

# Methods

Here I have outlined the programs used and how to run them on the GPU machine available at TIFR.

## 1.1 Implementation

### 1.1.1 Ising Model

#### Monte Carlo Algorithm

To begin with study of samples generated from deep learning models, we require data to train them. This data is generated using the Wolff Cluster algorithm, for various lattice sizes. The following code implements it using parallel processing for faster implementation.

CODE

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 import os
5 from tqdm import tqdm
6 import multiprocessing as mp
7 from itertools import repeat
8
9 train_path = '../data/Ising/train/'
10 valid_path = '../data/Ising/valid/'
11
12 def makeArray(L):
13     array = np.random.randint(0,2,size=(L,L), dtype=int)
14     array[array==0] = -1
15     return array
16
17 def cluster_flip(array,cluster):# Flip all spins in a cluster
18     """
19     Parameters:
20         array: numpy array containing the current spin configuration
21         cluster: List of spins to be flipped
22     """
23     for pair in cluster:
24         i,j=pair
25         array[i,j]*=-1
```

```

26
27 def Wolff(array,T,calcE=False):
28     # Wolff algorithm for a square lattice
29     """
30     Parameters:
31         array: numpy array containing the current spin configuration
32         T: Temperature
33         calcE: Flag to calculate and return the average spin energy for
34             the final configuration.
35     """
36     L,_ = array.shape
37     cluster_i = np.zeros((L,L)) # spins already in cluster
38
39     i,j = np.random.randint(0,L,size=2)
40     spin = array[i,j]
41     stack = [(i,j)]
42     cluster_i[i,j]=1
43
44     cluster = [(i,j)]
45     while len(stack)>0:
46         i,j = stack.pop()
47         neighbors = [(i,(j+1)%L),(i,(j-1)%L),((i+1)%L,j),((i-1)%L,j)]
48         for pair in neighbors:
49             l,m = pair
50             if (array[l,m]==spin and cluster_i[l,m]==0 and np.random.
51                 random()< (1.0-np.exp(-2.0/T))):
52                 cluster.append((l,m))
53                 stack.append((l,m))
54                 cluster_i[l,m]=1
55
56     cluster_flip(array,cluster)
57
58     if calcE:
59         avgE=0.0
60         for i in range(L):
61             for j in range(L):
62                 spin_final = array[i,j]
63                 neighbor_sum = array[(i+1)%L,j]+array[(i-1)%L,j]+array[
64                     i,(j+1)%L]+array[i,(j-1)%L]
65                 E_final = -spin_final*neighbor_sum
66                 avgE+=float(E_final)
67
68         avgE/=float(L**2)
69         return avgE
70
71 def sim(array,T,L,steps,step_r,lattice):
72     #MC simulation for a square lattice, using the Wolff algorithm
73     for step in range(steps):
74         Wolff(array,T)
75         if step>step_r and step%divide==0:
76             i=int((step-step_r)/divide)
77             lattice[i]=array
78
79 def save_data(T,L,steps,step_r,lattice):
80     array=makeArray(L)
81     sim(array,T,L,steps,step_r,lattice)
82     lattice=np.reshape(lattice,(length,L*L))

```

```

80 np.savetxt("isingdata_"+str(L*L)+f"{T:.2f}"+".csv",lattice,delimiter=
    ",")
81
82 Temps=np.arange(1.5,3.2,0.1) #in Kelvin
83
84 L=25
85 step_r=200 #configurations to be removed
86 steps = 200000+step_r
87 divide=20 #autocorrelation tim
88 length=int((steps-step_r)/divide) #size of dataset
89 lattice=np.zeros((len(Temps),length,L,L))
90 #Parallel processing
91 jobs=zip(Temps,repeat(L),repeat(steps),repeat(step_r),lattice)
92 p = mp.Pool(5)
93 p.starmap(save_data, jobs)
94 p.close()
95 p.join()

```

Listing 1.1: isingdata25\_wolff.py

This program saves the data as "isingdata\_ $L^2$ T{2 digits}.csv" with  $L^2$  rows and  $steps - step\_r$  columns. The `mp.Pool()` function is used to implement parallel processing.

To check is the data is generated correctly OR to study the properties of Ising model for various lattice sizes and temperatures, we can plot certain parameters. Here we plot the magnetisation defined as  $M = \frac{1}{N} \times \sum_N \left\{ \frac{1}{L^2} \times |\sum_{L^2} s_i| \right\} = \frac{1}{N} \times \sum_N s$ , where  $s_i$  is the spin at  $i^{th}$  lattice position, and  $N = steps - step\_r$ .

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4
5 L=35
6 Size=3000
7
8 mag=np.zeros((5))
9
10 sample = np.genfromtxt('/user1/utkarsh/dataising350', delimiter = ',')
    #changing the file type to .csv format
11 np.savetxt("dataising350.csv",sample, delimiter = ',')
12 print(sample)
13 df2 = pd.read_csv('/user1/utkarsh/dataising350.csv') #reading the .csv
    file
14 data=df2.to_numpy()
15 np.savetxt('mag.csv',data)
16 print(data.shape)
17 m=np.sum(data,axis=1)
18 m=np.sum(m)
19 print(m)
20 mag[0]=(m/L**2)/Size
21
22 sample = np.genfromtxt('/user1/utkarsh/dataising352', delimiter = ',')
23 np.savetxt("dataising352.csv",sample, delimiter = ',')
24 df2 = pd.read_csv('/user1/utkarsh/dataising352.csv')
25 data=df2.to_numpy()
26 m=np.sum(data,axis=1)
27 m=np.sum(m)
28 mag[1]=(m/L**2)/Size
29

```

```

30 sample = np.genfromtxt('/user1/utkarsh/dataising354', delimiter = ',')
31 np.savetxt("dataising354.csv",sample, delimiter = ',')
32 df2 = pd.read_csv('/user1/utkarsh/dataising354.csv')
33 data=df2.to_numpy()
34 m=np.sum(data,axis=1)
35 m=np.sum(m)
36 mag[2]=(m/L**2)/Size
37
38 sample = np.genfromtxt('/user1/utkarsh/dataising356', delimiter = ',')
39 np.savetxt("dataising356.csv",sample, delimiter = ',')
40 df2 = pd.read_csv('/user1/utkarsh/dataising356.csv')
41 data=df2.to_numpy()
42 m=np.sum(data,axis=1)
43 m=np.sum(m)
44 mag[3]=(m/L**2)/Size
45
46 sample = np.genfromtxt('/user1/utkarsh/dataising358', delimiter = ',')
47 np.savetxt("dataising358.csv",sample, delimiter = ',')
48 df2 = pd.read_csv('/user1/utkarsh/dataising358.csv')
49 data=df2.to_numpy()
50 m=np.sum(data,axis=1)
51 m=np.sum(m)
52 mag[4]=(m/L**2)/Size
53
54 print(mag)
55 plt.plot(mag)
56 plt.savefig('mag350_mc.png')

