Modeling Lattice Structures Using Structural Motifs

Krystal R. York

ECE 6970

Fall 2021

1 Introduction

1.1 Disorder Based on Structural Motifs

This project aims to generate a possible lattice for ternary compounds based on a specified degree of disorder. Therefore, the equations used in the programs that will achieve this goal will first be discussed.

Disorder in a lattice is a state of a crystalline solid in which some discrete lattice sites can be occupied by different atom types. In other words, a completely ordered lattice means that all of the atoms would be at their equilibrium locations, and a completely disordered lattice would have their atoms randomly distributed over the lattice sites. This degree of disorder can be quantified using S, which is the Bragg-Williams order parameter. S is defined as

$$S = r_\alpha + r_\beta - 1 \tag{1.1}$$

where r_α is the fraction of atom α is on the correct α atom site and r_β is the fraction of atom β is on the correct β atom site [1–3]. This project will only be using the stoichiometric case, which means that $r_\alpha = r_\beta$, so this equation is simplified to $S = 2r - 1$ in the program. The Bragg-Williams order parameter can have values between 0 and 1, where $S = 0$ corresponds to the completely disordered lattice (where $r_\alpha = r_\beta = 0.5$) and $S = 1$ corresponds to a completely ordered lattice (where $r_\alpha = r_\beta = 1$). Reflection high-energy electron diffraction, x-ray diffraction, scanning electron microscopy, and Raman spectroscopy can be used to measure the degree of disorder in a material [1,2].

A unit cell is typically used when representing a crystalline structure. However, it is easier to think of the structure made up of structural motifs when considering a disordered lattice. A structural motif is any of the possible arrangements of the nearest-neighbor bonding environment of a particularized atom. The number of possible arrangements depends on the material, however, every material has a specific base motif. A base motif is the only structural motif that would appear in a completely ordered lattice when the material is at a specified stoichiometry. Thus, as the degree of disorder in the material increases, the types of structural motifs that occur in the lattice also increases. In other words, the probability of the motifs occurring throughout the lattice is dependent on the degree of disorder. This probability has been calculated for a wide range of material types [2] including the ternary compounds that has been explored in this project:

$$\begin{aligned}
P_1 &= (r * 4)\\
P_2 &= 4 * (r * 2) * (1 - r) * 2\\
P_3 &= (1 - r) * 4\\
P_4 &= (r * 2) * (r - 1) * 2\\
P_5 &= 2 * (r * 3) * (1 - r)\\
P_6 &= 2 * r * (1 - r) * 3\\
P_7 &= (r * 2) * (r - 1) * 2\\
P_8 &= 2 * (r * 3) * (1 - r)\\
P_9 &= 2 * r * (1 - r) * 3
\end{aligned} \tag{1.2}$$

where r is the fraction of atoms on the correct site and $P_{\#}$ is the probability of each of the nine possible motifs for ternary compounds. All of these equations are used in the program in the Appendix.

1.2 Machine Learning

Machine learning is a sub-field of artificial intelligence where typically a large amount of data is provided to a machine to analyze and make a prediction. Machine learning is a predictive system that can have variable inputs and variable outputs. The large amount of data needed can be collected by sensors or open source datasets. However, semi-supervised and unsupervised machine learning models can be used if there are a lack of labeled data or data all together. These two learning algorithms can assist in labeling data and even creating more training data once a relatively smaller portion of data has been labeled and used to train a preliminary model [4].

2 Objectives

This project aims to use some initial data to train a machine learning model that will generate a possible lattice for ternary compounds based on a specified S^2 value. The following steps will be taken to achieve this:

1. Use VESTA [5] to generate the initial array that represents a completely ordered lattice.
2. Write a python code that can generate arrays that represent the $S = 0$ case.
3. Write a python code that can generate arrays that represent the $S = 0.5$ case.
4. Create a machine learning model that can assist in generating more arrays to represent other lattices at different S values.

The $S = 1$ and the $S = 0$ cases are both found first because they are the easiest to generate. This is because they only require a 64 motifs to accurately represent the lattice. The $S = 0.5$ case is the next easiest as it requires 256 motifs to represent the lattice. At this point, there may be enough motif arrays to begin training a machine learning model. All of these steps are explained further in the Methodology section.

3 Methodology

3.1 VESTA

The structural analysis program VESTA [5] was used to identify the array of each structural motif in the lattice. The first array that was established was the completely ordered array where $S = 1$. This was done first because the $S = 1$ case would only be constructed of the base motifs meaning that there would only be one possible array. The smallest possible repeating lattice needed for simulating the $S = 1$ case for $ZnSnN_2$ needs 128 atoms, which includes 64 nitrogen atoms and 32 zinc and tin atoms [2]. This ”unit cell” is shown in Fig. 3.1.

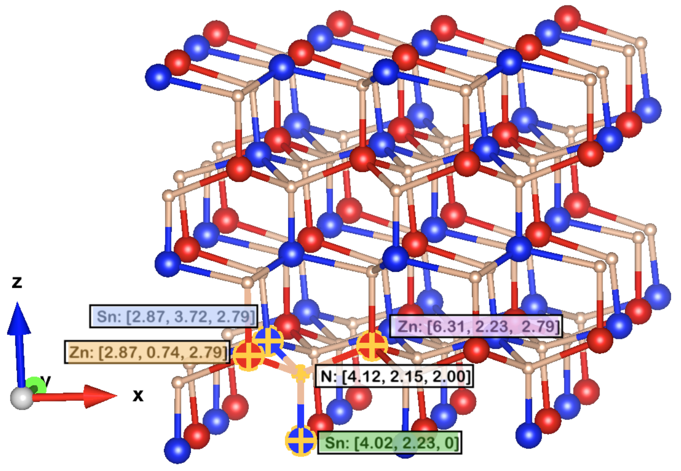
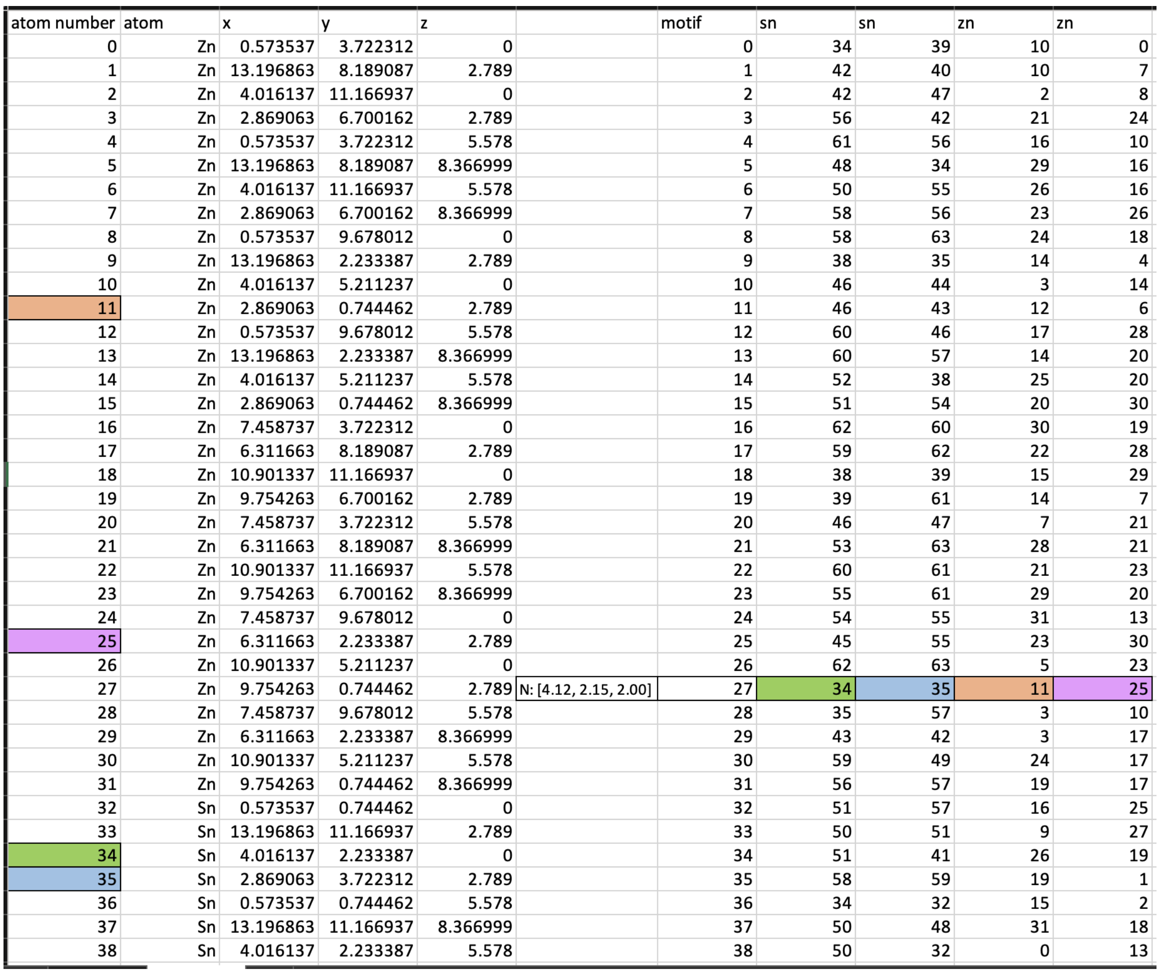


