Sourlas code

Yuxuan Lao
laoyx3@mail2.sysu.edu.cn

23/02/2025

Abstract

This is an assignment on the course Statistical Mechanics of Neural Networks at the School of Physics,Sun Yat-sen University in the fall of 2024.This programming assignment is to implement encoding and decoding scheme of Sourlas code,and show its decoding performance. It investigates the decoding performance at different temperatures: at the critical temperature, above and below it. The results show that decoding performance improves below the critical temperature, deteriorates above it, and exhibits instability at the critical temperature.And show the python codes in the last appendix part.

1 main result

The encoding of the Sourlas code was implemented, and the decoding program was set with the original code parameters $N = 50$, $K \equiv 3$, and $R \equiv 0.5$. Under these conditions, $\beta_c = \frac{1}{2}\ln\left(\frac{1-p}{p}\right)$ (as set by the problem), and $\beta = 5\beta_c$ (low-temperature simulation) and $\beta = 0.1\beta$ (high-temperature simulation) were used for comparison. Overall, the convergence is very unstable, as can be seen from the error bars in Figure 1.

At β_p, the overlap (decoding accuracy) sharply decreases in the range where $p \approx 0.03 \sim 0.04$, which may indicate a phase transition behavior.However ,for me,this phase transition behavior which happens in boundary line with one single parameter–flipping rate p is needed further study.

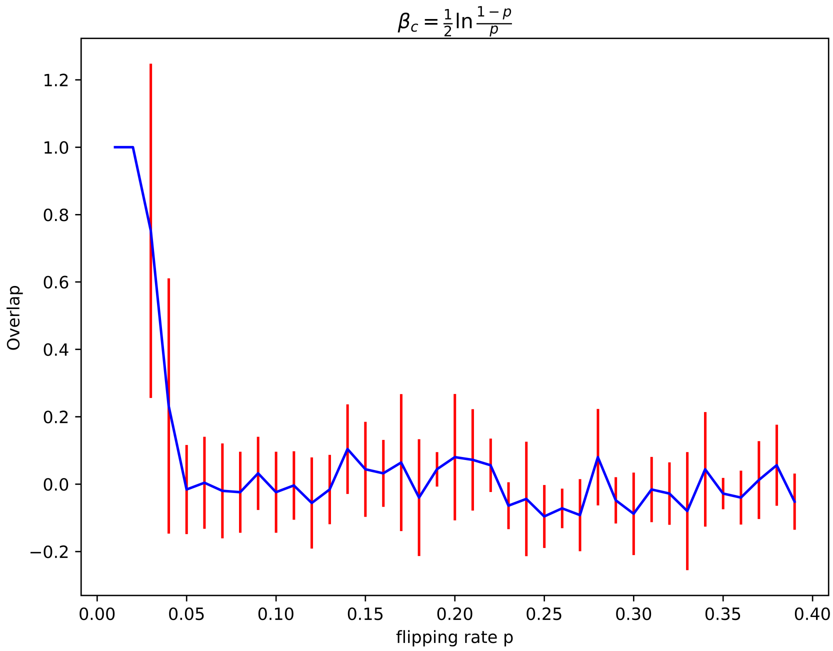

Figure 1: Decoding performance in critical temperature

When the temperature is above the critical temperature corresponding to $\beta_p(\beta = 0.1\beta_p)$, as shown in Figure 2, the decoding performance is very poor(even in the area where p is near zero), and the mean value of the overlap is approximately 0. This aligns with the theoretical result, indicating a transition into the paramagnetic phase.

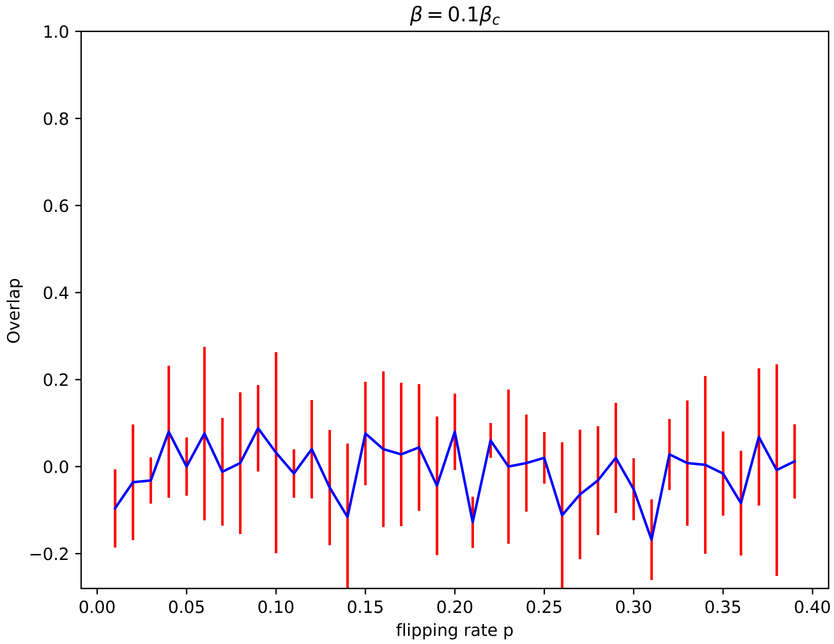

Figure 2: Decoding performance above critical temperature

In the low-temperature region: ($\beta = 5\beta_c$), as shown in the Figure3, the decoding performance clearly performs better over a larger range of p than the case with β_p. This is consistent with the prediction when the temperature is below the boundary temperature between the ferromagnetic and paramagnetic phases,decoding ability will be better.