```

Listing 1.2: magplot\_mc\_35.py

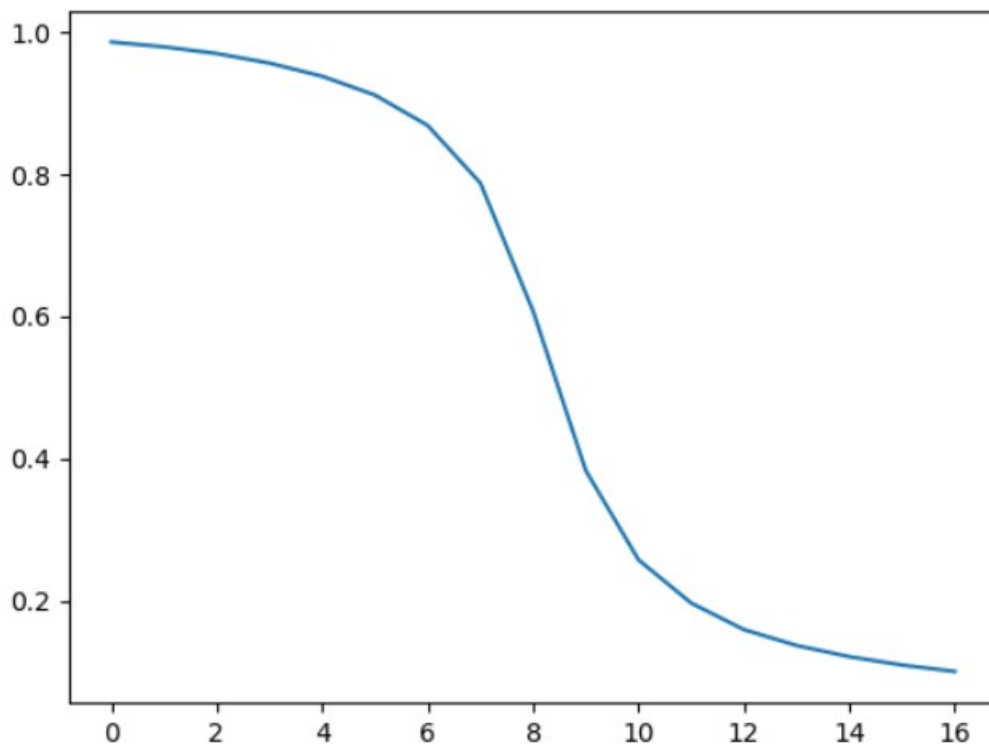


Figure 1.1: Magnetisation v/s Temperature index

The second parameter of interest is the Binder Cumulant given as  $U = 1 - \frac{\langle s^4 \rangle}{3 \langle s^2 \rangle^2}$ .

```

1
2 import pandas as pd
3 from matplotlib import pyplot as plt
4 import numpy as np
5
6 nbeta=5 # Number of data points
7 time=3000 #Number of samples per data point
8 L=35 #Lattice size
9
10 data=np.zeros((nbeta,time,d**2))
11 df2 = pd.read_csv('/user1/utkarsh/rbmdata350.csv',header=None) #keep
    the required file name
12 data[0]=df2.to_numpy()
13 df2 = pd.read_csv('/user1/utkarsh/rbmdata352.csv',header=None)
14 data[1]=df2.to_numpy()
15 df2 = pd.read_csv('/user1/utkarsh/rbmdata354.csv',header=None)
16 data[2]=df2.to_numpy()
17 df2 = pd.read_csv('/user1/utkarsh/rbmdata356.csv',header=None)
18 data[3]=df2.to_numpy()
19 df2 = pd.read_csv('/user1/utkarsh/rbmdata358.csv',header=None)
20 data[4]=df2.to_numpy()
21
22 mag_4_avg=np.zeros(nbeta) #Mag**4 average
23 mag_sq_avg=np.zeros(nbeta) #Mag**2 average
24 betas = np.linspace(0.43,0.45,5) #beta values
25
26
27 for i in range(nbeta): #Loop over all data points
28     mag=np.sum(data[i],axis=1)
29
30     for o in range(time): #Calculation for each data point
31         mag_4_avg[i]+=mag[o]*mag[o]*mag[o]*mag[o]
32         mag_sq_avg[i]+=mag[o]*mag[o]
33     print("sweep",i)
34     mag_4_avg[i]=mag_4_avg[i]/time
35     mag_sq_avg[i]=mag_sq_avg[i]/time
36 U1=1-(1/3)*(mag_4_avg/(mag_sq_avg*mag_sq_avg))
37
38
39 nbeta=5
40 time=3000
41 L=25
42
43 data=np.zeros((nbeta,time,d**2))
44 df2 = pd.read_csv('/user1/utkarsh/rbmdata250_.csv',header=None)
45 data[0]=df2.to_numpy()
46 df2 = pd.read_csv('/user1/utkarsh/rbmdata252_.csv',header=None)
47 data[1]=df2.to_numpy()
48 df2 = pd.read_csv('/user1/utkarsh/rbmdata254.csv',header=None)
49 data[2]=df2.to_numpy()
50 df2 = pd.read_csv('/user1/utkarsh/rbmdata256.csv',header=None)
51 data[3]=df2.to_numpy()
52 df2 = pd.read_csv('/user1/utkarsh/rbmdata258.csv',header=None)
53 data[4]=df2.to_numpy()
54
55 mag_4_avg=np.zeros(nbeta)

```

```

56 mag_sq_avg=np.zeros(nbeta)
57 betas = np.linspace(0.43,0.45,5)
58
59
60 for i in range(nbeta):
61     mag=np.sum(data[i],axis=1)
62
63     for o in range(time):
64         mag_4_avg[i]+=mag[o]*mag[o]*mag[o]*mag[o]
65         mag_sq_avg[i]+=mag[o]*mag[o]
66     print("sweep",i)
67     mag_4_avg[i]=mag_4_avg[i]/time
68     mag_sq_avg[i]=mag_sq_avg[i]/time
69 U2=1-(1/3)*(mag_4_avg/(mag_sq_avg*mag_sq_avg))
70
71 plt.plot(betas,U1/U2)
72 #plt.plot(betas,U2)
73 plt.savefig('UIratio.png')

```

Listing 1.3: BinderCumulant\_Ratio25\_35.py

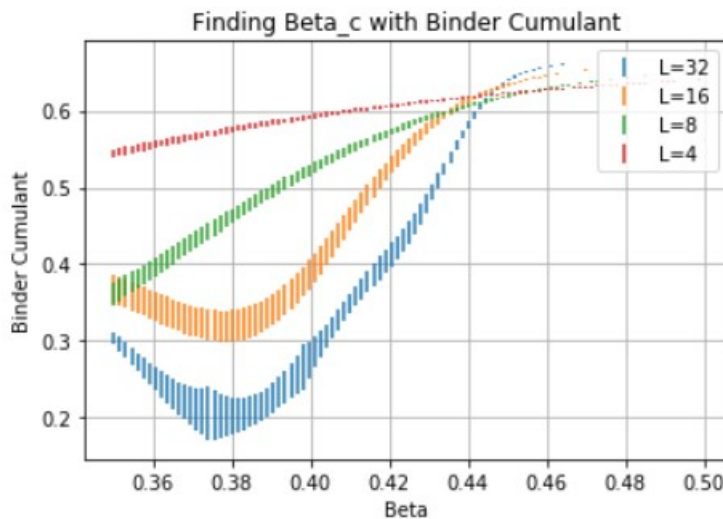


Figure 1.2: Binder Cumulant

Another parameter is  $\chi$  i.e., the susceptibility given by  $\chi = \langle \frac{s^2}{L^2} \rangle$ .

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 L=35 #Lattice size
5 mag=np.zeros((5))
6 chi=np.zeros((5))
7 sample = np.genfromtxt('/user1/utkarsh/dataising350', delimiter = ',')
8 #350 = lattice size=35, temperature index value = 0
9 np.savetxt("dataising350.csv",sample, delimiter = ',')
10 print(sample)
11 df2 = pd.read_csv('/user1/utkarsh/dataising350.csv')
12 data=df2.to_numpy()
13 np.savetxt('mag.csv',data)
14 print(data.shape)

```

```

14 m=np.sum(data,axis=1)
15 msq=m*m
16 m=np.sum(m)
17 msq=np.sum(msq)
18 print(m)
19 print(msq)
20 mag[0]=(m/L**2)/3000
21 chi[0]=(msq/L**4)/3000
22 sample = np.genfromtxt('/user1/utkarsh/dataising352', delimiter = ',')
23 np.savetxt("dataising352.csv",sample, delimiter = ',')
24 df2 = pd.read_csv('/user1/utkarsh/dataising352.csv')
25 data=df2.to_numpy()
26 m=np.sum(data,axis=1)
27 msq=m*m
28 m=np.sum(m)
29 msq=np.sum(msq)
30 print(m)
31 print(msq)
32 mag[1]=(m/L**2)/3000
33 chi[1]=(msq/L**4)/3000
34 sample = np.genfromtxt('/user1/utkarsh/dataising354', delimiter = ',')
35 np.savetxt("dataising354.csv",sample, delimiter = ',')
36 df2 = pd.read_csv('/user1/utkarsh/dataising354.csv')
37 data=df2.to_numpy()
38 m=np.sum(data,axis=1)
39 msq=m*m
40 m=np.sum(m)
41 msq=np.sum(msq)
42 print(m)
43 print(msq)
44 mag[2]=(m/L**2)/3000
45 chi[2]=(msq/L**4)/3000
46 sample = np.genfromtxt('/user1/utkarsh/dataising356', delimiter = ',')
47 np.savetxt("dataising356.csv",sample, delimiter = ',')
48 df2 = pd.read_csv('/user1/utkarsh/dataising356.csv')
49 data=df2.to_numpy()
50
51 m=np.sum(data,axis=1)
52 msq=m*m
53 m=np.sum(m)
54 msq=np.sum(msq)
55 print(m)
56 print(msq)
57 mag[3]=(m/L**2)/3000
58 chi[3]=(msq/L**4)/3000
59 sample = np.genfromtxt('/user1/utkarsh/dataising358', delimiter = ',')
60 np.savetxt("dataising358.csv",sample, delimiter = ',')
61 df2 = pd.read_csv('/user1/utkarsh/dataising358.csv')
62 data=df2.to_numpy()
63
64 m=np.sum(data,axis=1)
65 msq=m*m
66 m=np.sum(m)
67 msq=np.sum(msq)
68 print(m)
69 print(msq)
70 mag[4]=(m/L**2)/3000
71 chi[4]=(msq/L**4)/3000

```

```

72 print(mag)
73 print(chi)
74 plt.plot(chi)
75 plt.savefig('chi350_mc.png')

```

Listing 1.4: *chi\_plot\_mc.py*

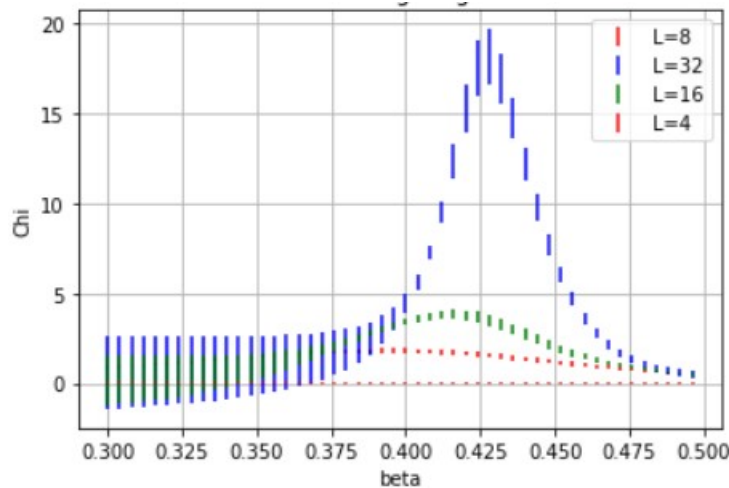


Figure 1.3:  $\chi v/s\beta$

## Restricted Boltzmann Machine

A Restricted Boltzmann Machine is trained with dataset generated from the Wolff Cluster Algorithm. We had flatten the dataset to a 2D matrix of shape  $[N, L^2]$  and feed it in the input nodes. The second layer size is a hyperparameter. The following program implements the training for one value of the temperature. The complete code can be found [here](#).