Figure 3.1: *The smallest repeating structure needed to simulate a completely ordered lattice for $ZnSnN_2$. The red atoms represent Zn, the blue atoms represent Sn, and the tan atoms represent N. The highlighted motif correlates to the motif used in Fig. 3.2.*

The xyz position data was extracted to determine the location of each of the atoms. Since the amount of and the locations of the Zn and Sn atoms around the nitrogen atom determine what type of motif it is, only the positions of the Zn and Sn atoms are of interest. Each of the Zn and Sn atoms were numbered 0 to 63. since this lattice structure has 64 total motifs (corresponding to each of the nitrogen atoms) the Zn and Sn atoms were then organized into motifs that were also labeled from 0 to 63. This is depicted in Fig. 3.2. This motif array is used in the python code (labeled ”motif”) shown in the Appendix to simulate the 128 atom lattice.



atom number	atom	x	y	z		motif	sn	sn	zn	zn
0	Zn	0.573537	3.722312	0		0	34	39	10	0
1	Zn	13.196863	8.189087	2.789		1	42	40	10	7
2	Zn	4.016137	11.166937	0		2	42	47	2	8
3	Zn	2.869063	6.700162	2.789		3	56	42	21	24
4	Zn	0.573537	3.722312	5.578		4	61	56	16	10
5	Zn	13.196863	8.189087	8.366999		5	48	34	29	16
6	Zn	4.016137	11.166937	5.578		6	50	55	26	16
7	Zn	2.869063	6.700162	8.366999		7	58	56	23	26
8	Zn	0.573537	9.678012	0		8	58	63	24	18
9	Zn	13.196863	2.233387	2.789		9	38	35	14	4
10	Zn	4.016137	5.211237	0		10	46	44	3	14
11	Zn	2.869063	0.744462	2.789		11	46	43	12	6
12	Zn	0.573537	9.678012	5.578		12	60	46	17	28
13	Zn	13.196863	2.233387	8.366999		13	60	57	14	20
14	Zn	4.016137	5.211237	5.578		14	52	38	25	20
15	Zn	2.869063	0.744462	8.366999		15	51	54	20	30
16	Zn	7.458737	3.722312	0		16	62	60	30	19
17	Zn	6.311663	8.189087	2.789		17	59	62	22	28
18	Zn	10.901337	11.166937	0		18	38	39	15	29
19	Zn	9.754263	6.700162	2.789		19	39	61	14	7
20	Zn	7.458737	3.722312	5.578		20	46	47	7	21
21	Zn	6.311663	8.189087	8.366999		21	53	63	28	21
22	Zn	10.901337	11.166937	5.578		22	60	61	21	23
23	Zn	9.754263	6.700162	8.366999		23	55	61	29	20
24	Zn	7.458737	9.678012	0		24	54	55	31	13
25	Zn	6.311663	2.233387	2.789		25	45	55	23	30
26	Zn	10.901337	5.211237	0		26	62	63	5	23
27	Zn	9.754263	0.744462	2.789	N: [4.12, 2.15, 2.00]	27	34	35	11	25
28	Zn	7.458737	9.678012	5.578		28	35	57	3	10
29	Zn	6.311663	2.233387	8.366999		29	43	42	3	17
30	Zn	10.901337	5.211237	5.578		30	59	49	24	17
31	Zn	9.754263	0.744462	8.366999		31	56	57	19	17
32	Sn	0.573537	0.744462	0		32	51	57	16	25
33	Sn	13.196863	11.166937	2.789		33	50	51	9	27
34	Sn	4.016137	2.233387	0		34	51	41	26	19
35	Sn	2.869063	3.722312	2.789		35	58	59	19	1
36	Sn	0.573537	0.744462	5.578		36	34	32	15	2
37	Sn	13.196863	11.166937	8.366999		37	50	48	31	18
38	Sn	4.016137	2.233387	5.578		38	50	32	0	13

Figure 3.2: *An example of how the motifs were organized into an array that represents the lattice. The highlighted cells correlate with the highlighted positions of the atoms in Fig. 3.1.*

The motif array described above would only represent a completely ordered array if the Zn and Sn atoms were in the correct location. Therefore, the Zn and Sn atoms need to be identified. The Zn atoms were labeled as 1 and the Sn atoms were labeled as 0. Therefore, for a completely ordered lattice, atom number 0-31 would be labeled as 1 and atom number 32-63 would be labeled as 0. This is a single row array labeled ″znsn″ in the first python code in the Appendix. This would also mean that for a completely ordered lattice, the motif array would end up with just one motif type for all 64 of its motifs: [0 0 1 1].

For the heterovalent ternary nitride materials such as $ZnSnN_2$, there are a total of nine different motif types [2]. To represent this in the python code, there was a specified single row array called the ″count_ array.″ This counted the number of each of the nine motif types. In a completely ordered array, the ″count_ array″ would equal [64 0 0 0 0 0 0 0 0], since all 64 motifs would be motif type 1. Once the zeros (Sn atoms) and ones (Zn atoms) in the ″znsn″ array begin to be switched around, other motif types will appear in the ″count_ array″.

