FINAL REPORT

Statistical Brain Theory and Parallel Computing

Name: **Faza Thirafi**

Student ID: **20M18840**

Task 1: Probabilistic and Binary Model

Background

Probabilistic and Binary Model is a mathematical model to determine a value of variable based on random number ($rand()$) and some other known probability (p)

$$y = \begin{cases} 1; & if\ rand() \leq p \\ 0; & if\ rand() > p \end{cases}$$

To find the probability, we use a network that has some input value (x) that are connected to the result (y) which has weight (w) on the connection. This connection enables us to find the weighted sum ($\hat{s}$) as the representation of the network

$$\hat{s} = \sum_{i=1}^{N} x_i \sim w_i$$

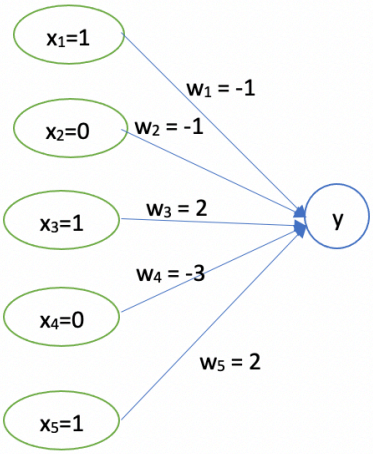
Weighted sum is a parameter that can be evaluated to the probabilistic binary model as the probability parameter. Before that, we need to add a logistic function to have a value of 0 ~ 1 as probabilistic value. Here, we use Sigmoid Function to give the good value of probability.

$$P = \sigma(\hat{s}) = \frac{1}{1 + e^{-\alpha \hat{s}}}$$

where α denotes the gain constant, e denotes the natural number (2.71…), and we put the weighted sum as the sigmoid function's parameter. Bigger gain constant causes sharper curve (more like step function), and smaller gain will do otherwise.

Experiment (source code: task_1_solver.py)

The figure below represents the connection with its input value and weight for each edge.



To see the gain constant effect, we use multiple values of gain: α: $[0.1, 0.5, 1.0, 2.0, 5.0]$

We use 10000 epochs to count the output which gives 0 and 1 value. Here is the result:

alpha (α)	1 Ratio (%)	0 Ratio (%)
0.1	56.94	43.06
0.5	82.01	17.99
1.0	95.18	4.82
2.0	99.81	0.19
5.0	100	0

From this result, we see that the gain constant will affects the value even more. Smaller value will give more random result than the bigger one.

Task 2: Recurrent Neural Network

Background

Recurrent Neural Network can be used to solve a polynomial equation where the variable value represents the input neuron. Energy function need to be constructed based on the list of equations that need to be solved. In this case, below are the equations

$$\begin{cases} x_1 - x_2 + x_3 + x_4 = 2 \\ -x_1 + 2x_2 - x_3 - x_4 = -2 \\ -x_1 + 2x_2 - x_3 - x_4 = 4 \\ -x_1 - x_2 + x_3 + x_4 = 0 \end{cases}$$

Based on those equations we need to transform them to a single Energy function that must have the minimum result. In order to obtain the minimum result, we will utilize the characteristic of square number. Whatever the number is, if we get it squared, the result will be 0 or positive. This leads to the idea of making all the equations square, then sum all of them in an energy function that should have 0 result as minimum value.

$$E(x_1, x_2, x_3, x_4) = (x_1 - x_2 + x_3 + x_4 - 2)^2 + (-x_1 + 2x_2 - x_3 - x_4 + 2)^2 + (2x_1 - x_2 + 2x_3 + x_4 - 4)^2 + (-x_1 - x_2 + x_3 + x_4)^2$$

After generating random values of the variables, we have to step by step minimize the energy function so that we can get closer to the solution. This could be done by changing the neuron values in RNN.

For the network itself, we have to determine all the thresholds of each neuron and all the weights of each connecting neurons. We assume that all neurons are connected each other. Each weight is calculated as formula below

$$w_{ij} = E|_{x_i=1;\, others=0} + E|_{x_j=1;\, others=0} - E|_{x_i=1;\, x_j=1} - E|_{x^{\forall}=0}$$

For the thresholds, we calculate on all neurons with the formula below

$$\theta_i = E|_{x_i=1;\, others=0} - E|_{x^{\forall}=0}$$

Experiment (source code: task_2_solver.py)