```

1 import numpy as np
2 import random
3 import numpy as np
4 import math
5 import matplotlib
6 import matplotlib.pyplot as plt
7 import pandas as pd
8 import os
9
10 epoch=25
11 size_h=1500
12 #return the average of h given x
13 def h_mean(x,w,b):
14     #x: visible layer. One row is one data
15     #w: weight, size_h*784
16     #b: bias, size_h
17
18     #return data*size_h
19     b_shaped=np.reshape(b,(1,b.size))
20     b_shaped=np.repeat(b_shaped,x.shape[0],axis=0)
21     y=np.dot(x,w.T)+b_shaped
22     return 1.0/(np.exp(-y)+1.0)

```



```

23
24 def x_mean(h,w,c):
25     #h: visible layer. One row is one data
26     #w: weight, size_h*784
27     #c: bias, size_h
28
29     c_shaped=np.reshape(c,(1,c.size))
30     c_shaped=np.repeat(c_shaped,h.shape[0],axis=0)
31     y=np.dot(h,w)+c_shaped
32     return np.tanh(y),1/(np.exp(-1.0*y)+1.0)
33
34 def gibbs_cd(x,kcd,w,b,c):
35     h_p=h_mean(x,w,b)
36     rad_h=np.random.rand(h_p.shape[0],h_p.shape[1])
37     h_samp=np.zeros(h_p.shape)
38     h_samp[rad_h<h_p]=1
39     for i in range(kcd):
40         #sample x
41         _, x_prob=x_mean(h_samp,w,c)
42         rad_x=np.random.rand(x_prob.shape[0],x_prob.shape[1])
43         x_samp=-np.ones(x_prob.shape)
44         x_samp[rad_x<x_prob]=1
45         #sample h
46         h_p=h_mean(x_samp,w,b)
47         rad_h=np.random.rand(h_p.shape[0],h_p.shape[1])
48         h_samp=np.zeros(h_p.shape)
49         h_samp[rad_h<h_p]=1
50     return x_samp
51
52 def loss(x_in,x_p):
53     l=(x_in-x_p)**2
54     loss=np.sum(l)/x_in.size
55     return loss
56
57 #####
58
59 df2 = pd.read_csv('/user1/utkarsh/isingdata_6251.50.csv')
60 data=df2.to_numpy()
61
62 trX=data[0:8000]
63 test_data=data[8000:10000]
64
65 spin_flatten=trX
66
67 #learning rate
68 gamma=0.005
69 #momentum
70 beta=0.0
71 #l1 regularization
72 lambd=0
73
74 #number of hidden units
75
76 size_v=spin_flatten.shape[1]
77 w=np.reshape(np.random.normal(scale=0.1,size=size_v*size_h),(size_h,
78     size_v))
79 b=np.zeros(size_h)
80 c=np.zeros(size_v)

```

```

80
81 batch=100 #batch size
82 kcd=20    #cd number
83
84
85 train_loss=np.zeros(epoch)
86 grad_w0=np.zeros(w.shape)
87 grad_b0=np.zeros(b.shape)
88 grad_c0=np.zeros(c.shape)
89
90 print('start')
91
92 for i in range(epoch):
93     spin_train=np.random.permutation(spin_flatten)
94     for j in range(math.floor(spin_train.shape[0]/batch)):
95         x_in=spin_train[j*batch:j*batch+batch,:]
96
97         x_sampling= gibbs_cd(x_in,kcd,w,b,c)
98         #gradient update
99         grad_w=np.zeros(w.shape)
100         for k in range(batch):
101             x_k=np.reshape(x_in[k,:],(1,x_in[k,:].size))
102             x_ksample=np.reshape(x_sampling[k,:],(1,x_sampling[k,:].
size))
103             grad_w=grad_w+(np.outer( h_mean(x_k,w,b),x_k)-np.outer(
h_mean(x_ksample,w,b),x_ksample))
104
105
106             w=w+gamma*(grad_w-lambd*np.sign(w))
107             b=b+gamma*np.sum( h_mean(x_in,w,b)- h_mean(x_sampling,w,b),axis
=0)
108
109
110             c=c+gamma*np.sum(x_in-x_sampling,axis=0)
111
112         print('epoch'+str(i))
113
114         #training reconstruction
115         h_p= h_mean(spin_train,w,b)
116         x_p,_= x_mean(h_p,w,c)
117         train_loss[i]= loss(spin_train,x_p)
118         print('train'+str(train_loss[i]))
119
120
121 fnamew='w_ising'+ '6251.50.csv'
122 np.savetxt(fnamew,w, delimiter=",")
123 fnameb='b_ising'+ '6251.50.csv'
124 np.savetxt(fnameb,b, delimiter=",")
125 fnamec='c_ising'+ '6251.50.csv'
126 np.savetxt(fnamec,c, delimiter=",")

```

*Listing 1.5: rbmising25.py*

Once the model is trained with the data for a given lattice size and temperature, we can generate data from the models with the following code.

```

1 import pandas as pd
2
3 #return the average of h given x

```

```

4 def h_mean(x,w,b):
5     #x: visible layer. One row is one data
6     #w: weight, size_h*784
7     #b: bias, size_h
8
9     #return data*size_h
10    b_shaped=np.reshape(b,(1,b.size))
11    b_shaped=np.repeat(b_shaped,x.shape[0],axis=0)
12    y=np.dot(x,w.T)+b_shaped
13    return 1.0/(np.exp(-y)+1.0)
14
15 def x_mean(h,w,c):
16     #h: visible layer. One row is one data
17     #w: weight, size_h*784
18     #c: bias, size_h
19
20    c_shaped=np.reshape(c,(1,c.size))
21    c_shaped=np.repeat(c_shaped,h.shape[0],axis=0)
22    y=np.dot(h,w)+c_shaped
23    return np.tanh(y),1/(np.exp(-1.0*y)+1.0)
24
25 def gibbs_cd(x,kcd,w,b,c):
26     h_p=h_mean(x,w,b)
27     rad_h=np.random.rand(h_p.shape[0],h_p.shape[1])
28     h_samp=np.zeros(h_p.shape)
29     h_samp[rad_h<h_p]=1
30     for i in range(kcd):
31         #sample x
32         _, x_prob=x_mean(h_samp,w,c)
33         rad_x=np.random.rand(x_prob.shape[0],x_prob.shape[1])
34         x_samp=-np.ones(x_prob.shape)
35         x_samp[rad_x<x_prob]=1
36         #sample h
37         h_p=h_mean(x_samp,w,b)
38         rad_h=np.random.rand(h_p.shape[0],h_p.shape[1])
39         h_samp=np.zeros(h_p.shape)
40         h_samp[rad_h<h_p]=1
41     return x_samp
42
43 def loss(x_in,x_p):
44     l=(x_in-x_p)**2
45     loss=np.sum(l)/x_in.size
46     return loss
47
48 #sampling from the rbm
49 import numpy as np
50 import math
51 import matplotlib
52 import matplotlib.pyplot as plt
53 #import rbm_xy_fun as rbm_fun
54 import os
55 def generate(n,w,b,c,x_data):
56
57     for i in range(n):
58         rad_x=np.random.rand(1,c.size)
59         x_sampling=gibbs_cd(rad_x,kcd,w,b,c)
60         h_p=h_mean(x_sampling,w,b)
61         x_p,x_dat=x_mean(h_p,w,c)

```