3.2 Generating More Lattices

Now that the $S = 1$ arrays have been made by finding and organizing each motif and atom position, a python code can be created to generate arrays for the $S = 0$ case. There will be more than one ″znsn″ array that can represent the $S = 0$ case. To determine what ″znsn″ array qualifies, the probability equations (shown in equations 1.2) for each of the motif types will be used. When $S = 0$, $r = 0.5$ according to equation 1.1. Plugging in $r = 0.5$ into equations 1.2, the probability array of [4 16 4 4 8 8 4 8 8] is found. To find the correct ″znsn″ array to get the correct number of each motif type, a row array of 64 zeros and ones were randomly generated and ran through the python program in the Appendix until the ″count_ array″ equaled the ″prob_ array.″ This was done multiple times to find many ″znsn″ arrays.

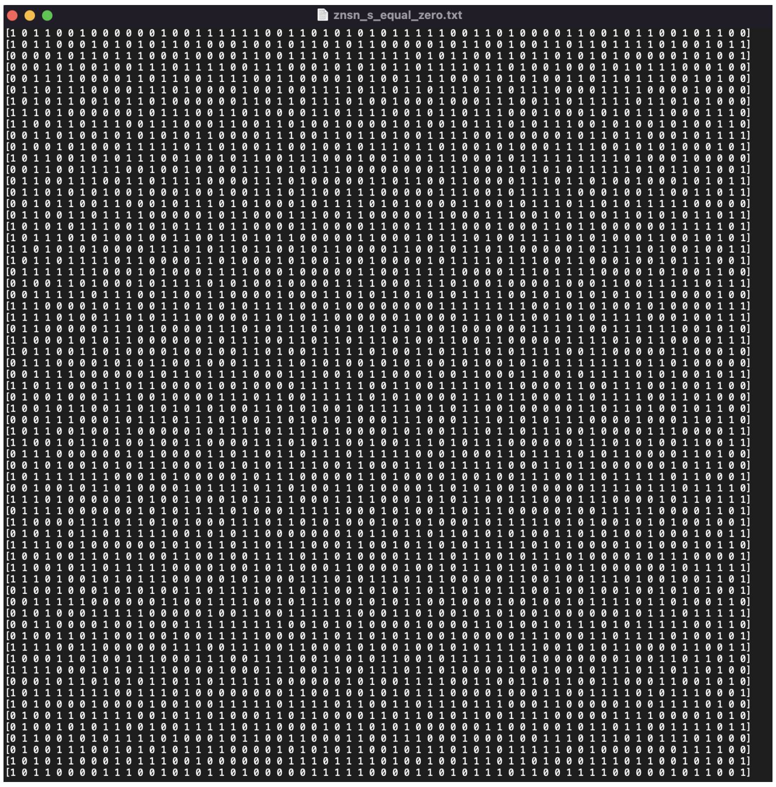
The $S = 0.5$ case was slightly more complex because a 512 atom lattice made up of 256 motifs needed to be used to accurately calculate this case. If $S = 0.5$, $r = 0.75$ and the ″prob_ array″ would equal [81 36 1 9 54 6 9 54 6]. Since this would certainly take a lot longer to generate the correct ″znsn″ row arrays now that they are 256 values long, an attempt was made to simplify the python code. It was realized that since we are only considering the stoichiometric case, the number of zeros (Sn atoms) and ones (Zn atoms) would have to be equal. Therefore, if the program generated an array that did not have an equal number of zeros and ones, it would not put that array through the code and that would speed up the processing time.

3.3 Machine Learning

Google's Colab was used to train the machine learning model. Google's Colab is free to use and allows people to execute python code to develop machine learning models using TensorFlow. It is especially useful because it allows free Cloud service with free GPU. Colab makes it relatively easy to learn about machine learning with several examples that are easily executed through a browser. Executing code is done by simply clicking on the play button that can be seen next to the code. Additionally, the code can be saved via GitHub or Google Drive, and the models and other item that are formed when you execute the code (such as images and graphs) can be downloaded to your own computer.

4 Results and Discussion

The python code would only generate about one $S = 0$ "znsn" array per hour. A total of 66 arrays were found before moving on to the other steps. They were stored in a text file for later use, as seen in Fig. 4.1