Based on the energy function, we get the threshold as follows

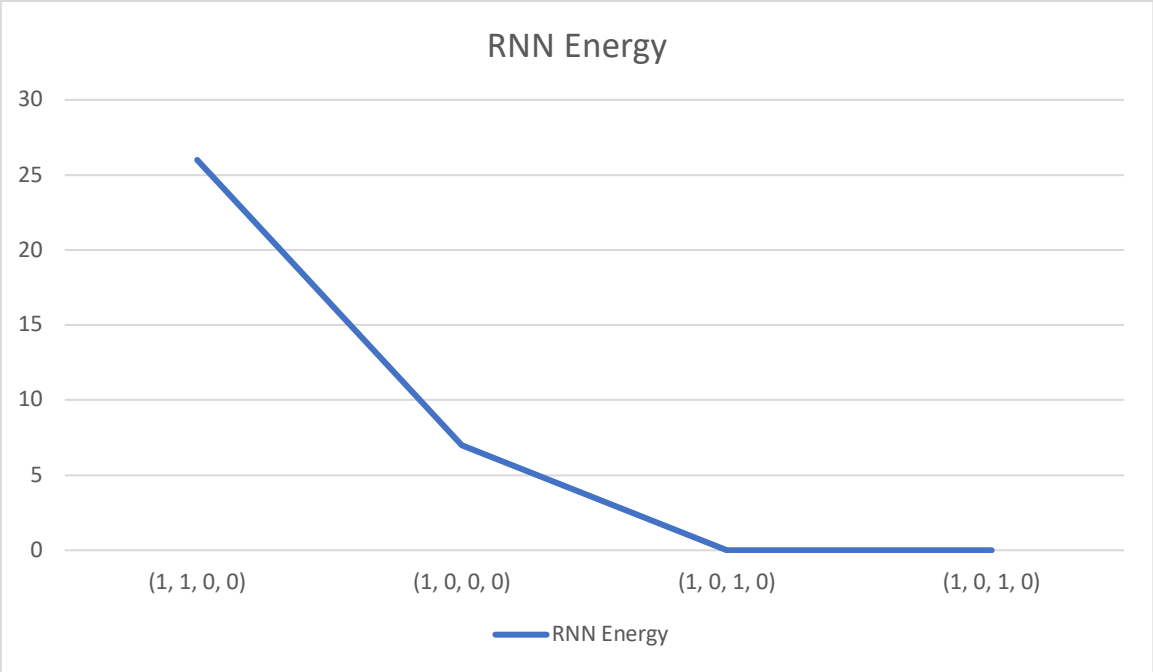
Thresholds (θ)	Neuron 1	Neuron 2	Neuron 3	Neuron 4
	-17.0	27.0	-17.0	-12.0

For the weights, the values are showed in the following table

Weight (w_{ij})	Neuron 1	Neuron 2	Neuron 3	Neuron 4
Neuron 1	0	8.0	-10.0	-6.0
Neuron 2	8.0	0	12.0	10.0
Neuron 3	-10.0	12.0	0	-10.0
Neuron 4	-6.0	10.0	-10.0	0

With $\alpha = 1.0$, the table below shows the steps of updating neuron values so that energy decreases for every steps

Time	States	Energy
0	(1, 1, 0, 0)	26.0
1	(1, 1, 0, 0)	26.0
2	(1, 0, 0, 0)	7.0
3	(1, 0, 1, 0)	0.0
4	(1, 0, 1, 0)	0.0



With the result above, we got that the solution of tuples (x_1, x_2, x_3, x_4) which resulting the minimum energy is **(1, 0, 1, 0)**.

Task 3: Boltzmann Distribution on Energy

Background

There are so many kinds of distribution function in Mathematics. One of the most common one to use in this context is Boltzmann Distribution. This distribution could be utilized to find the equilibrium state.

$$\begin{cases} x_1 - 2x_2 + x_3 = -1 \\ -x_1 + x_2 + x_3 = 2 \\ 2x_1 - x_2 - x_3 = -2 \end{cases}$$

$$E(x_1, x_2, x_3) = (x_1 - 2x_2 + x_3 + 1)^2 + (-x_1 + x_2 + x_3 - 2)^2 + (2x_1 - x_2 - x_3 + 2)^2$$

Standard form of Energy function

$$E(x_1, x_2, x_3) = -\frac{1}{2}\sum_{n=1}^{3}\sum_{m=1}^{3} w_{nm} x_n x_m + \sum_{n=1}^{3} \theta_n x_n + C$$

In this task, we are introduced with gibbs copy that represents the number of RNN copies in a certain state of neurons at the same time

$$N(x_1, x_2, x_3) = Ae^{E(x_1, x_2, x_3)}$$

Where A denotes the normalization coefficient so that total number of RNN copies will be equal to the given number (theoretically)

Experiment (source code: task_3_solver.py)

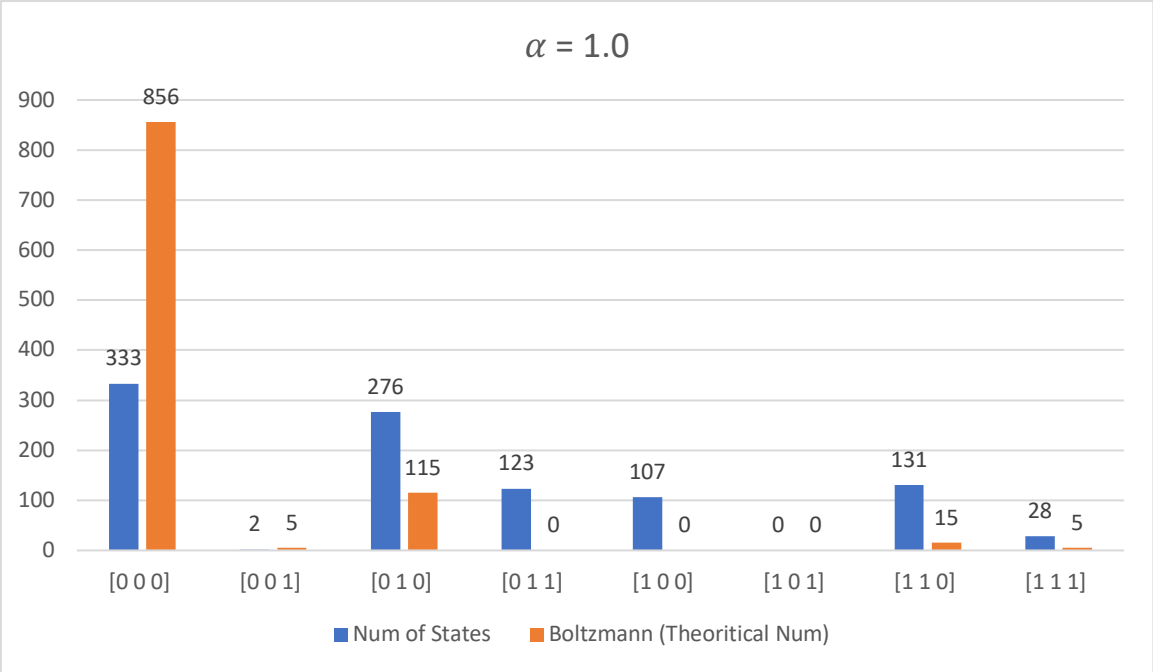
Based on the energy function, we get the threshold as follows