```

62     rad_x=np.random.rand(x_dat.shape[0],x_dat.shape[1])
63     x_samp=-np.ones(x_dat.shape)
64     x_samp[rad_x<x_p]=1
65     x_shaped=np.reshape(x_samp,(d,d))
66     x_data[i]=x_shaped
67
68 d=16
69 n=1000
70 kcd=100
71 sigma=1
72 m=np.zeros(17)
73 msq=np.zeros(17)
74 #
75     #####
76 df = pd.read_csv('/user1/utkarsh/b_ising2561.50.csv', header=None)
77 b=df.to_numpy()
78 df = pd.read_csv('/user1/utkarsh/c_ising2561.50.csv', header=None)
79 c=df.to_numpy()
80 df = pd.read_csv('/user1/utkarsh/w_ising2561.50.csv', header=None)
81 w=df.to_numpy()
82
83 x_data=np.zeros((n,d,d))
84 generate(n,w,b,c,x_data)
85
86 x_data=np.reshape(x_data,(n,d**2))
87
88 mag=np.sum(x_data, axis=1)
89 ms=mag*mag
90 msq[0]=np.sum(ms)
91 m[0]=np.sum(abs(mag))
92
93 np.savetxt("rbmdata2561.5.csv", x_data, delimiter=",")
94 #
95     #####
96 df = pd.read_csv('/user1/utkarsh/b_ising2561.60.csv', header=None)
97 b=df.to_numpy()
98 df = pd.read_csv('/user1/utkarsh/c_ising2561.60.csv', header=None)
99 c=df.to_numpy()
100 df = pd.read_csv('/user1/utkarsh/w_ising2561.60.csv', header=None)
101 w=df.to_numpy()
102
103 x_data=np.zeros((n,d,d))
104 generate(n,w,b,c,x_data)
105
106 x_data=np.reshape(x_data,(n,d**2))
107
108 mag=np.sum(x_data, axis=1)
109 ms=mag*mag
110 msq[1]=np.sum(ms)
111 m[1]=np.sum(abs(mag))
112
113 np.savetxt("rbmdata2561.6.csv", x_data, delimiter=",")
114 #
115     #####

```

```

115
116 df = pd.read_csv('/user1/utkarsh/b_ising2561.70.csv', header=None)
117 b=df.to_numpy()
118 df = pd.read_csv('/user1/utkarsh/c_ising2561.70.csv', header=None)
119 c=df.to_numpy()
120 df = pd.read_csv('/user1/utkarsh/w_ising2561.70.csv', header=None)
121 w=df.to_numpy()
122
123 x_data=np.zeros((n,d,d))
124 generate(n,w,b,c,x_data)
125
126 x_data=np.reshape(x_data,(n,d**2))
127
128 mag=np.sum(x_data,axis=1)
129 ms=mag*mag
130 msq[2]=np.sum(ms)
131 m[2]=np.sum(abs(mag))
132
133 np.savetxt("rbmdata2561.7.csv", x_data, delimiter=",")
134 #
135 #####
136
136 df = pd.read_csv('/user1/utkarsh/b_ising2561.80.csv', header=None)
137 b=df.to_numpy()
138 df = pd.read_csv('/user1/utkarsh/c_ising2561.80.csv', header=None)
139 c=df.to_numpy()
140 df = pd.read_csv('/user1/utkarsh/w_ising2561.80.csv', header=None)
141 w=df.to_numpy()
142
143 x_data=np.zeros((n,d,d))
144 generate(n,w,b,c,x_data)
145
146 x_data=np.reshape(x_data,(n,d**2))
147
148 mag=np.sum(x_data,axis=1)
149 ms=mag*mag
150 msq[3]=np.sum(ms)
151 m[3]=np.sum(abs(mag))
152
153 np.savetxt("rbmdata2561.8.csv", x_data, delimiter=",")
154 #
155 #####
156
156 df = pd.read_csv('/user1/utkarsh/b_ising2561.90.csv', header=None)
157 b=df.to_numpy()
158 df = pd.read_csv('/user1/utkarsh/c_ising2561.90.csv', header=None)
159 c=df.to_numpy()
160 df = pd.read_csv('/user1/utkarsh/w_ising2561.90.csv', header=None)
161 w=df.to_numpy()
162
163 x_data=np.zeros((n,d,d))
164 generate(n,w,b,c,x_data)
165
166 x_data=np.reshape(x_data,(n,d**2))
167

```

```

168 mag=np.sum(x_data, axis=1)
169 ms=mag*mag
170 msq[4]=np.sum(ms)
171 m[4]=np.sum(abs(mag))
172
173 np.savetxt("rbmdata2561.9.csv", x_data, delimiter=",")
174 #
175     #####
176
177 df = pd.read_csv('/user1/utkarsh/b_ising2562.00.csv', header=None)
178 b=df.to_numpy()
179 df = pd.read_csv('/user1/utkarsh/c_ising2562.00.csv', header=None)
180 c=df.to_numpy()
181 df = pd.read_csv('/user1/utkarsh/w_ising2562.00.csv', header=None)
182 w=df.to_numpy()
183
184 x_data=np.zeros((n,d,d))
185 generate(n,w,b,c,x_data)
186
187 x_data=np.reshape(x_data,(n,d**2))
188
189 mag=np.sum(x_data, axis=1)
190 ms=mag*mag
191 msq[5]=np.sum(ms)
192 m[5]=np.sum(abs(mag))
193
194 np.savetxt("rbmdata2562.0.csv", x_data, delimiter=",")
195 #
196     #####
197
198 df = pd.read_csv('/user1/utkarsh/b_ising2562.10.csv', header=None)
199 b=df.to_numpy()
200 df = pd.read_csv('/user1/utkarsh/c_ising2562.10.csv', header=None)
201 c=df.to_numpy()
202 df = pd.read_csv('/user1/utkarsh/w_ising2562.10.csv', header=None)
203 w=df.to_numpy()
204
205 x_data=np.zeros((n,d,d))
206 generate(n,w,b,c,x_data)
207
208 x_data=np.reshape(x_data,(n,d**2))
209
210 mag=np.sum(x_data, axis=1)
211 ms=mag*mag
212 msq[6]=np.sum(ms)
213 m[6]=np.sum(abs(mag))
214
215 np.savetxt("rbmdata2562.1.csv", x_data, delimiter=",")
216 #
217     #####
218
219 df = pd.read_csv('/user1/utkarsh/b_ising2562.20.csv', header=None)
220 b=df.to_numpy()
221 df = pd.read_csv('/user1/utkarsh/c_ising2562.20.csv', header=None)
222 c=df.to_numpy()

```

```

220 df = pd.read_csv('/user1/utkarsh/w_ising2562.20.csv', header=None)
221 w=df.to_numpy()
222
223 x_data=np.zeros((n,d,d))
224 generate(n,w,b,c,x_data)
225
226 x_data=np.reshape(x_data,(n,d**2))
227
228 mag=np.sum(x_data, axis=1)
229 ms=mag*mag
230 msq[7]=np.sum(ms)
231 m[7]=np.sum(abs(mag))
232
233 np.savetxt("rbmdata2562.2.csv", x_data, delimiter=",")
234 #
235 #####
236
236 df = pd.read_csv('/user1/utkarsh/b_ising2562.30.csv', header=None)
237 b=df.to_numpy()
238 df = pd.read_csv('/user1/utkarsh/c_ising2562.30.csv', header=None)
239 c=df.to_numpy()
240 df = pd.read_csv('/user1/utkarsh/w_ising2562.30.csv', header=None)
241 w=df.to_numpy()
242
243 x_data=np.zeros((n,d,d))
244 generate(n,w,b,c,x_data)
245
246 x_data=np.reshape(x_data,(n,d**2))
247
248 mag=np.sum(x_data, axis=1)
249 ms=mag*mag
250 msq[8]=np.sum(ms)
251 m[8]=np.sum(abs(mag))
252
253 np.savetxt("rbmdata2562.3.csv", x_data, delimiter=",")
254 #
255 #####
256
256 df = pd.read_csv('/user1/utkarsh/b_ising2562.40.csv', header=None)
257 b=df.to_numpy()
258 df = pd.read_csv('/user1/utkarsh/c_ising2562.40.csv', header=None)
259 c=df.to_numpy()
260 df = pd.read_csv('/user1/utkarsh/w_ising2562.40.csv', header=None)
261 w=df.to_numpy()
262
263 x_data=np.zeros((n,d,d))
264 generate(n,w,b,c,x_data)
265
266 x_data=np.reshape(x_data,(n,d**2))
267
268 mag=np.sum(x_data, axis=1)
269 ms=mag*mag
270 msq[9]=np.sum(ms)
271 m[9]=np.sum(abs(mag))
272
273 np.savetxt("rbmdata2562.4.csv", x_data, delimiter=",")

```

```

274 #
275 #####
276 df = pd.read_csv('/user1/utkarsh/b_ising2562.50.csv', header=None)
277 b=df.to_numpy()
278 df = pd.read_csv('/user1/utkarsh/c_ising2562.50.csv', header=None)
279 c=df.to_numpy()
280 df = pd.read_csv('/user1/utkarsh/w_ising2562.50.csv', header=None)
281 w=df.to_numpy()
282
283 x_data=np.zeros((n,d,d))
284 generate(n,w,b,c,x_data)
285
286 x_data=np.reshape(x_data,(n,d**2))
287
288 mag=np.sum(x_data, axis=1)
289 ms=mag*mag
290 msq[10]=np.sum(ms)
291 m[10]=np.sum(abs(mag))
292
293 np.savetxt("rbmdata2562.5.csv", x_data, delimiter=",")
294 #
295 #####
296 df = pd.read_csv('/user1/utkarsh/b_ising2562.60.csv', header=None)
297 b=df.to_numpy()
298 df = pd.read_csv('/user1/utkarsh/c_ising2562.60.csv', header=None)
299 c=df.to_numpy()
300 df = pd.read_csv('/user1/utkarsh/w_ising2562.60.csv', header=None)
301 w=df.to_numpy()
302
303 x_data=np.zeros((n,d,d))
304 generate(n,w,b,c,x_data)
305
306 x_data=np.reshape(x_data,(n,d**2))
307
308 mag=np.sum(x_data, axis=1)
309 ms=mag*mag
310 msq[11]=np.sum(ms)
311 m[11]=np.sum(abs(mag))
312
313 np.savetxt("rbmdata2562.6.csv", x_data, delimiter=",")
314 #
315 #####
316 df = pd.read_csv('/user1/utkarsh/b_ising2562.70.csv', header=None)
317 b=df.to_numpy()
318 df = pd.read_csv('/user1/utkarsh/c_ising2562.70.csv', header=None)
319 c=df.to_numpy()
320 df = pd.read_csv('/user1/utkarsh/w_ising2562.70.csv', header=None)
321 w=df.to_numpy()
322
323 x_data=np.zeros((n,d,d))
324 generate(n,w,b,c,x_data)
325

```



