynamical parameters: The ising model case. *Physical Review B*, 100(6), Aug 2019.