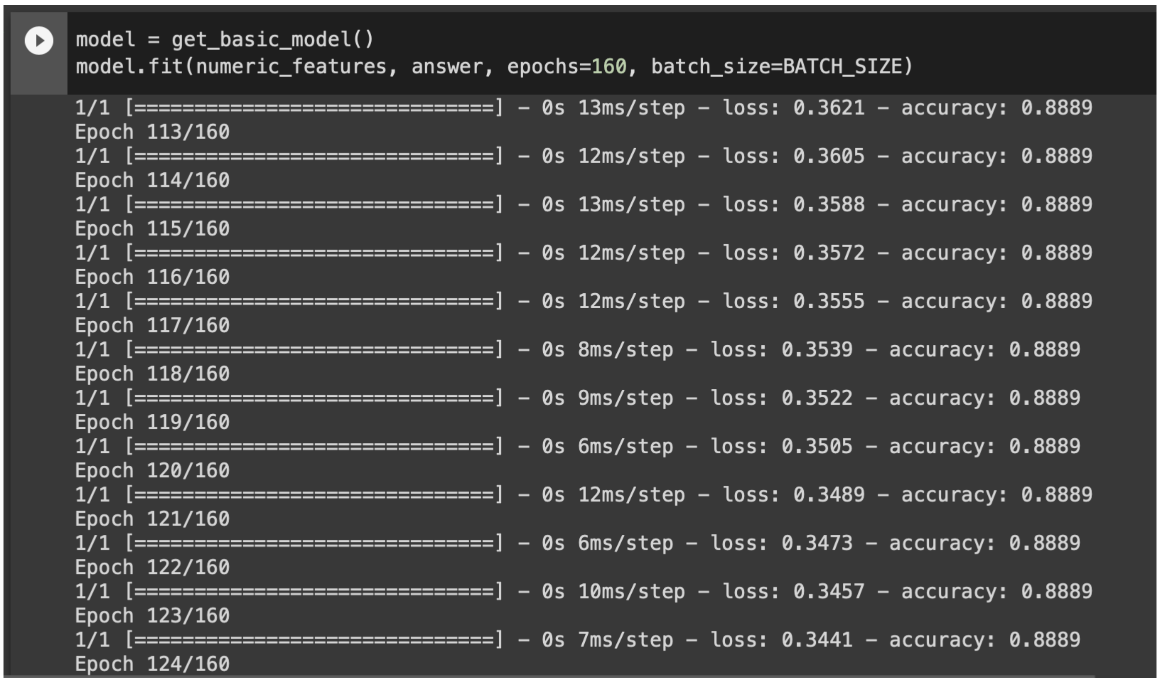


```
[1 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0 1 1 1 1 1 1 0 0 1 1 0 1 0 1 0 1 0 1 1 1 1 1 0 0 1 1 0 1 0 0 0 0 1 1 0 0 1 0 1 1 0 0 1 0 1 1 0 0]
[1 0 1 1 0 0 0 1 0 1 0 1 0 1 1 0 1 0 0 0 1 0 0 1 0 1 1 0 1 0 1 1 0 0 0 0 0 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1 0 0 1 0 1 1]
[0 0 0 0 1 0 1 1 0 1 1 1 0 0 0 1 0 0 0 0 1 1 0 0 1 1 1 0 1 1 1 1 1 1 1 0 1 0 1 1 0 0 1 1 0 1 1 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1]
[0 0 0 1 0 1 0 0 1 0 0 1 1 1 0 1 1 1 1 0 0 1 1 1 0 0 0 1 0 1 0 1 0 1 1 0 1 1 1 1 0 1 1 0 1 0 0 1 0 0 0 1 0 1 0 1 1 1 0 0 0 1 0 0]
[0 0 1 1 1 1 0 0 0 0 1 1 0 1 1 0 0 1 1 1 1 0 0 1 1 0 0 0 0 1 0 1 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0 1 1 0 1 1 0 1 1 1 0 0 1 0 1 0 0]
[0 1 1 0 1 1 1 0 0 0 0 1 1 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 0]
[1 0 1 0 1 1 0 0 1 0 1 1 0 1 0 0 0 0 0 0 1 1 0 1 1 0 1 1 1 0 1 0 0 1 0 0 0 1 0 0 0 1 1 1 0 0 1 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0 0]
[1 1 1 0 1 0 0 0 0 0 0 1 0 1 1 1 0 0 1 1 0 1 0 0 0 0 1 1 0 1 1 1 1 0 0 1 0 1 1 0 1 1 1 0 0 0 1 0 0 0 1 0 1 0 1 1 1 0 0 0 1 1 1 0]
[1 1 0 0 1 1 0 1 1 1 0 0 1 1 1 0 0 0 1 1 0 0 1 1 0 1 0 0 1 0 0 0 0 1 0 1 0 0 1 0 1 1 1 0 1 0 1 1 1 0 0 1 0 1 0 0 1 0 1 0 0 1 1 0]
[0 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 0 1 1 1 0 0 1 1 0 1 1 0 1 0 0 1 1 1 1 0 0 1 0 0 0 0 0 1 0 1 1 0 1 1 0 0 0 1 0 1 1 1 1]
[0 1 0 0 1 0 1 0 0 0 1 1 1 1 1 0 1 1 0 1 0 0 1 1 0 0 1 0 0 1 0 1 1 1 0 1 1 0 1 0 0 1 0 1 0 0 0 1 1 1 1 0 0 1 0 1 0 1 0 0 0 1 0 1]
[1 0 1 1 0 0 1 0 1 0 1 1 1 0 0 1 0 0 1 0 1 1 0 0 1 1 1 0 0 0 1 0 0 1 0 0 1 1 1 0 0 0 1 0 1 1 1 1 1 1 1 1 1 0 1 0 0 0 1 0 0 0 0 0]
[0 0 1 1 0 0 1 1 1 1 0 0 1 0 0 1 0 1 0 0 1 1 1 0 1 0 1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 0 1 0 1 1 1 1 1 0 1 0 1 1 0 1 0 0 1]
[0 1 1 0 0 1 1 1 0 0 1 1 0 1 1 1 1 0 0 0 0 1 1 1 0 1 0 0 0 0 0 1 1 0 1 1 0 0 1 1 0 0 0 0 1 1 1 0 1 1 0 1 0 0 1 0 0 0 1 0 1 0 1 1]
[0 1 1 0 1 0 1 0 1 0 0 1 0 0 0 1 0 0 1 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 1 0 1 1 1 1 1 0 0 0 1 0 0 1 1 0 0 1 1 0 1 1]
[0 0 1 0 1 1 0 0 1 1 0 0 0 1 0 1 1 1 0 1 0 1 0 0 0 1 0 1 1 1 1 0 1 0 1 0 0 0 0 1 1 0 0 1 0 1 1 1 0 1 1 0 1 0 1 1 1 1 1 0 0 0 0 0]
[0 1 1 0 0 1 1 0 1 1 1 1 0 0 0 0 0 1 0 1 1 0 0 0 1 1 1 0 0 1 1 0 0 0 0 0 1 1 0 0 0 1 1 1 0 0 1 0 1 1 0 0 1 1 0 1 0 1 0 1 1 0 1 1]
[1 0 1 0 1 0 1 1 1 0 0 1 0 1 0 1 1 1 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 1 1 1 1 0 0 0 1 0 0 0 1 1 0 1 1 0 0 0 0 0 0 1 1 1 1 1 0 1]
[1 0 1 1 1 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1 0 1 0 1 1 0 0 0 0 0 1 1 0 0 0 1 0 1 1 1 0 1 0 0 1 1 1 1 0 1 0 1 0 0 0 1 1 0 0 1 0 1 0 1]
[1 1 0 1 0 1 0 1 0 0 0 0 1 1 1 0 1 0 1 1 0 1 1 0 0 1 0 1 1 0 0 0 0 1 1 0 0 1 0 1 1 0 1 1 0 0 0 0 0 1 0 1 1 1 1 0 1 0 0 1 0 0 1 1]
[1 0 1 1 0 1 1 1 1 0 1 1 0 0 0 0 1 1 0 1 0 0 0 1 0 1 0 0 1 0 0 1 0 0 0 0 1 0 1 1 1 1 0 1 1 1 0 0 1 1 0 0 0 1 0 0 1 0 1 1 1 0 0 1]
[0 1 1 1 1 1 1 1 0 0 0 1 0 1 0 0 0 1 1 1 1 0 0 0 1 0 0 0 0 0 1 1 0 0 1 1 1 1 1 0 0 0 0 1 1 1 0 1 1 1 1 0 0 0 0 1 0 1 0 0 1 1 0 0]
[0 1 0 0 1 1 0 1 0 0 0 1 0 1 1 1 1 0 1 0 1 0 0 1 0 0 0 0 1 1 1 1 0 0 0 1 1 1 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1 1 1 1 1 0 1 1 1]
[0 0 1 1 1 1 1 0 1 1 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 1 0 1 1 0 1 0 1 0 1 1 1 1 0 0 1 0 1 0 1 0 1 0 1 0 1 1 0 0 0 0 1 0 0]
[1 1 1 0 0 0 0 1 0 1 1 0 0 1 1 0 1 1 0 1 0 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 1 1 1]
[1 1 1 1 0 1 0 0 1 1 0 1 0 1 1 1 0 0 0 0 0 1 1 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 1 1 0 0 1 0 1 0 1 1 1 1 0 0 0 1 0 0 1 1 1]
[0 1 1 0 0 0 0 0 1 1 1 0 1 0 0 0 0 1 1 1 0 1 0 1 1 1 0 1 0 1 0 1 0 1 0 1 0 0 1 0 0 0 0 0 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 1 0 1 0]
[1 1 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 1 0 1 1 1 0 0 1 1 0 1 1 0 1 1 1 0 1 0 0 1 1 0 0 1 1 0 0 0 0 0 1 1 1 1 0 1 0 1 1 0 0 0 0 1 1 1]
[1 0 1 1 0 0 1 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 0 1 0 0 1 1 1 1 1 0 1 0 0 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 1 1 0 0 0 0 0 1 1 0 0 0 1 0]
[0 1 1 1 0 0 0 0 1 0 1 0 1 1 0 0 1 0 0 0 1 1 1 1 1 1 0 1 0 1 0 0 1 0 1 0 1 0 0 1 0 1 0 0 1 0 1 0 1 1 1 1 1 1 1 1 0 1 1 0 1 0 0 0 0 0]
[0 0 1 1 1 1 0 0 0 0 0 0 1 0 1 1 0 1 1 1 1 0 0 0 1 1 1 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 0 0 1 1 0 0 1 0 1 1 1 1 0 1 0 1 0 0 1 0 1 1]
[1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 1 1 1 1 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 0 1 1 0 0 1 1 1 0 0 1 0 0 1 1 0 0]
[0 1 0 0 1 0 0 0 1 1 1 0 0 1 0 0 0 0 1 1 1 0 1 1 0 0 1 1 1 1 1 0 1 0 1 0 0 0 0 1 1 1 0 1 1 0 0 0 1 1 0 1 1 1 1 0 0 1 1 0 1 0 0 0]
[1 0 0 1 0 1 1 0 0 1 1 0 1 0 1 0 1 0 1 0 0 1 1 0 1 0 1 0 0 1 0 1 1 1 1 0 1 1 0 1 1 0 0 1 0 0 0 0 0 1 1 0 1 1 0 1 0 0 1 0 1 1 0 0]