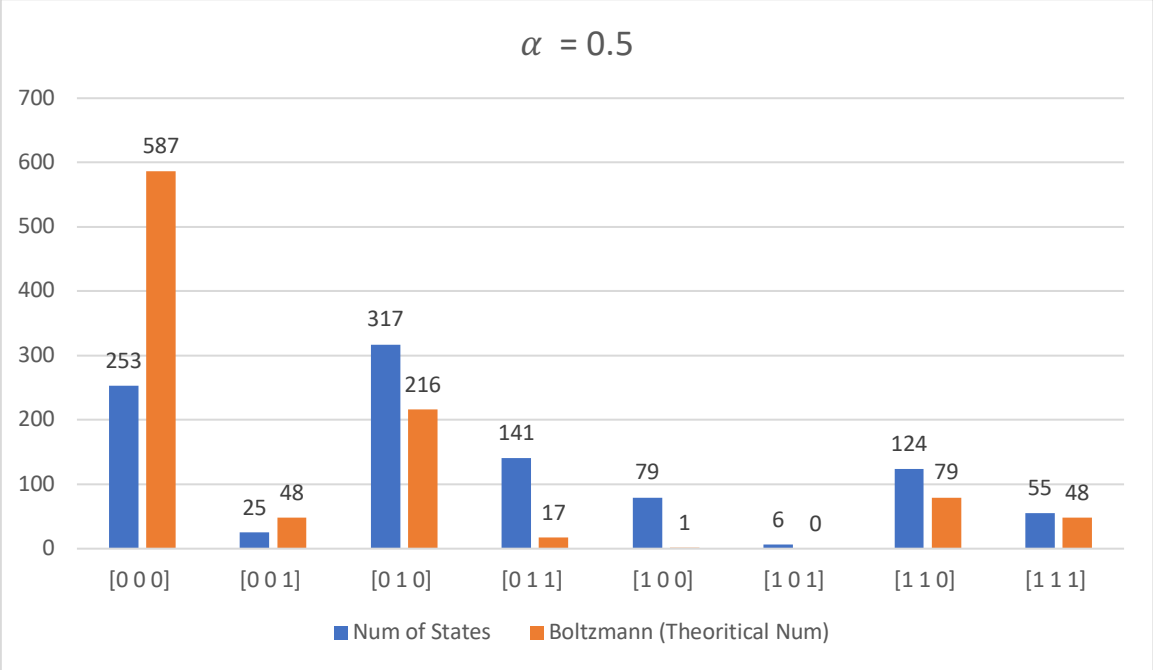
Thresholds (θ)	Neuron 1	Neuron 2	Neuron 3
	12.0	2.0	5.0

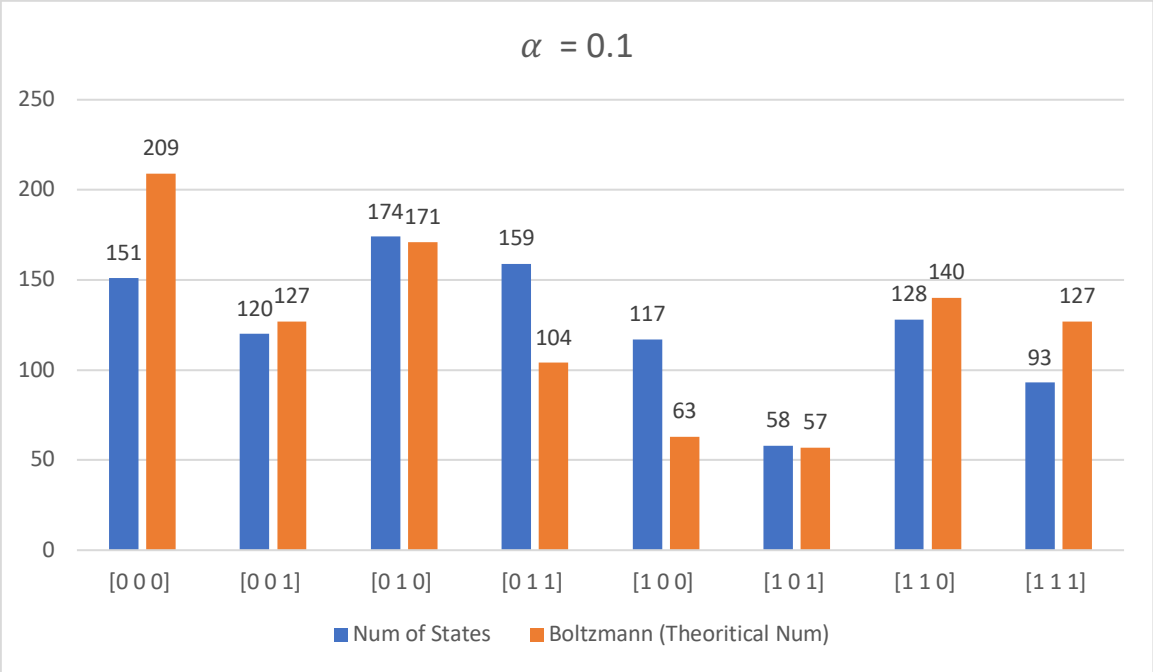
For the weights, the values are showed in the following table