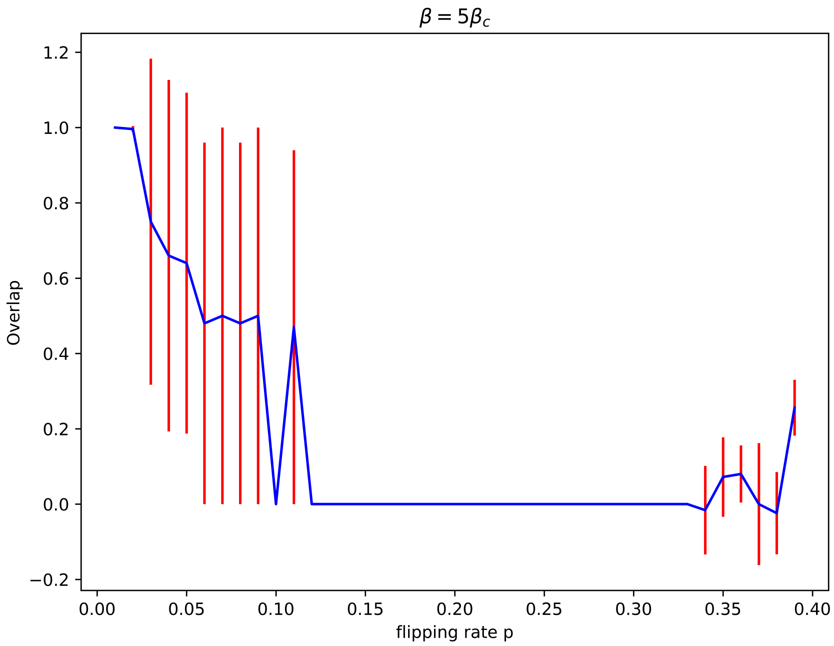

Figure 3: Decoding performance below critical temperature

2 Appendix:main.py

```
    import matplotlib.pyplot as plt
import numpy as np
import func

def main(p,beta):
 #

 N=50
 v,f,J,J_n,s=func.encode(N,p)
 #print(v)
 #print(f'v is',v)
 h,u,flag=func.iteration(v,f,J_n,beta,N)
 o=func.decode(v,u,beta,N,s)
 return o,flag

# o=0
# R=10
P=np.arange(0.01,0.4,0.01)
s=0
overlapc=[]
overlaph=[]
overlapl=[]

for p in P:
    beta=np.log((1-p)/p)*0.5

    for i in range(100):
        ele_oc,flagc=main(p,beta)

        while len(overlapc) <= s:
            overlapc.append([])
        if flagc:
            overlapc[s].append(ele_oc)
        print(f'now p is {p}')
        if len(overlapc[s])>=5:
            break
    s+=1


overlapmeanc=[np.mean(i) for i in overlapc]
overlapstdc=[np.std(i) for i in overlapc]
plt.figure(figsize=(8, 6))
plt.xlabel('flipping rate p')
plt.ylabel('Overlap')
plt.errorbar(P,overlapmeanc,overlapstdc,ecolor='r',color='b')
plt.title(r'$\beta_c=\frac{1}{2}\ln{\frac{1-p}{p}}$')
plt.savefig('overlapc.png',dpi=1000)
plt.show()
s=0
for p in P:
    beta=np.log((1-p)/p)*0.5

    for i in range(100):
        ele_oh,flagh=main(p,0.1*beta)

        while len(overlaph) <= s:
            overlaph.append([])
        if flagh:
            overlaph[s].append(ele_oh)
        print(f'now p is {p}')
        if len(overlaph[s])>=5:
            break
    s+=1
```

```
overlapmeanh=[np.mean(i) for i in overlaph]
overlapstdh=[np.std(i) for i in overlaph]
plt.figure(figsize=(8, 6))
plt.xlabel('flipping rate p')
plt.ylabel('Overlap')
plt.errorbar(P,overlapmeanh,overlapstdh,ecolor='r',color='b')
plt.title(r'$\beta=0.1\beta_c$')
plt.savefig('overlaph.png',dpi=1000)
plt.show()
s=0
for p in P:
    beta=np.log((1-p)/p)*0.5

    for i in range(100):
        ele_ol,flagl=main(p,5*beta)

        while len(overlapl) <= s:
            overlapl.append([])
        if flagl:
            overlapl[s].append(ele_ol)
        print(f'now p is {p}')
        if len(overlapl[s])>=5:
            break
    s+=1
overlapmeanl=[np.mean(i) for i in overlapl]
overlapstdl=[np.std(i) for i in overlapl]
plt.figure(figsize=(8, 6))
plt.xlabel('flipping rate p')
plt.ylabel('Overlap')
plt.errorbar(P,overlapmeanl,overlapstdl,ecolor='r',color='b')
plt.title(r'$\beta=5\beta_c$')
plt.savefig('overlapl.png',dpi=1000)
plt.show()
```