[0 0 0 1 1 1 0 0 0 1 0 1 1 1 0 1 1 1 0 1 0 0 1 1 0 1 0 1 0 1 0 0 0 1 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 0 0 0 1 0 0 0 1 1 0 1 1 0]
[1 0 1 1 0 0 1 0 0 1 1 0 0 0 0 0 1 0 1 1 1 1 0 1 1 1 1 0 1 0 0 0 0 1 0 1 0 0 1 1 1 0 1 1 0 1 1 1 0 0 1 0 0 0 0 1 1 1 0 0 0 0 1 1]
[1 1 0 0 1 0 1 1 0 1 0 0 1 0 0 1 1 0 0 0 0 1 1 1 0 1 0 1 1 0 0 1 0 0 1 1 1 0 1 0 1 1 1 0 0 0 0 0 0 1 1 1 0 1 0 1 0 0 1 1 0 0 1 1]
[0 1 1 1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 0 0 0 1 0 0 0 1 1 0 1 1 1 1 0 1 0 1 1 0 0 0 0 1 1 0 1 0 0]
[0 0 1 0 1 0 0 1 0 1 0 1 1 1 0 0 0 1 0 1 0 1 0 1 1 1 1 0 0 1 1 0 0 0 1 1 1 0 1 1 1 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0 0 1 0 1 1 1 0 0]
[1 0 1 1 1 1 1 1 1 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1 1 0 0 0 0 0 1 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 1 0 0 1 1 0 1 1 1 1 1 1 0 1 1 0 0 0 1]
[0 0 1 0 0 1 0 1 1 0 1 0 0 0 0 1 0 1 1 1 1 0 1 1 0 1 0 0 1 1 0 1 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 0 0 1 1 1 1 0 1 1 1 0 1 1 1 1 0]
[1 1 1 0 1 0 0 0 0 0 1 0 1 0 0 1 0 0 0 1 0 1 1 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 1 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 1 0 1 1 0 1 1 1]
[0 1 1 1 1 0 0 0 0 0 0 1 0 1 0 1 1 1 1 0 1 0 0 0 1 1 1 1 1 0 0 0 1 0 1 0 0 1 1 0 1 1 1 0 0 0 0 0 1 0 0 1 1 1 1 1 0 0 0 0 1 1 0 1]
[1 1 0 0 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 1 0 1 1 0 1 0 1 0 0 0 1 0 1 0 1 0 0 0 1 0 1 1 1 1 1 1 0 1 0 1 0 0 1 0 1 0 1 0 0 1 0 0 1]
[0 1 0 1 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 1 0 1 1 0 1 1 0 1 1 0 1 0 1 0 1 0 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0 0 1 1]
[1 1 1 1 0 0 1 0 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 1 1 0 0 0 1 1 0 0 1 0 1 1 0 1 0 1 1 1 1 1 1 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 1 0]
[1 0 0 1 0 0 1 1 0 1 0 1 0 0 1 1 0 0 1 0 0 1 1 1 1 0 1 1 0 1 0 0 0 0 1 1 1 1 0 1 1 0 0 1 0 1 1 1 0 1 0 0 0 0 1 0 1 1 1 0 0 0 0 1]
[1 1 0 0 1 0 1 1 0 1 1 1 1 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 1 0 0 0 0 1 0 0 1 0 1 1 1 0 1 1 0 1 0 0 1 1 0 0 0 0 0 0 1 0 1 1 1 1 1]
[1 1 1 0 1 0 0 1 0 1 0 1 1 1 0 0 0 0 0 1 0 1 0 0 0 1 1 1 0 1 0 1 1 1 0 1 1 1 0 0 0 0 0 1 1 0 0 1 0 0 1 1 1 0 1 0 0 1 0 0 1 1 0 1]
[0 1 0 0 1 0 0 0 1 0 1 0 1 0 0 1 1 0 1 1 1 1 0 1 1 0 1 1 0 0 1 0 1 0 1 1 0 1 1 0 1 0 1 1 1 0 0 1 0 0 1 0 0 1 0 0 1 0 1 0 1 0 0 1]
[0 0 1 1 1 1 1 0 0 0 0 0 0 1 1 0 0 1 1 1 1 0 0 1 0 1 1 1 0 0 1 0 1 0 1 1 0 0 1 0 0 0 1 0 0 1 0 0 1 0 1 1 1 1 0 1 1 0 1 0 0 1 1 0]
[0 1 0 1 0 0 0 1 1 1 1 1 0 0 0 0 0 1 0 0 1 1 0 0 1 1 1 1 1 1 0 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 1 0 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1]
[0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 0 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 0]
[0 1 0 0 1 1 0 1 1 0 0 1 0 0 1 0 0 1 1 1 1 1 0 0 0 0 1 1 0 1 1 0 1 0 1 1 0 0 1 0 0 0 0 0 1 1 1 0 0 0 1 1 0 1 1 1 1 0 1 0 0 1 0 1]
[1 1 1 1 0 0 1 1 0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 1 1 0 0 0 1 0 1 0 0 1 0 0 1 0 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 0 0 0 0 1 1 0 0 1 1]
[1 0 0 0 1 1 0 1 0 0 1 1 1 0 0 0 1 1 1 0 0 1 1 1 1 0 0 1 0 0 1 0 1 1 0 0 1 0 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 0 1 0]
[1 1 1 1 0 0 0 1 0 1 0 1 1 1 0 0 0 0 1 0 0 0 1 1 1 0 0 1 1 0 0 1 1 1 0 1 1 0 1 0 0 0 0 1 0 0 1 0 0 1 0 1 1 1 0 1 1 0 1 1 0 1 0 0]
[0 0 0 1 0 1 1 0 1 0 1 0 1 1 0 1 1 0 1 1 1 1 1 0 0 0 0 0 0 1 0 1 0 0 1 1 1 1 0 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 0 0 1 1 0 0 1 0 1 0]
[1 0 1 1 1 1 1 1 1 0 0 1 1 1 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 1 0 1 1 1 0 0 0 0 1 0 0 0 1 1 0 0 1 1 1 1 0 1 0 1 1 1 0 0 0 1]
[1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 1 1 1 1 1 0 1 1 1 1 0 1 1 0 0 1 1 0 1 1 1 0 1 1 1 0 0 0 1 1 0 0 0 0 0 1 0 0 1 1 1 1 0 1 0 0]
[0 1 0 0 1 1 0 1 1 1 1 0 0 1 0 1 1 0 1 0 0 0 1 1 0 1 1 0 0 0 0 1 1 0 1 1 0 1 0 1 1 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 1 0]
[0 1 0 0 1 0 1 0 1 1 0 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 1 1 0 1 1 0 0 1 1 1 1 0 1 1]
[0 1 1 0 0 1 0 1 0 1 1 1 1 0 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 0 0 1 1 1 0 0 0 1 0 0 0 1 0 0 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 0 0 0]
[0 1 0 0 1 1 1 0 0 1 0 1 0 1 0 1 1 1 1 0 0 0 0 0 1 0 1 0 1 0 1 1 0 0 1 0 1 1 1 0 1 0 1 0 1 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0]
[1 0 1 0 1 1 0 0 0 1 0 1 1 1 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0 0 1 1 1 0 1 1 0 1 0 0 0 1 1 1 0 1 1 0 0 1 0 1 0 0 1]
[1 0 1 1 0 0 0 0 1 1 1 0 0 1 0 1 0 1 1 0 1 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 0 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 0 0 0 0 0 1 0 1 1 0 0 1]
```