Weight (w_{ij})	Neuron 1	Neuron 2	Neuron 3
Neuron 1	0	10.0	4.0
Neuron 2	10.0	0	0.0
Neuron 3	4.0	0.0	0

In this problem, we set $A = 1000$ that represents the total gibbs copies of RNN. In order to find the best setup, we adjust the value of $\alpha = [0.1, 0.5, 1.0]$.







From the above results, we got that gain value is really affecting the randomness of the obtained result. When it gets smaller, the number of gibbs copies of each states between experimental and theoretical number gets closer. It Is caused by the randomness factor that is controlled by gain constant here.

Task 4: Usage of Beta on RNN

Background

For RNN problems, we can also utilize another factor to control the local minima that is caused by the equation. We can attach β constant to control local minima and get more closer to the global minima. The case of this problem is represented by the energy function below

$$E(x_1, x_2, x_3, x_4) = (x_1 - 0)^2 + (x_2 - 1)^2 + (x_3 - 1)^2 + (x_4 - 0)^2$$
$$+[(x_1 - 1)^2 + (x_2 - 0)^2 + (x_3 - 1)^2 + (x_4 - 0)^2]\beta$$
$$+(x_1 - x_2 + x_3 + x_4)^2 + (2x_1 - x_3 + x_4 + 1)^2$$

On the first line, we got that the local minima would be (0, 1, 1, 0) for the tuple. But if we look at the second line, the local minima would be (1, 0, 1, 0). It means that we really need to avoid the local minima to obtain the global minima.

Experiment (source code: task_4_solver.py)

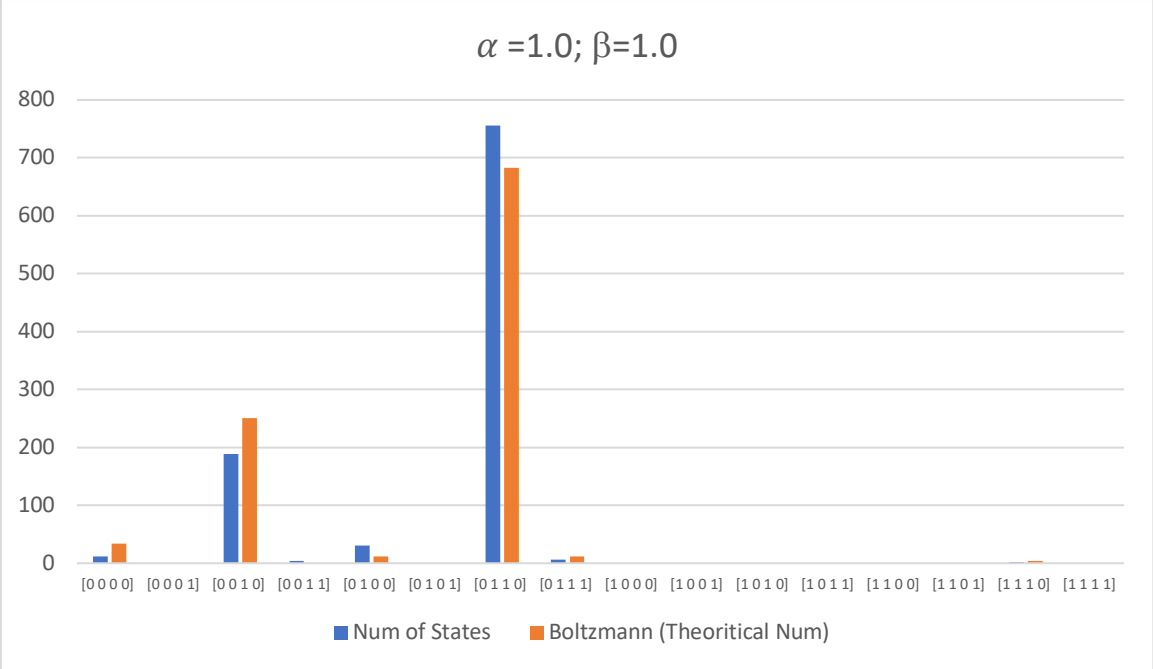
Based on the energy function, we get the threshold as follows