```

326 x_data=np.reshape(x_data,(n,d**2))
327
328 mag=np.sum(x_data,axis=1)
329 ms=mag*mag
330 msq[12]=np.sum(ms)
331 m[12]=np.sum(abs(mag))
332
333 np.savetxt("rbmdata2562.7.csv", x_data, delimiter=",")
334 #
335 #####
336
337 df = pd.read_csv('/user1/utkarsh/b_ising2562.80.csv', header=None)
338 b=df.to_numpy()
339 df = pd.read_csv('/user1/utkarsh/c_ising2562.80.csv', header=None)
340 c=df.to_numpy()
341 df = pd.read_csv('/user1/utkarsh/w_ising2562.80.csv', header=None)
342 w=df.to_numpy()
343
344 x_data=np.zeros((n,d,d))
345 generate(n,w,b,c,x_data)
346
347 x_data=np.reshape(x_data,(n,d**2))
348
349 mag=np.sum(x_data,axis=1)
350 ms=mag*mag
351 msq[13]=np.sum(ms)
352 m[13]=np.sum(abs(mag))
353
354 np.savetxt("rbmdata2562.8.csv", x_data, delimiter=",")
355 #
356 #####
357
358 df = pd.read_csv('/user1/utkarsh/b_ising2562.90.csv', header=None)
359 b=df.to_numpy()
360 df = pd.read_csv('/user1/utkarsh/c_ising2562.90.csv', header=None)
361 c=df.to_numpy()
362 df = pd.read_csv('/user1/utkarsh/w_ising2562.90.csv', header=None)
363 w=df.to_numpy()
364
365 x_data=np.zeros((n,d,d))
366 generate(n,w,b,c,x_data)
367
368 x_data=np.reshape(x_data,(n,d**2))
369
370 mag=np.sum(x_data,axis=1)
371 ms=mag*mag
372 msq[14]=np.sum(ms)
373 m[14]=np.sum(abs(mag))
374
375 np.savetxt("rbmdata2562.9.csv", x_data, delimiter=",")
376 #
377 #####
378
379 df = pd.read_csv('/user1/utkarsh/b_ising2563.00.csv', header=None)
380 b=df.to_numpy()

```

```

378 df = pd.read_csv('/user1/utkarsh/c_ising2563.00.csv', header=None)
379 c=df.to_numpy()
380 df = pd.read_csv('/user1/utkarsh/w_ising2563.00.csv', header=None)
381 w=df.to_numpy()
382
383 x_data=np.zeros((n,d,d))
384 generate(n,w,b,c,x_data)
385
386 x_data=np.reshape(x_data,(n,d**2))
387
388 mag=np.sum(x_data, axis=1)
389 ms=mag*mag
390 msq[15]=np.sum(ms)
391 m[15]=np.sum(abs(mag))
392
393 np.savetxt("rbmdata2563.0.csv", x_data, delimiter=",")
394 #
395 #####
396 df = pd.read_csv('/user1/utkarsh/b_ising2563.10.csv', header=None)
397 b=df.to_numpy()
398 df = pd.read_csv('/user1/utkarsh/c_ising2563.10.csv', header=None)
399 c=df.to_numpy()
400 df = pd.read_csv('/user1/utkarsh/w_ising2563.10.csv', header=None)
401 w=df.to_numpy()
402
403 x_data=np.zeros((n,d,d))
404 generate(n,w,b,c,x_data)
405
406 x_data=np.reshape(x_data,(n,d**2))
407
408 mag=np.sum(x_data, axis=1)
409 ms=mag*mag
410 msq[16]=np.sum(ms)
411 m[16]=np.sum(abs(mag))
412
413
414 np.savetxt("rbmdata2563.1.csv", x_data, delimiter=",")
415
416 #
417 #####
418
417 m=m/d**2
418 m=m/n
419
420 chi=(msq/d**4)/n
421
422 err=np.sqrt(abs(chi-m**2))
423 x=np.arange(1.5,3.2,0.1)
424
425 plt.errorbar(x,m,err)
426
427 plt.savefig("magrbm16_err_new.png")

```

Listing 1.6: rbmgenerate\_err.py

The generated datafile is now used to reproduce the plots for magnetisation, magnetic susceptibility and Binder Cumulant. The following plots present these results.

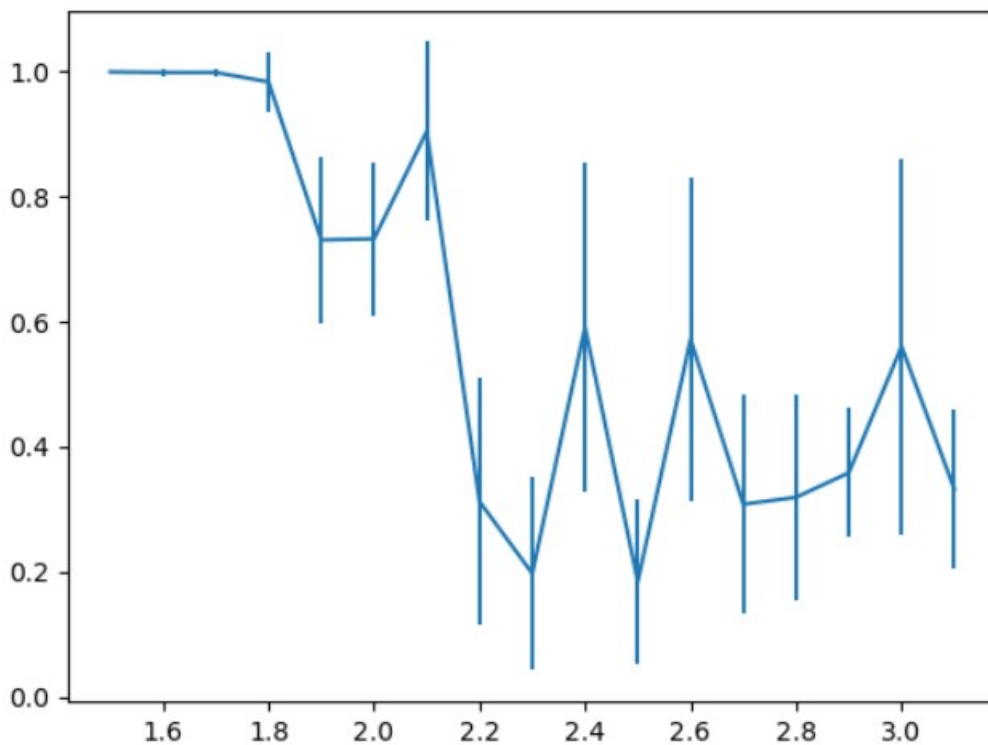


Figure 1.4:  $Mv/sT(K)$ ,  $L=8$

## GPU Programs

### 1.1.2 XY Model

#### Monte Carlo Algorithm

Here is the code for the Monte Carlo algorithm to generate uncorrelated samples for the classical XY model.