Figure 4.1: *The 66 "znsn" arrays that were originally found for the $S = 0$ case.*

The python code used to try to generate the $S = 0.5$ cases was set up on a lab computer since it was taking a long time to generate the correct "znsn" arrays. Even with the simpler code shown in the Appendix, it still would not output an array. It ran for multiple days, but unfortunately did not give an array that satisfied the $S = 0.5$ case.

Since none of the $S = 0.5$ arrays were generated, an attempt was made to train a machine learning model to determine whether a ”znsn” array would have an S value equal to zero or an S value equal to anything else. This was done by feeding the machine learning model $S = 0$ arrays and arrays that did not give an $S = 0$ value. Once training the model, the hope was that it would be able to tell whether a ”znsn” array could represent a lattice that was completely disordered. Unfortunately, no matter what array the trained models were given, most of the time, it would predict a value of zero. Some models predicted fractions, which was also incorrect. This was even when the model claimed an accuracy of around 89%, as seen in Fig. 4.2. The machine learning model that was supposedly 89% accurate can be found in the Appendix.



```
model = get_basic_model()
model.fit(numeric_features, answer, epochs=160, batch_size=BATCH_SIZE)
```

```
1/1 [==============================] - 0s 13ms/step - loss: 0.3621 - accuracy: 0.8889
Epoch 113/160
1/1 [==============================] - 0s 12ms/step - loss: 0.3605 - accuracy: 0.8889
Epoch 114/160
1/1 [==============================] - 0s 13ms/step - loss: 0.3588 - accuracy: 0.8889
Epoch 115/160
1/1 [==============================] - 0s 12ms/step - loss: 0.3572 - accuracy: 0.8889
Epoch 116/160
1/1 [==============================] - 0s 12ms/step - loss: 0.3555 - accuracy: 0.8889
Epoch 117/160
1/1 [==============================] - 0s 8ms/step - loss: 0.3539 - accuracy: 0.8889
Epoch 118/160
1/1 [==============================] - 0s 9ms/step - loss: 0.3522 - accuracy: 0.8889
Epoch 119/160
1/1 [==============================] - 0s 6ms/step - loss: 0.3505 - accuracy: 0.8889
Epoch 120/160
1/1 [==============================] - 0s 12ms/step - loss: 0.3489 - accuracy: 0.8889
Epoch 121/160
1/1 [==============================] - 0s 6ms/step - loss: 0.3473 - accuracy: 0.8889
Epoch 122/160
1/1 [==============================] - 0s 10ms/step - loss: 0.3457 - accuracy: 0.8889
Epoch 123/160
1/1 [==============================] - 0s 7ms/step - loss: 0.3441 - accuracy: 0.8889
Epoch 124/160
```

Figure 4.2: *The model being trained in Google’s Colab. The model will repeatably run the training steps depending on the number of epochs you assign. The accuracy and validation accuracy typically improves after each round of training.*

5 Conclusions and Further Steps

The machine learning model will need more data to be able to properly train it. The code that was used to find the ″znsn″ arrays that gave us $S = 0$ can be run for a longer time to find more arrays since there is more than the 66 that were found. However, without more arrays that give other S values, the machine learning model will not be properly trained. Therefore, to be able to generate some initial $S = 0.5$ arrays, the python program will now be set up on GPU. This will allow the program to run faster and hopefully output a ″znsn″ array that achieves $S = 0.5$.

Additionally, different machine learning algorithms could be attempted. The algorithm used for this project was a type of classification algorithm. Although a classification algorithm is the correct one to use in this case, there are many different types of classification algorithms that I have yet to explore. When reviewing the classification algorithm used for this project, it seems that it works best with larger datasets. There are other algorithms that are more useful when only smaller datasets are available [4].