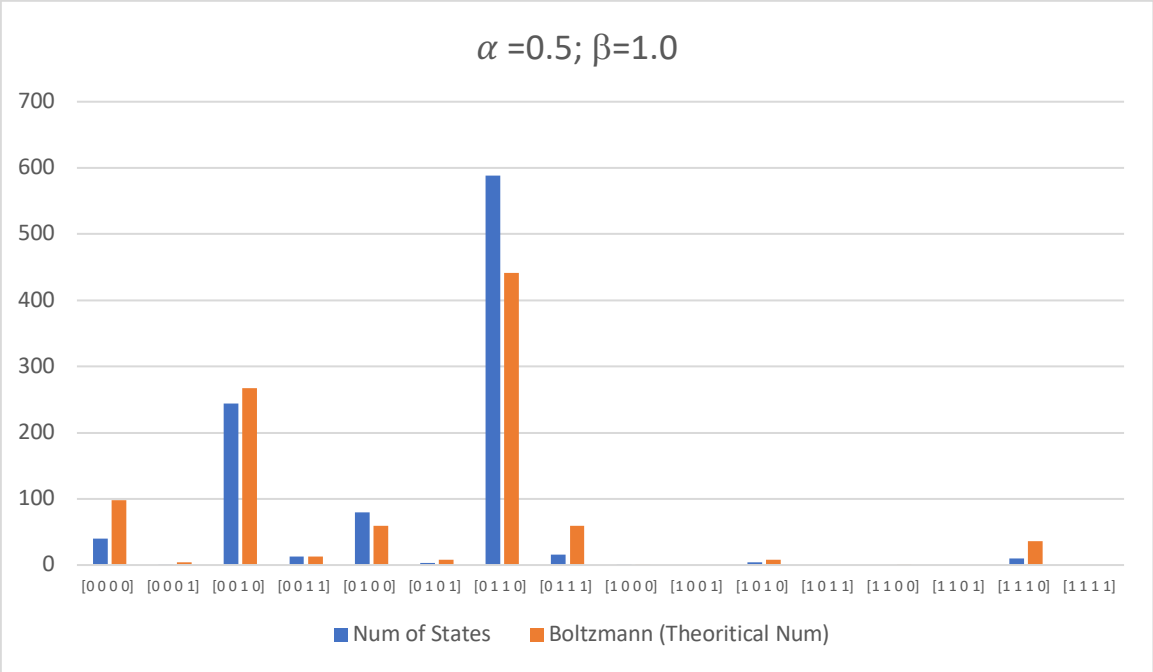
Thresholds (θ)	Neuron 1	Neuron 2	Neuron 3	Neuron 4
	9.0	1.0	-2.0	6.0

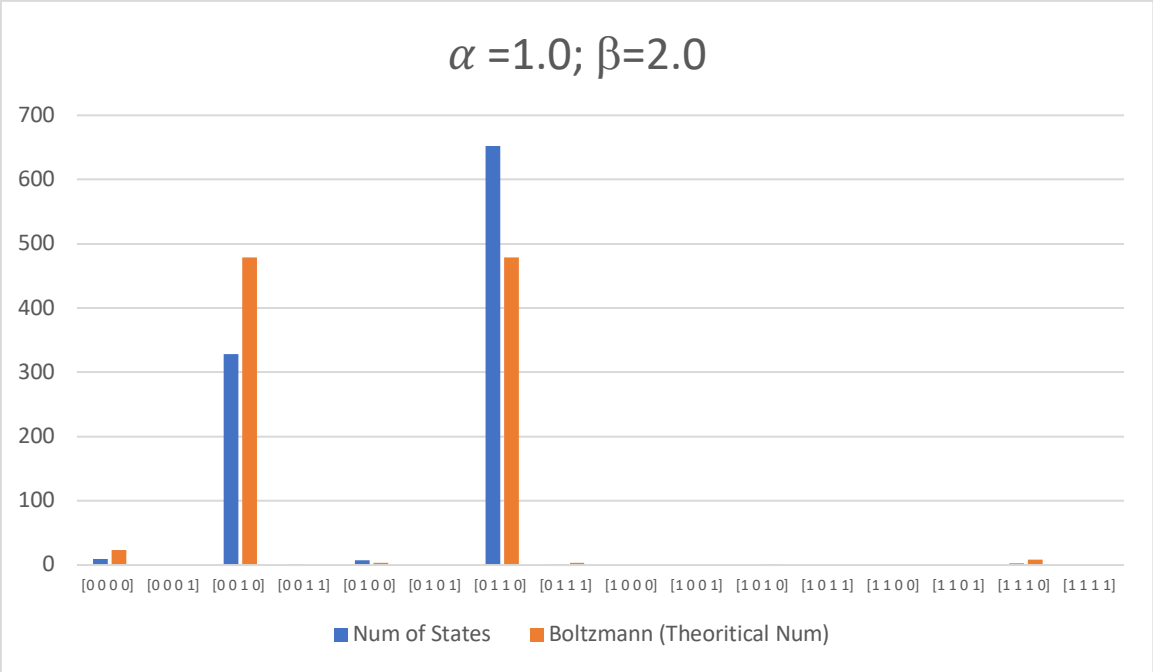
For the weights, the values are showed in the following table