```

1 # -*- coding: utf-8 -*-
2 """dataxy32.ipynb
3
4 Automatically generated by Colaboratory.
5
6 Original file is located at
7     https://colab.research.google.com/drive/1
8     IE1vmGDRo25BgQcJ2zu0SWi6ERlB6a-h
9 """
10 import matplotlib
11 matplotlib.use('Agg')
12 import numpy as np
13 from numpy import linalg as LA
14 import matplotlib.pyplot as plt
15
16 L = 32
17 step_r = 3000

```

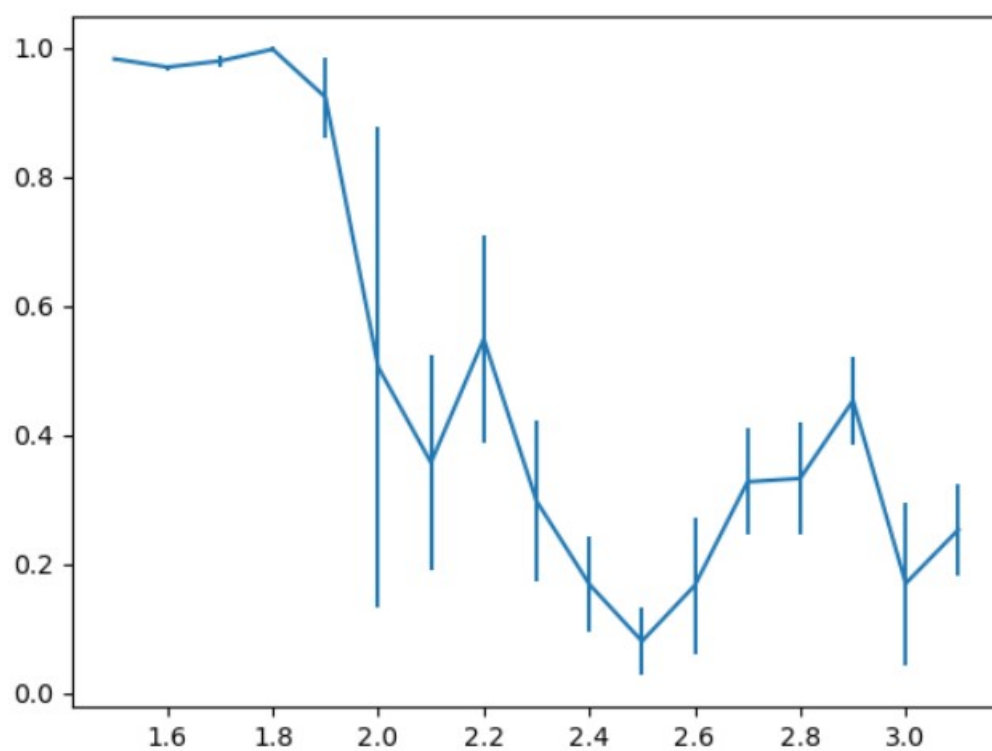


Figure 1.5:  $Mv/sT(K)$ ,  $L=16$

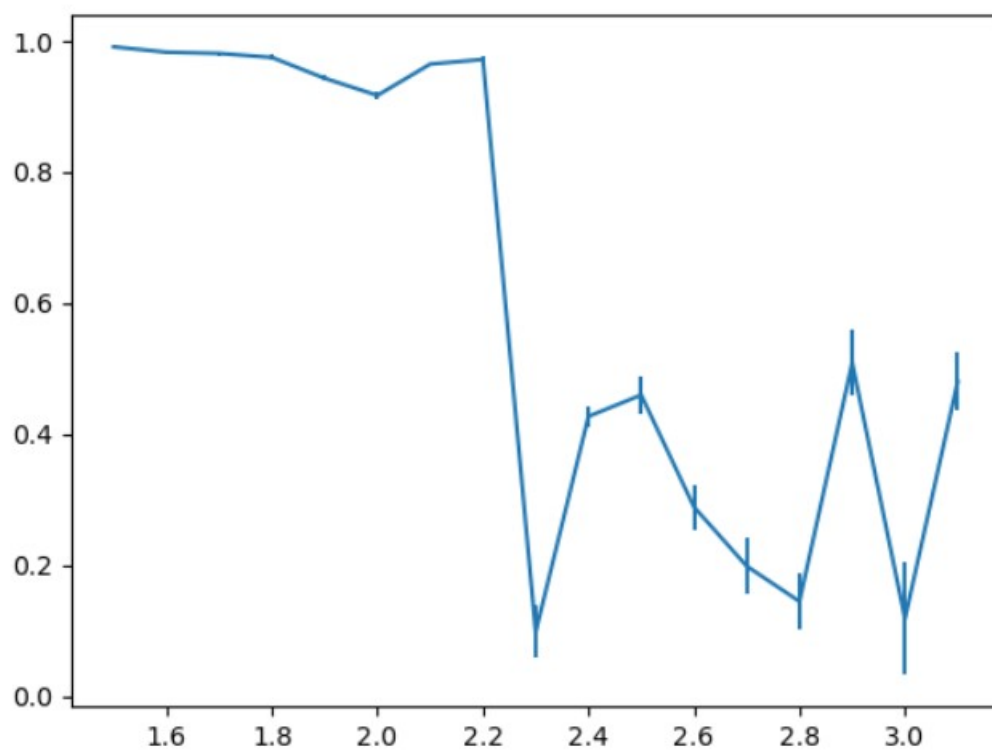
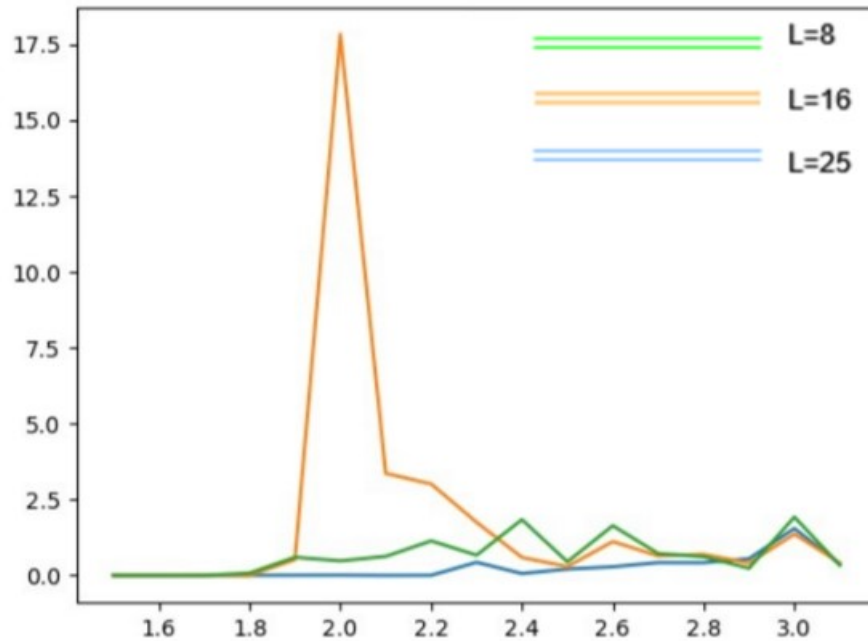


Figure 1.6:  $Mv/sT(K)$ ,  $L=25$

Figure 1.7:  $\chi v / sT(K)$ 

```

18 step = 750000
19 nt=10
20 n_remove=30
21 J = 1 # J>0 to make it ferromagnetic
22
23
24 # Intitalize the XY network
25 def init():
26     return np.random.rand(L, L)*2*np.pi
27     #return np.ones([L, L])
28
29 # periodic neighbor
30 def next(x):
31     if x == L-1:
32         return 0
33     else:
34         return x+1
35
36 # construct the bond lattice
37 def FreezeBonds(Ising,T,S):
38     iBondFrozen = np.zeros([L,L])
39     jBondFrozen = np.zeros([L,L])
40     for i in np.arange(L):
41         for j in np.arange(L):
42             freezProb_nexti = 1 - np.exp(-2 * J * S[i][j] * S[next(i)][
j] / T)
43             freezProb_nextj = 1 - np.exp(-2 * J * S[i][j] * S[i][next(j
)]) / T)
44             if (Ising[i][j] == Ising[next(i)][j]) and (np.random.rand()
< freezProb_nexti):
45                 iBondFrozen[i][j] = 1

```

```

46         if (Ising[i][j] == Ising[i][next(j)]) and (np.random.rand()
47             < freezProb_nextj):
48             jBondFrozen[i][j] = 1
49         return iBondFrozen, jBondFrozen
50 # H-K algorithm to identify clusters
51 def properlabel(prp_label,i):
52     while prp_label[int(i)] != i:
53         i = prp_label[int(i)]
54     return i
55
56 # Swendsen-Wang cluster
57 def clusterfind(iBondFrozen,jBondFrozen):
58     cluster = np.zeros([L, L])
59     prp_label = np.zeros(L**2)
60     label = 0
61     for i in np.arange(L):
62         for j in np.arange(L):
63             bonds = 0
64             ibonds = np.zeros(4)
65             jbonds = np.zeros(4)
66
67             # check to (i-1,j)
68             if (i > 0) and iBondFrozen[i-1][j]:
69                 ibonds[bonds] = i-1
70                 jbonds[bonds] = j
71                 bonds += 1
72             # (i,j) at i edge, check to (i+1,j)
73             if (i == L-1) and iBondFrozen[i][j]:
74                 ibonds[bonds] = 0
75                 jbonds[bonds] = j
76                 bonds += 1
77             # check to (i,j-1)
78             if (j > 0) and jBondFrozen[i][j-1]:
79                 ibonds[bonds] = i
80                 jbonds[bonds] = j-1
81                 bonds += 1
82             # (i,j) at j edge, check to (i,j+1)
83             if (j == L-1) and jBondFrozen[i][j]:
84                 ibonds[bonds] = i
85                 jbonds[bonds] = 0
86                 bonds += 1
87
88             # check and label clusters
89             if bonds == 0:
90                 cluster[i][j] = label
91                 prp_label[label] = label
92                 label += 1
93             else:
94                 minlabel = label
95                 for b in np.arange(bonds):
96                     plabel = properlabel(prp_label,cluster[int(ibonds[b
97 ])[int(jbonds[b])])
98                     if minlabel > plabel:
99                         minlabel = plabel
100
101                 cluster[i][j] = minlabel
102             # link to the previous labels

```