6 Appendix: Code

6.1 Determining the S=0 Arrays

```
import array as arr
import numpy as np

def count_mt():

    motif = np.array([[34,39,10,0],[42,40,10,7],[42,47,2,8],[56,42,21,24],
    [61,56,16,10],[48,34,29,16],[50,55,26,16],[58,56,23,26],[58,63,24,18],
    [38,35,14,4],[46,44,3,14],[46,43,12,6],[60,46,17,28],[60,57,14,20],
    [52,38,25,20],[51,54,20,30],[62,60,30,19],[59,62,22,28],[38,39,15,29],
    [39,61,14,7],[46,47,7,21],[53,63,28,21],[60,61,21,23],[55,61,29,20],
    [54,55,31,13],[45,55,23,30],[62,63,5,23],[34,35,11,25],[35,57,3,10],
    [43,42,3,17],[59,49,24,17],[56,57,19,17],[51,57,16,25],[50,51,9,27],
    [51,41,26,19],[58,59,19,1],[34,32,15,2],[50,48,31,18],[50,32,0,13],
    [45,40,0,26],[40,58,5,8],[37,32,8,18],[38,36,6,11],[54,52,22,27],
    [54,36,4,9],[44,41,30,4],[44,62,12,1],[33,36,12,22],[39,45,13,4],
    [44,45,7,5],[47,37,12,5],[47,53,15,6],[63,37,31,22],[32,33,11,9],
    [49,48,25,27],[35,41,0,9],[41,40,3,1],[43,33,1,8],[49,43,11,2],
    [33,59,27,18],[53,48,24,2],[36,37,13,15],[49,52,6,28],[52,53,31,29]])

    znsn = np.array([1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0])

    rows = len(znsn[motif])
    cols = len(motif[0])

    mt1=np.array([0,0,1,1])
    mt2_1=np.array([1,0,0,1])
    mt2_2=np.array([0,1,1,0])
    mt2_3=np.array([1,0,1,0])
    mt2_4=np.array([0,1,0,1])
    mt3=np.array([1,1,0,0])
    mt4=np.array([0,0,0,0])
    mt5_1=np.array([0,0,0,1])
    mt5_2=np.array([0,0,1,0])
    mt6_1=np.array([0,1,0,0])
    mt6_2=np.array([1,0,0,0])
    mt7=np.array([1,1,1,1])
    mt8_1=np.array([1,0,1,1])
    mt8_2=np.array([0,1,1,1])
    mt9_1=np.array([1,1,1,0])
    mt9_2=np.array([1,1,0,1])

    for i in range(800000):
        mt1_count=0
        mt2_count=0
        mt3_count=0
        mt4_count=0
```

```
mt5_count=0
mt6_count=0
mt7_count=0
mt8_count=0
mt9_count=0
for i in range(rows):
    current_motif_indexes = motif[i,:]
    current_motif = np.array([znsn[current_motif_indexes[0]], znsn[current_motif_
    indexes[1]],znsn[current_motif_indexes[2]],znsn[current_motif_indexes[3]]])
    if np.array_equal(current_motif, mt1):
        mt1_count+=1
    elif np.array_equal(current_motif, mt2_1):
        mt2_count+=1
    elif np.array_equal(current_motif, mt2_2):
        mt2_count+=1
    elif np.array_equal(current_motif, mt2_3):
        mt2_count+=1
    elif np.array_equal(current_motif, mt2_4):
        mt2_count+=1
    elif np.array_equal(current_motif, mt3):
        mt3_count+=1
    elif np.array_equal(current_motif, mt4):
        mt4_count+=1
    elif np.array_equal(current_motif, mt5_1):
        mt5_count+=1
    elif np.array_equal(current_motif, mt5_2):
        mt5_count+=1
    elif np.array_equal(current_motif, mt6_1):
        mt6_count+=1
    elif np.array_equal(current_motif, mt6_2):
        mt6_count+=1
    elif np.array_equal(current_motif, mt7):
        mt7_count+=1
    elif np.array_equal(current_motif, mt8_1):
        mt8_count+=1
    elif np.array_equal(current_motif, mt8_2):
        mt8_count+=1
    elif np.array_equal(current_motif, mt9_1):
        mt9_count+=1
    elif np.array_equal(current_motif, mt9_2):
        mt9_count+=1
count_array=np.array([mt1_count,mt2_count,mt3_count,mt4_count,
mt5_count,mt6_count,mt7_count,mt8_count,mt9_count])
#print(count_array)

#uncomment the next line and
#comment the S=0 line if you would like to manually input a S value
#S = float(input("Enter S value: "))
S=0
r=(S+1)/2

P_1=(r**4)
P_2=4*(r**2)*(1-r)**2
P_3=(1-r)**4
P_4=(r**2)*(1-r
)**2
        P_5=2*(r**3)*(1-r)
        P_6=2*r*(1-r)**3
        P_7=(r**2)*(1-r
)**2
        P_8=2*(r**3)*(1-r)
        P_9=2*r*(1-r)**3

        P_1=P_1*rows
        P_2=P_2*rows
        P_3=P_3*rows
        P_4=P_4*rows
        P_5=P_5*rows
        P_6=P_6*rows
        P_7=P_7*rows
        P_8=P_8*rows
        P_9=P_9*rows

        prob_array=np.array([P_1,P_2,P_3,P_4,P_5,P_6,P_7,P_8,P_9], dtype=int)
        #print(prob_array)

        if np.array_equal(prob_array[0], count_array[0]):
            if np.array_equal(prob_array[1], count_array[1]):
                if np.array_equal(prob_array[2], count_array[2]):
                    if np.array_equal(prob_array[3], count_array[3]):
                        if np.array_equal(prob_array[4], count_array[4]):
                            if np.array_equal(prob_array[5], count_array[5]):
                                if np.array_equal(prob_array[6], count_array[6]):
                                    if np.array_equal(prob_array[7], count_array[7]):
                                        if np.array_equal(prob_array[8], count_array[8]):
                                            print(np.array(znsn))
                                            print("successful array")
                                            count_mt()
                                        else:
                                            znsn=np.random.randint(0,2,size=64)
                                            print(np.array(znsn))
                                    else:
                                        znsn=np.random.randint(0,2,size=64)
                                        print(np.array(znsn))
                                else:
                                    znsn=np.random.randint(0,2,size=64)
                                    print(np.array(znsn))
                            else:
                                znsn=np.random.randint(0,2,size=64)
                                print(np.array(znsn))
                        else:
                            znsn=np.random.randint(0,2,size=64)
                            print(np.array(znsn))
                    else:
                        znsn=np.random.randint(0,2,size=64)
                        print(np.array(znsn))
                else:
                    znsn=np.random.randint(0,2,size=64)
                    print(np.array(znsn))
            else:
                znsn=np.random.randint(0,2,size=64)
                print(np.array(znsn))
        else:
            znsn=np.random.randint(0,2,size=64)
            print(np.array(znsn))

count_mt()
```