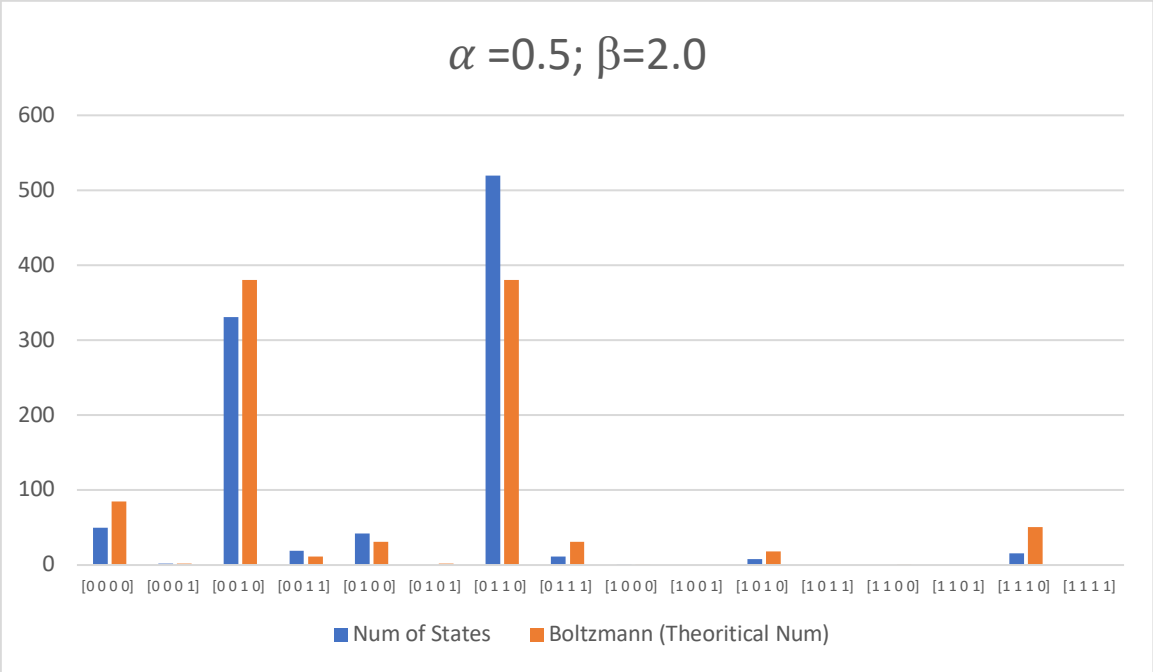
Weight (w_{ij})	Neuron 1	Neuron 2	Neuron 3	Neuron 4
Neuron 1	0	2.0	2.0	-6.0
Neuron 2	2.0	0	2.0	2.0
Neuron 3	2.0	2.0	0	0.0
Neuron 4	-6.0	2.0	0.0	0

In this problem, we set $A = 1000$ that represents the total gibbs copies of RNN. In order to find the best setup, we adjust the value of $\alpha = [0.5, 1.0]$ and $\beta = [1.0, 2.0]$ to find the effect of both variables.




α =0.5; β=1.0
700
600
500
400
300
200
100
0
[0 0 0 0] [0 0 0 1] [0 0 1 0] [0 0 1 1] [0 1 0 0] [0 1 0 1] [0 1 1 0] [0 1 1 1] [1 0 0 0] [1 0 0 1] [1 0 1 0] [1 0 1 1] [1 1 0 0] [1 1 0 1] [1 1 1 0] [1 1 1 1]
Num of States
Boltzmann (Theoritical Num)


α =1.0; β=2.0
700
600
500
400
300
200
100
0
[0 0 0 0] [0 0 0 1] [0 0 1 0] [0 0 1 1] [0 1 0 0] [0 1 0 1] [0 1 1 0] [0 1 1 1] [1 0 0 0] [1 0 0 1] [1 0 1 0] [1 0 1 1] [1 1 0 0] [1 1 0 1] [1 1 1 0] [1 1 1 1]
Num of States
Boltzmann (Theoritical Num)



From above results, we got that α will control the randomness. Less value of it will make it more random. For the β variable, we got that for bigger value of β =2.0, there are more copies trapped into the local minima. So, the smaller value is needed here which is β = 1.0. So, we have to adjust it properly to get the correct answer

Task 5: Four Queens Problem

Background

Queens problem is one of the problems in mathematics to put some queens in a chess board so that no queen is placed in the same horizontal and vertical line (it's okay for the diagonal line). In this case, we use 4 queens on a 4x4 chess board.

Energy function that we can use to find the places of queens so that it satisfies the rules is as written below

$$E(x_{11}, x_{12} \dots x_{44}) = \sum_{i=1}^{4}\left(\sum_{j=1}^{4} x_{ij} - 1\right)^2 + \sum_{j=1}^{4}\left(\sum_{i=1}^{4} x_{ij} - 1\right)^2$$

Where the neurons (x_{ij}) represent cell of queen position. If x_{ij} is 1, then there exists a queen at (i, j) position. Otherwise, 0 means there is no queen at that position.