```

102         for b in np.arange(bonds):
103             plabel_n = cluster[int(ibonds[b])][int(jbonds[b])]
104             prp_label[int(plabel_n)] = minlabel
105             # re-set the labels on connected sites
106             cluster[int(ibonds[b])][int(jbonds[b])] = minlabel
107     return cluster, prp_label
108
109 # flip the cluster spins
110 def flipCluster(Ising, cluster, prp_label):
111     for i in np.arange(L):
112         for j in np.arange(L):
113             # relabel all the cluster labels with the right ones
114             cluster[i][j] = properlabel(prp_label, cluster[i][j])
115     sNewChosen = np.zeros(L**2)
116     sNew = np.zeros(L**2)
117     flips = 0 # get the number of flipped spins to calculate the Endiff
118               # and Magdiff
119     for i in np.arange(L):
120         for j in np.arange(L):
121             label = cluster[i][j]
122             randn = np.random.rand()
123             # mark the flipped label, use this label to flip all the
124             # cluster elements with this label
125             if (not sNewChosen[int(label)]) and randn < 0.5:
126                 sNew[int(label)] = +1
127                 sNewChosen[int(label)] = True
128             elif (not sNewChosen[int(label)]) and randn >= 0.5:
129                 sNew[int(label)] = -1
130                 sNewChosen[int(label)] = True
131
132             if Ising[i][j] != sNew[int(label)]:
133                 Ising[i][j] = sNew[int(label)]
134                 flips += 1
135
136     return Ising, flips
137
138 # Swendsen-Wang Algorithm in Ising model (with coupling constant
139 # dependency on sites)
140 # One-step for Ising
141 def oneMCstepIsing(Ising, T, S):
142     [iBondFrozen, jBondFrozen] = FreezeBonds(Ising, T, S)
143     [SWcluster, prp_label] = clusterfind(iBondFrozen, jBondFrozen)
144     [Ising, flips] = flipCluster(Ising, SWcluster, prp_label)
145     return Ising
146
147 # Decompose XY network to two Ising networks with project direction
148 # proj
149 def decompose(XY, proj):
150     x = np.cos(XY)
151     y = np.sin(XY)
152     x_rot = np.multiply(x, np.cos(proj)) + np.multiply(y, np.sin(proj))
153     y_rot = -np.multiply(x, np.sin(proj)) + np.multiply(y, np.cos(proj))
154     Isingx = np.sign(x_rot)
155     Isingy = np.sign(y_rot)
156     S_x = np.absolute(x_rot)
157     S_y = np.absolute(y_rot)
158     return Isingx, Isingy, S_x, S_y

```

```

156 # Compose two Ising networks to XY network
157 def compose(Isingx_new, Isingy_new, proj, S_x, S_y):
158     x_rot_new = np.multiply(Isingx_new, S_x)
159     y_rot_new = np.multiply(Isingy_new, S_y)
160     x_new = np.multiply(x_rot_new, np.cos(proj)) - np.multiply(y_rot_new,
161     np.sin(proj))
162     y_new = np.multiply(x_rot_new, np.sin(proj)) + np.multiply(y_rot_new,
163     np.cos(proj))
164     XY_new = np.arctan2(y_new, x_new)
165     return XY_new
166
167 def oneMCstepXY(XY):
168     proj = np.random.rand()
169     [Isingx, Isingy, S_x, S_y] = decompose(XY, proj)
170     Isingx_new = oneMCstepIsing(Isingx, S_x)
171     Isingy_new = oneMCstepIsing(Isingy, S_y)
172     XY_new = compose(Isingx_new, Isingy_new, proj, S_x, S_y)
173     return XY_new
174
175 # Calculate the energy for XY network
176 def EnMag(XY):
177     energy = 0
178     for i in np.arange(L):
179         for j in np.arange(L):
180             # energy
181             energy = energy - (np.cos(XY[i][j] - XY[(i-1)%L][j]) + np.cos(
182             XY[i][j] - XY[(i+1)%L][j]) + np.cos(XY[i][j] - XY[i][(j-1)%L]) + np.cos(XY[i]
183             [j] - XY[i][(j+1)%L]))
184     magx = np.sum(np.cos(XY))
185     magy = np.sum(np.sin(XY))
186     mag = np.array([magx, magy])
187     return energy * 0.5, LA.norm(mag)/(L**2)
188
189 # Swendsen Wang method for XY model
190 def SWang(T):
191     XY = init()
192     # thermal steps to get the equilibrium
193     for step in np.arange(step_r):
194         XY = oneMCstepXY(XY)
195     # finish with thermal equilibrium, and begin to calc observables
196     E_sum = 0
197     M_sum = 0
198     Esq_sum = 0
199     Msq_sum = 0
200     lattice_data = np.zeros((step, L, L))
201     mag = []
202     for step in np.arange(step):
203         XY = oneMCstepXY(XY)
204         [E, M] = EnMag(XY)
205         mag = np.append(mag, M)
206         lattice_data[step] = XY
207         E_sum += E
208         M_sum += M
209         Esq_sum += E**2
210         Msq_sum += M**2
211
212     E_mean = E_sum/step/(L**2)
213     M_mean = M_sum/step

```



```

210     Esq_mean = Esq_sum/step/(L**4)
211     Msq_mean = Msq_sum/step
212
213     return lattice_data, XY, E_mean, M_mean, Esq_mean, Msq_mean, mag;
214
215 M = np.array([])
216 E = np.array([])
217 M_sus = np.array([])
218 SpcH = np.array([])
219
220 Trange = np.linspace(0.75, 1.25, nt)
221 Lattice_XY=np.zeros((step*nt,L,L))
222 Mag=[]
223 for t in range(Trange.shape[0]):
224     T=Trange[t]
225     lattice_data, Ising, E_mean, M_mean, Esq_mean, Msq_mean, mag =
226     SWang(T)
227     M = np.append(M, np.abs(M_mean))
228     E = np.append(E, E_mean)
229     Mag=np.append(Mag,mag, axis=0)
230     M_sus = np.append(M_sus, 1/T*(Msq_mean-M_mean**2))
231     SpcH = np.append(SpcH, 1/T**2*(Esq_mean-E_mean**2))
232     for h in range(step):
233         l=t*step+h
234         Lattice_XY[l] = lattice_data[h]
235
236     print(T)
237     print(lattice_data.shape)
238 # plot the figures
239 T = Trange
240
241 plt.figure()
242 plt.plot(T, E, 'rx-')
243 plt.xlabel(r'Temperature  $(\frac{J}{k_B})$ ')
244 plt.ylabel(r' $\langle E \rangle$  per site  $(J)$ ')
245 plt.savefig("E.pdf", format='pdf', bbox_inches='tight')
246
247 plt.figure()
248 plt.plot(T, SpcH, 'kx-')
249 plt.xlabel(r'Temperature  $(\frac{J}{k_B})$ ')
250 plt.ylabel(r' $C_V$  per site  $(\frac{J^2}{k_B^2})$ ')
251 plt.savefig("Cv.pdf", format='pdf', bbox_inches='tight')
252
253 plt.figure()
254 plt.plot(T, M, 'bx-')
255 plt.xlabel(r'Temperature  $(\frac{J}{k_B})$ ')
256 plt.ylabel(r' $\langle |M| \rangle$  per site  $(\mu)$ ')
257 plt.savefig("M.pdf", format='pdf', bbox_inches='tight')
258
259 plt.figure()
260 plt.plot(T, M_sus, 'gx-')
261 plt.xlabel(r'Temperature  $(\frac{J}{k_B})$ ')
262 plt.ylabel(r' $\chi$   $(\frac{\mu}{k_B})$ ')
263 plt.savefig("chi.pdf", format='pdf', bbox_inches='tight')
264
265 plt.tight_layout()
266 fig = plt.gcf()
267 plt.show()

```

```

267 np.savetxt('output.data', np.c_[T, E, SpcH, M, M_sus])
268 np.savetxt('mag_data.csv', Mag)
269 #T = 0.1
270 # [XY, E_mean, M_mean, Esq_mean, Msq_mean] = SWang(T)
271 # Cv = 1 / T**2 * (Esq_mean - E_mean**2)
272 # M_sus = 1 / T * (Msq_mean - M_mean**2)
273 # [E1, M1] = EnMag(XY)
274 # E2 = E1/(L**2)
275 # print(E_mean, E2, M_mean, M1, Cv, M_sus)
276 ## plot the network cluster
277 # plt.figure()
278 # plt.matshow(XY, cmap='cool')
279 # plt.axis('off')
280 # np.savetxt('latticeXYpredict8_25000.csv', Lattice_XY)
281
282 num_lattice = nt * step / n_remove
283 num_lattice = int(num_lattice)
284 latticexy = np.zeros((num_lattice, L, L, 1))
285
286 Lattice_XY = Lattice_XY.reshape((nt * step, L, L, 1))
287 print(Lattice_XY.shape)
288 k = 0
289 for i in range(Lattice_XY.shape[0]):
290     if i % n_remove == 0:
291         latticexy[k] = Lattice_XY[i]
292         k += 1
293
294 spin_flatten = np.zeros((Lattice_XY.shape[0], Lattice_XY.shape[1]**2))
295 for i in range(Lattice_XY.shape[0]):
296     spin_flatten[i, :] = Lattice_XY[i, :, :].flatten()
297
298 np.savetxt('latticeXY32.csv', spin_flatten)

```

*Listing 1.7: dataxy32.py*

## Convolutional Neural Network

The following code implements the CNN to regenerate the transition temperature for the classical XY model for various lattice sizes.