3 Appendix:used function.py

```
import numpy as np

def flip_with_probability(value, p):
    # value to be  flippedp probability
    if np.random.rand() < p:
        return -1*value # fliiped
    else:
        return value
def encode_even(N,p,R=3):
    varia_node={}#i to a

    func_node={}#a to i
    all_indices=np.arange(N)
    np.random.shuffle(all_indices)

    selected_indices=[]
    for i in range(2*N):
        available_indices = np.setdiff1d(all_indices, selected_indices)
        if len(available_indices) < R:
            available_indices = all_indices

        b = np.random.choice(available_indices, R, replace=False)

        selected_indices.extend(b)
        func_node[i] = b
        for j in b:
            if j not in varia_node:
```

```
                varia_node[j] = []
            varia_node[j].append(i)

    source_code = np.random.choice((-1, 1), N)
    #print(f'source code is', source_code, '\n')
    J = {}
    J_withnoise={}
    for funcnode in func_node.keys(): # generate interaction,i.e.encode
        j = 1
        for varianode in func_node[funcnode]:
            j *= source_code[varianode]
        J[funcnode] = j
        J_withnoise[funcnode] = flip_with_probability(j, p)
    #print(f'J is\n', J, "\n")
    #print(f'J_n is\n', J_withnoise, "\n")
    #print(varia_node)
    return varia_node, func_node, J, J_withnoise,source_code


def encode(N,p):
    varia_node={}#i to a
    func_node={}#a to i
    for i in range(2*N):
        b = np.random.choice(range(N), 3, replace=False)
        func_node[i]=b
        for j in b:
            if j not in varia_node:
                varia_node[j]=[]
            varia_node[j].append(i)
    source_code=np.random.choice((-1,1),N)
    #print(f'source code is',source_code,'\n')
    J={}
    J_withnoise = {}
    for funcnode in func_node.keys():#generate interation i.e.encode
        j=1
        for varianode in func_node[funcnode]:
            j *= source_code[varianode]
        J[funcnode]=j
        J_withnoise[funcnode] = flip_with_probability(j, p)
    #print(f'J is\n',J,"\n")
    return varia_node,func_node,J,J_withnoise,source_code


def iteration(v,f,J,beta,N,max_iter=500,tol=1e-4,K=3):
    flag=False
    h={}
    u={}
    #np.random.seed(42)
    for varianode in v.keys():#h_{i\to a}initialize
        for funcnode in v[varianode]:
            if varianode not in h:
                h[varianode] = {}
            #h[varianode][funcnode]=np.random.normal(0,1,1)
            h[varianode][funcnode] = np.random.uniform(-1, 1, 1)
            #h[varianode][funcnode] = np.random.choice((-1,1), 1)
    for funcnode in f.keys():#u_{a\to i}initialize
        for varianode in f[funcnode]:
            if funcnode not in u:
                u[funcnode] = {}
            #u[funcnode][varianode]=np.random.normal(0,1,1)
            u[funcnode][varianode] = np.random.uniform(-1, 1, 1)
            #u[funcnode][varianode] = np.random.choice((-1,1), 1)
    for t in range(max_iter):#iteration
        delta=0.0
        for i in v.keys():
            for a in v[i]:#h_{ia}
```

```
                h_new=0
                for b in v[i]:#all nodes connected to i
                    if b!=a:
                        h_new += u[b][i]
                        #print(u[b][i])
                delta+=abs(h_new-h[i][a])
                h[i][a]=h_new

        for a in f.keys():
            for i in f[a]:#u_{ai}
                tanhprod=1
                for j in f[a]:#all nodes connected to a
                    if j!=i:
                        tanhprod *= np.tanh(beta*h[j][a])
                u_new=(np.arctanh(np.tanh(beta*J[a])*tanhprod))/(beta)
                delta+=abs(u_new-u[a][i])
                u[a][i]=u_new


        if delta/(N*K)<tol:
            flag=True
            break
    if flag:
        print(t,'time Converged')
    else:
        print('Not Converged')
    return h,u,flag

def decode(v,u,beta,N,origin):
    decoded_code=np.zeros(N)
    #print(f'u is\n',u,'\n')
    for i in range(N):
        sum=0
        if i not in v:
            continue
        for b in v[i]:
            #print(u[funcnode][i])
            sum+=u[b][i]
        #print(sum)

        decoded_code[i]=np.sign(sum)

    #print(f'decoded code is\n',decoded_code)
    overlap=(np.inner(origin,decoded_code))/N
    # l=0
    # for i in range(N):
    #    if decoded_code[i]!=origin[i]:
    #        print(f'wrong length is',len(v[i]))
    #        l+=len(v[i])
    #    else:
    #        print(f'right length is',len(v[i]))
    #        l+=len(v[i])

    print(f'Overlap:\n {overlap}')
    return overlap
#------------------------------------------
```