Experiment (source code: task_5_solver.py)

Based on the energy function, we get the threshold as follows

Thresholds (θ)	1	2	3	4
1	0	-2.0	-2.0	-2.0
2	-2.0	0	-2.0	-2.0
3	-2.0	-2.0	0	-2.0
4	-2.0	-2.0	-2.0	0

For the weights, the values are showed in the following table

	11	12	13	14	21	22	23	24	31	32	33	34	41	42	43	44
11	0	-2	-2	-2	-2	0	0	0	-2	0	0	0	-2	0	0	0
12	-2	0	-2	-2	0	-2	0	0	0	-2	0	0	0	-2	0	0
13	-2	-2	0	-2	0	0	-2	0	0	0	-2	0	0	0	-2	0
14	-2	-2	-2	0	0	0	0	-2	0	0	0	-2	0	0	0	-2
21	-2	0	0	0	0	-2	-2	-2	-2	0	0	0	-2	0	0	0
22	0	-2	0	0	-2	0	-2	-2	0	-2	0	0	0	-2	0	0
23	0	0	-2	0	-2	-2	0	-2	0	0	-2	0	0	0	-2	0
24	0	0	0	-2	-2	-2	-2	0	0	0	0	-2	0	0	0	-2
31	-2	0	0	0	-2	0	0	0	0	-2	-2	-2	-2	0	0	0
32	0	-2	0	0	0	-2	0	0	-2	0	-2	-2	0	-2	0	0
33	0	0	-2	0	0	0	-2	0	-2	-2	0	-2	0	0	-2	0
34	0	0	0	-2	0	0	0	-2	-2	-2	-2	0	0	0	0	-2
41	-2	0	0	0	-2	0	0	0	-2	0	0	0	0	-2	-2	-2
42	0	-2	0	0	0	-2	0	0	0	-2	0	0	-2	0	-2	-2
43	0	0	-2	0	0	0	-2	0	0	0	-2	0	-2	-2	0	-2
44	0	0	0	-2	0	0	0	-2	0	0	0	-2	-2	-2	-2	0

Result:

State pos	Num of States	Boltzmann
0	44.0	0.0
153	1.0	0.0
1360	1.0	0.0
2720	1.0	0.0
4680	0.0	41.0
4740	0.0	41.0
5160	0.0	41.0
5250	0.0	41.0
6180	0.0	41.0
6210	0.0	41.0
8520	0.0	41.0
8580	0.0	41.0
9240	0.0	41.0
9345	0.0	41.0
10260	0.0	41.0
10305	0.0	41.0
16680	0.0	41.0
16770	0.0	41.0
16920	0.0	41.0
17025	1.0	41.0
18450	0.0	41.0
18465	0.0	41.0
26112	1.0	0.0
33060	0.0	41.0
33090	0.0	41.0
33300	0.0	41.0
33345	0.0	41.0
33810	0.0	41.0
33825	0.0	41.0
37008	1.0	0.0
49344	2.0	0.0
65534	1.0	0.0
65535	947.0	0.0

Note on **State Pos**:

- "0" represents:
  0 0 0 0
  0 0 0 0
  0 0 0 0
  0 0 0 0

- “33825” represents:
  1 0 0 0
  0 1 0 0
  0 0 1 0
  0 0 0 1

If we look at the value, there possibly a bug happened where looking for the state changing. But for the theoretical number, it will be the correct state where the solution appears in an equal number of copies (One of the solutions: “33825”).

Task 6: Travelling Salesman Problem

Background

Travelling salesman problem is one of the problems in mathematics to find the shortest path of a salesman who is travelling to some points of places from the starting point and get back to the same point after visiting all listed cities/places

There are two energy functions that represents the way we solve this problem. The first energy (E_C) is used to find the appropriate state of the way the salesman visited those cities once for each city.

$$E_C(x_{11}, x_{12} \dots x_{44}) = \sum_{i=1}^{4}\left(\sum_{j=1}^{4} x_{ij} - 1\right)^2 + \sum_{j=1}^{4}\left(\sum_{i=1}^{4} x_{ij} - 1\right)^2$$

The second energy is used to find the order of visiting cities to ensure that the total distance must be the lowest.

$$E_L(x_{11}, x_{12} \dots x_{44}) = \sum_{i=1}^{4}\left(\sum_{k=1}^{4}\sum_{l=1}^{4} d_{kl} x_{ik} x_{(i+1)k}\right)$$

The final energy function will be considering those 2 energy functions so that the both conditions will be satisfied

$$E(x_{11}, x_{12} \dots x_{44}) = E_C(x_{11}, x_{12} \dots x_{44}) + \beta E_L(x_{11}, x_{12} \dots x_{44})$$

Experiment (source code: task_6_solver.py)

As we could define the distance of every connecting cities, the table below represents the distance between cities

	City 1	City 2	City 3	City 4
City 1	0	1.72	6.23	3.10
City 2	1.72	0	2.24	2.24
City 3	6.23	2.24	0	1.41
City 4	3.10	2.24	1.41	0

Result of RNN training:

Sequance	Num	Boltzmann
4680	0.0	4.0
4740	0.0	4.0
5160	0.0	4.0
5250	0.0	4.0
6180	0.0	4.0
6210	0.0	4.0
8520	0.0	4.0
8580	0.0	4.0
9240	0.0	4.0
9345	0.0	4.0
10260	0.0	4.0
10305	0.0	4.0
16680	0.0	4.0
16770	0.0	4.0
16920	0.0	4.0
17025	0.0	4.0
18450	0.0	4.0
18465	0.0	4.0
33060	0.0	4.0
33090	0.0	4.0
33300	0.0	4.0
33345	0.0	4.0
33810	0.0	4.0
33825	0.0	4.0
65519	1.0	0.0
65535	999.0	0.0

Note on **State Pos**:

- "0" represents:
  0 0 0 0
  0 0 0 0
  0 0 0 0
  0 0 0 0
- "33825" represents:
  1 0 0 0
  0 1 0 0
  0 0 1 0
  0 0 0 1

Again, if we look at the value, there possibly a bug happened where looking for the state changing. But for the theoretical number, it will be the correct state where the solution appears in an equal number of copies (One of the solutions: "33825").