```

1 # -*- coding: utf-8 -*-
2 """Copy of cnnXY.ipynb
3
4 Automatically generated by Colaboratory.
5
6 Original file is located at
7     https://colab.research.google.com/drive/1
8     HfqSm4Q3sVTq9yfwMp_tre5QvntrsMVh
9     """
10 # Commented out IPython magic to ensure Python compatibility.
11 import numpy as np
12 import pandas as pd
13 import matplotlib.pyplot as plt
14 # %matplotlib inline
15 import keras
16 from keras.datasets import mnist

```

```

17 from keras.models import Sequential
18 from keras.layers import Dense, Dropout, Activation, Flatten
19 from tensorflow.keras.optimizers import Adam
20 from keras.layers import BatchNormalization
21 from keras.utils import np_utils
22 from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D,
    GlobalAveragePooling2D
23 from keras.layers.advanced_activations import LeakyReLU
24 from keras.preprocessing.image import ImageDataGenerator
25 from tensorflow.keras import datasets, layers, models
26
27 model = models.Sequential()
28 model.add(layers.Conv2D(12, (3, 3), activation='relu', input_shape
    =(8,8,1)))
29 #model.add(layers.MaxPooling2D((2, 2)))
30 model.add(layers.Conv2D(8, (3, 3), activation='relu'))
31 model.add(layers.MaxPooling2D((2, 2)))
32 #model.add(layers.Conv2D(64, (3, 3), activation='relu'))
33
34
35 model.add(layers.Flatten())
36 model.add(layers.Dense(128, activation='relu'))
37 model.add(layers.Dense(2))
38 model.add(Activation('sigmoid'))
39
40 model.summary()
41
42 model.compile(loss='categorical_crossentropy', optimizer=Adam(),
    metrics=['accuracy'])
43
44 from google.colab import drive
45 drive.mount('/content/drive', force_remount=True)
46
47 import pandas as pd
48 import io
49 latticexy = pd.read_csv('/content/drive/MyDrive/
    latticexypredict16_25000.csv', delimiter=' ', header= None)
50 latticexy=latticexy.to_numpy()
51
52 np.shape(latticexy)
53
54 nt=10
55 step=50000
56 L=16
57 x=np.linspace(0.75,1,nt)
58 temp=np.zeros(step*nt)
59 for j in range(nt):
60     for i in range(step):
61         temp[j*step+i]=x[j]
62
63 latticexy
64
65 temp
66
67 '''k=0
68 for i in range(50000):
69     if temp[i]==1:
70         k+=1

```

```

71
72 print(k)
73 '''
74
75
76
77 latticexy=np.reshape(latticexy, (10*step,L,L,1))
78
79
80
81 print(latticexy.shape)
82 print(temp.shape)
83
84 latticexy_train=latticexy[0:40000]
85 print(np.shape(latticexy_train))
86 for i in range(nt):
87     if i!=0:
88         latticexy_train=np.append(latticexy_train,latticexy[50000*i:50000*i
89         +40000],axis=0)
89         print(np.shape(latticexy_train))
90 temp_train=temp[0:40000]
91 for i in range(10):
92     if i!=0:
93         temp_train=np.append(temp_train,temp[50000*i:50000*i+40000],axis=0)
94         print(temp_train.shape)
95
96 temp_train.shape
97
98 latticexy_test=latticexy[40000:50000]
99 for i in range(10):
100     if i!=0:
101         latticexy_test=np.append(latticexy_test,latticexy[50000*i
102         +40000:50000*i+50000],axis=0)
102 temp_test=temp[40000:50000]
103 for i in range(10):
104     if i!=0:
105         temp_test=np.append(temp_test,temp[50000*i+40000:50000*i+50000],
106         axis=0)
106
107 temp_train[temp_train<0.892]=0
108
109 temp_train[temp_train>0.892]=1
110
111 temp_test[temp_test>0.892]=1
112 temp_test[temp_test<0.892]=0
113
114 plt.imshow(latticexy_test[1990,:,: ,0])
115 plt.show
116
117 temp_train.shape
118
119 temperature_train=np.zeros((temp_train.shape[0],2))
120 temperature_test=np.zeros((temp_test.shape[0],2))
121 for i in range(temp_train.shape[0]):
122     if temp_train[i]==0:
123         temperature_train[i]=[1,0]
124     else:
125         temperature_train[i]=[0,1]

```

```
126
127 for i in range(temp_test.shape[0]):
128     if temp_test[i]==0:
129         temperature_test[i]=[1,0]
130     else:
131         temperature_test[i]=[0,1]
132
133 latticexy_test/=2*np.pi
134 latticexy_train/=2*np.pi
135
136 #model.fit(latticexy_train, temp_train, epochs=5, validation_data=tuple
137           (latticexy_test,temp_test))
138 model.fit(latticexy_train, temperature_train, batch_size=64, epochs=50,
139           validation_data=(latticexy_test,temperature_test))
140
141 score = model.evaluate(latticexy_test,temperature_test)
142 print()
143 print('Test accuracy: ', score[1])
144
145 predictions = model.predict(latticexy_test)
146
147 t = np.linspace(0.75, 1, 10)
148
149 y=np.zeros((10,1000,2))
150 for i in range(10):
151     y[i]=predictions[1000*i:1000*(i+1)]
152
153 y_avg=np.mean(y, axis=1)
154 y_avg_sum=np.mean(y_avg, axis=0)
155
156 y_avg.shape
157
158 line=np.full(10,0.8935)
159 y_line=np.linspace(0, 1,10)
160
161 plt.plot(t,y_avg[:,0])
162 plt.plot(t,y_avg[:,1])
163 plt.plot(line,y_line)
```

*Listing 1.8: cnnxy.py*

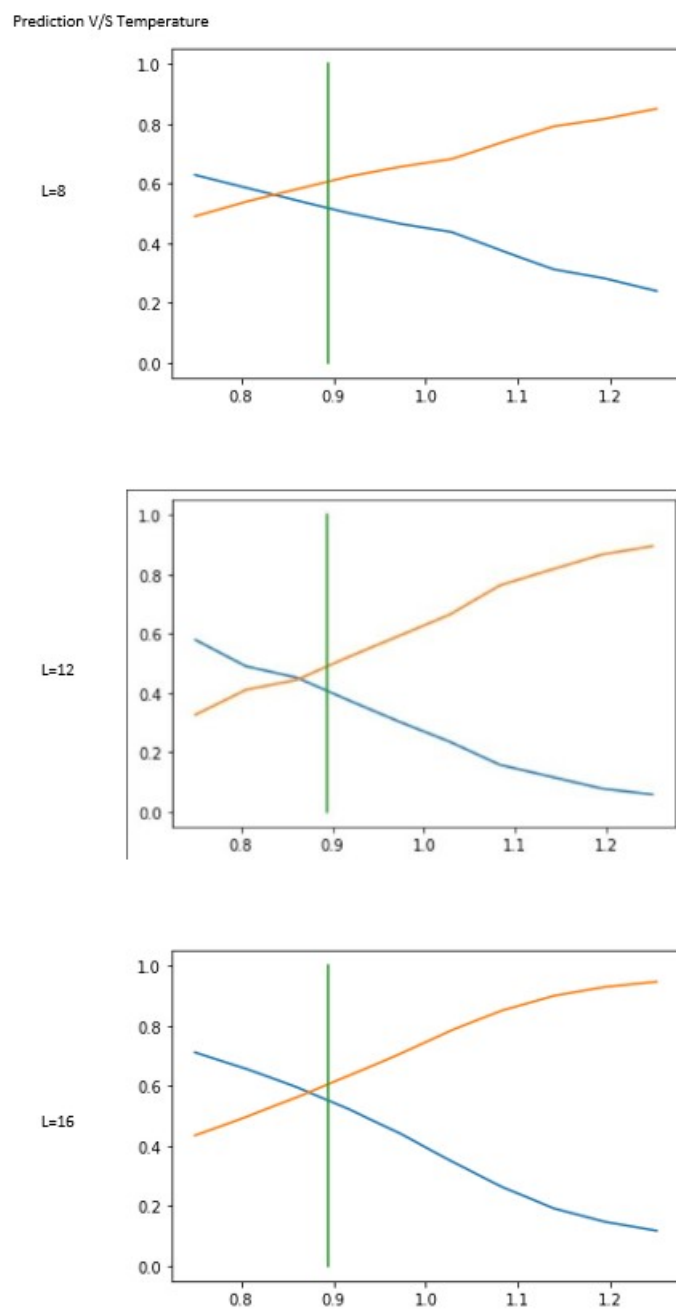


Figure 1.8: Transition temperature with CNN