Task 7: RNN Training Procedure

Background

Recurrent Neural Network is one kind of Deep Neural Network that has ability to connect random 2 neurons, regardless of the layer. This network needs to be trained with some epochs to get the representing model for the input/output values. Every end of the epoch, all of the weights between connected neuron should be updated based on the result by the following formula

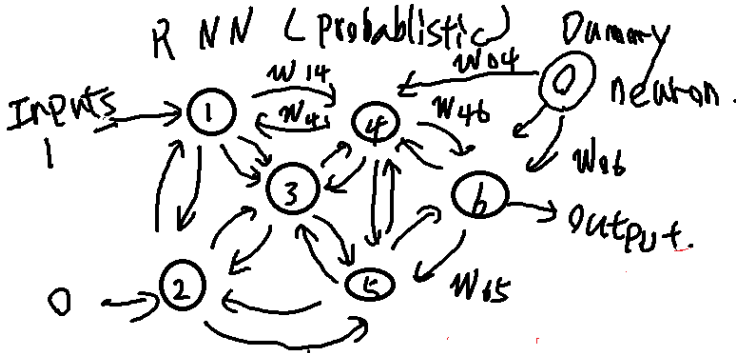
$$w_{ij}{}^{new} = w_{ij}{}^{old} - \varepsilon \frac{\delta E}{\delta w_{ij}}$$

Where E denotes error and ε denotes learning rate.

After a certain epoch, the final weight tuples will be representing the trained RNN model that has been ready to be used for any inputs.

Experiment (source code: task_7_solver.py)

The network that we use for this task is showed with the figure below



In order to determine the value of inputs and output, we use conditional probability

$$P\ (x_6|x_1, x_2)$$

Which is represented in the table below

P (x_1, x_2)	x_1 (value)	x_2 (value)	x_6	
			0 (prob)	1 (prob)
0.2	**0**	**0**	0.7	0.3
0.3	**1**	**0**	0.4	0.6
0.4	**0**	**1**	0.1	0.9
0.1	**1**	**1**	0.8	0.2

Result of training:

100 Epochs

```
================================================
TASK 7
================================================
INITIAL WEIGHT:
[[0.         0.         0.         0.         0.93680176 0.
  0.3361612 ]
 [0.         0.         0.90188247 0.44323594 0.24889124 0.
  0.        ]
 [0.         0.90188247 0.         0.02323021 0.         0.16020721
  0.        ]
 [0.         0.44323594 0.02323021 0.         0.13955214 0.8010221
  0.        ]
 [0.93680176 0.24889124 0.         0.13955214 0.         0.93041702
  0.63842457]
 [0.         0.         0.16020721 0.8010221  0.93041702 0.
  0.7451438 ]
 [0.3361612  0.         0.         0.         0.63842457 0.7451438
  0.        ]]


FINAL WEIGHT:
 [[0.         0.         0.         0.         1.00664176 0.
  0.3985312 ]
 [0.         0.         0.95317247 0.47067594 0.28272124 0.
  0.        ]
 [0.         0.95317247 0.         0.04447021 0.         0.18350721
  0.        ]
 [0.         0.47067594 0.04447021 0.         0.15101214 0.8072121
  0.        ]
 [1.00664176 0.28272124 0.         0.15101214 0.         0.94190702
  0.66441457]
 [0.         0.         0.18350721 0.8072121  0.94190702 0.
  0.7675938 ]
 [0.3985312  0.         0.         0.         0.66441457 0.7675938
  0.        ]]
```

1000 Epochs

```
INITIAL WEIGHT:
[[0.         0.         0.         0.         0.49389819 0.
  0.99676876]
 [0.         0.         0.87200016 0.0326647  0.79083747 0.
  0.        ]
 [0.         0.87200016 0.         0.16229315 0.         0.43005396
  0.        ]
 [0.         0.0326647  0.16229315 0.         0.17710638 0.17414895
```

```
  0.        ]
 [0.49389819 0.79083747 0.         0.17710638 0.         0.95304195
  0.95071094]
 [0.         0.         0.43005396 0.17414895 0.95304195 0.
  0.31859544]
 [0.99676876 0.         0.         0.         0.95071094 0.31859544
  0.        ]]


FINAL WEIGHT:
 [[0.         0.         0.         0.         1.35190019 0.
  1.82882376]
 [0.         0.         1.59101916 0.3604367  1.22154747 0.
  0.        ]
 [0.         1.59101916 0.         0.48568815 0.         0.84162896
  0.        ]
 [0.         0.3604367  0.48568815 0.         0.30209438 0.26696095
  0.        ]
 [1.35190019 1.22154747 0.         0.30209438 0.         1.10312495
  1.31485894]
 [0.         0.         0.84162896 0.26696095 1.10312495 0.
  0.65858144]
 [1.82882376 0.         0.         0.         1.31485894 0.65858144

  0.        ]]
```