6.2 Determining the S=0.5 Arrays

```
import array as arr
import numpy as np

def count_mt(array_to_check):

    motif_2 = np.array([[34,39,10,0],[42,40,10,7],[42,47,2,8],[56,42,21,24],
    [61,56,16,10],[48,34,29,16],[50,55,26,16],[58,56,23,26],[58,63,24,18],
    [38,35,14,4],[46,44,3,14],[46,43,12,6],[60,46,17,28],[60,57,14,20],
    [52,38,25,20],[51,54,20,30],[62,60,30,19],[59,62,22,28],[38,39,15,29],
    [39,61,14,7],[46,47,7,21],[53,63,28,21],[60,61,21,23],[55,61,29,20],
    [54,55,31,13],[45,55,23,30],[62,63,5,23],[34,35,11,25],[35,57,3,10],
    [43,42,3,17],[59,49,24,17],[56,57,19,17],[51,57,16,25],[50,51,9,27],
    [51,41,26,19],[58,59,19,1],[34,32,15,2],[50,48,31,18],[50,32,0,13],
    [45,40,0,26],[40,58,5,8],[37,32,8,18],[38,36,6,11],[54,52,22,27],
    [54,36,4,9],[44,41,30,4],[44,62,12,1],[33,36,12,22],[39,45,13,4],
    [44,45,7,5],[47,37,12,5],[47,53,15,6],[63,37,31,22],[32,33,11,9],
    [49,48,25,27],[35,41,0,9],[41,40,3,1],[43,33,1,8],[49,43,11,2],
    [33,59,27,18],[53,48,24,2],[36,37,13,15],[49,52,6,28],[52,53,31,29],
    [98,103,74,64],[106,104,74,71],[106,111,66,72],[120,106,85,88],
    [125,120,80,74],[112,98,93,80],[114,119,90,80],[122,120,87,90],
    [122,127,88,82],[102,99,78,68],[110,108,67,78],[110,107,76,70],
    [124,110,81,92],[124,121,78,84],[116,102,89,84],[115,118,84,94],
    [126,124,94,83],[123,126,86,92],[102,103,79,93],[103,125,78,71],
    [110,111,71,85],[117,127,92,85],[124,125,85,87],[119,125,93,84],
    [118,119,95,77],[109,119,87,94],[126,127,69,87],[98,99,75,89],
    [99,121,67,74],[107,106,67,81],[123,113,88,81],[120,121,83,81],
    [115,121,80,89],[114,115,73,91],[115,105,90,83],[122,123,83,65],
    [98,96,79,66],[114,112,95,82],[114,96,64,77],[109,104,64,90],
    [104,122,69,72],[101,96,72,82],[102,100,70,75],[118,116,86,91],
    [118,100,68,73],[108,105,94,68],[108,126,76,65],[97,100,76,86],
    [103,109,77,68],[108,109,71,69],[111,101,76,69],[111,117,79,70],
    [127,101,95,86],[96,97,75,73],[113,112,89,91],[99,105,64,73],
    [105,104,67,65],[107,97,65,72],[113,107,75,66],[97,123,91,82],
    [117,112,88,66],[100,101,77,79],[113,116,70,92],[116,117,95,93],
    [162,167,138,128],[170,168,138,135],[170,175,130,136],
    [184,170,149,152],[189,184,144,138],[176,162,157,144],
    [178,183,154,144],[186,184,151,154],[186,191,152,146],
    [166,163,142,132],[174,172,131,142],[174,171,140,134],
    [188,174,145,156],[188,185,142,148],[180,166,153,148],
    [179,182,148,158],[190,188,158,147],[187,190,150,156],
    [166,167,143,157],[167,189,142,135],[174,175,135,149],
    [181,191,156,149],[188,189,149,151],[183,189,157,148],
    [182,183,159,141],[173,183,151,158],[190,191,133,151],
    [162,163,139,153],[163,185,131,138],[171,170,131,145],
[187,177,152,145],[184,185,147,145],[179,185,144,153],
[178,179,137,155],[179,169,154,147],[186,187,147,129],
[162,160,143,130],[178,176,159,146],[178,160,128,141],
[173,168,128,154],[168,186,133,136],[165,160,136,146],
[166,164,134,139],[182,180,150,155],[182,164,132,137],
[172,169,158,132],[172,190,140,129],[161,164,140,150],
[167,173,141,132],[172,173,135,133],[175,165,140,133],
[175,181,143,134],[191,165,159,150],[160,161,139,137],
[177,176,153,155],[163,169,128,137],[169,168,131,129],
[171,161,129,136],[177,171,139,130],[161,187,155,146],
[181,176,152,130],[164,165,141,143],[177,180,134,156],
[180,181,159,157],[226,231,202,192],[234,232,202,199],
[234,239,194,200],[248,234,213,216],[253,248,208,202],
[240,226,221,208],[242,247,218,208],[250,248,215,218],
[250,255,216,210],[230,227,206,196],[238,236,195,206],
[238,235,204,198],[252,238,209,220],[252,249,206,212],
[244,230,217,212],[243,246,212,222],[254,252,222,211],
[251,254,214,220],[230,231,207,221],[231,253,206,199],
[238,239,199,213],[245,255,220,213],[252,253,213,215],
[247,253,221,212],[246,247,223,205],[237,247,215,222],
[254,255,197,215],[226,227,203,217],[227,249,195,202],
[235,234,195,209],[251,241,216,209],[248,249,211,209],
[243,249,208,217],[242,243,201,219],[243,233,218,211],
[250,251,211,193],[226,224,207,194],[242,240,223,210],
[242,224,192,205],[237,232,192,218],[232,250,197,200],
[229,224,200,210],[230,228,198,203],[246,244,214,219],
[246,228,196,201],[236,233,222,196],[236,254,204,193],
[225,228,204,214],[231,237,205,196],[236,237,199,197],
[239,229,204,197],[239,245,207,198],[255,229,223,214],
[224,225,203,201],[241,240,217,219],[227,233,192,201],
[233,232,195,193],[235,225,193,200],[241,235,203,194],
[225,251,219,210],[245,240,216,194],[228,229,205,207],
[241,244,198,220],[244,245,223,221]])

rows = len(array_to_check[motif_2])
cols = len(motif_2[0])

mt1=np.array([0,0,1,1])
mt2_1=np.array([1,0,0,1])
mt2_2=np.array([0,1,1,0])
mt2_3=np.array([1,0,1,0])
mt2_4=np.array([0,1,0,1])
mt3=np.array([1,1,0,0])
mt4=np.array([0,0,0,0])
mt5_1=np.array([0,0,0,1])
mt5_2=np.array([0,0,1,0])
mt6_1=np.array([0,1,0,0])
mt6_2=np.array([1,0,0,0])
mt7=np.array([1,1,1,1])
mt8_1=np.array([1,0,1,1])
mt8_2=np.array([0,1,1,1])
mt9_1=np.array([1,1,1,0])
mt9_2=np.array([1,1,0,1])

mt1_count=0
```

```
    mt2_count=0
    mt3_count=0
    mt4_count=0
    mt5_count=0
    mt6_count=0
    mt7_count=0
    mt8_count=0
    mt9_count=0
    for i in range(rows):
        current_motif_indexes = motif_2[i,:]
        current_motif = np.array([array_to_check[current_motif_indexes[0]],array_to_check
        [current_motif_indexes[1]],array_to_check[current_motif_indexes[2]],
        array_to_check[current_motif_indexes[3]]])
        if np.array_equal(current_motif, mt1):
            mt1_count+=1
        elif np.array_equal(current_motif, mt2_1):
            mt2_count+=1
        elif np.array_equal(current_motif, mt2_2):
            mt2_count+=1
        elif np.array_equal(current_motif, mt2_3):
            mt2_count+=1
        elif np.array_equal(current_motif, mt2_4):
            mt2_count+=1
        elif np.array_equal(current_motif, mt3):
            mt3_count+=1
        elif np.array_equal(current_motif, mt4):
            mt4_count+=1
        elif np.array_equal(current_motif, mt5_1):
            mt5_count+=1
        elif np.array_equal(current_motif, mt5_2):
            mt5_count+=1
        elif np.array_equal(current_motif, mt6_1):
            mt6_count+=1
        elif np.array_equal(current_motif, mt6_2):
            mt6_count+=1
        elif np.array_equal(current_motif, mt7):
            mt7_count+=1
        elif np.array_equal(current_motif, mt8_1):
            mt8_count+=1
        elif np.array_equal(current_motif, mt8_2):
            mt8_count+=1
        elif np.array_equal(current_motif, mt9_1):
            mt9_count+=1
        elif np.array_equal(current_motif, mt9_2):
            mt9_count+=1
    count_array=np.array([mt1_count,mt2_count,mt3_count,mt4_count,
    mt5_count,mt6_count,mt7_count,mt8_count,mt9_count])

    S=0.5
    r=(S+1)/2
    P_1=(r**4)
    P_2=4*(r**2)*(1-r)**2
    P_3=(1-r)**4
    P_4=(r**2)*(1-r
)**2
```

```
    P_5=2*(r**3)*(1-r)
    P_6=2*r*(1-r)**3
    P_7=(r**2)*(1-r
)**2
    P_8=2*(r**3)*(1-r)
    P_9=2*r*(1-r)**3

    P_1=P_1*rows
    P_2=P_2*rows
    P_3=P_3*rows
    P_4=P_4*rows
    P_5=P_5*rows
    P_6=P_6*rows
    P_7=P_7*rows
    P_8=P_8*rows
    P_9=P_9*rows

    prob_array=np.array([P_1,P_2,P_3,P_4,P_5,P_6,P_7,P_8,P_9], dtype=int)

    if np.array_equal(prob_array, count_array):
        print(np.array(array_to_check))
        print("successful array")

for i in range(10000000):
    rand_gen = np.random.randint(0,2,size=256)
    checkStoich = sum(rand_gen)
    if checkStoich == len(rand_gen)/2:
        count_mt(rand_gen)
```

6.3 Machine Learning Model

```
import pandas as pd
import tensorflow as tf
import itertools
import os

SHUFFLE_BUFFER = 10000
BATCH_SIZE = 270

csv_file = tf.keras.utils.get_file('znsn_s_equal_more2.csv',
'https://raw.githubusercontent.com/krystalyork1696/s_equals_zero/main/znsn_s_equal_more2.csv')
df = pd.read_csv(csv_file)
df.head()

answer = df.pop('answer')
numeric_feature_names = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13',
 '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29',
 '30', '31', '32', '33', '34', '35', '36', '37', '38', '39', '40', '41', '42', '43', '44', '45',
 '46', '47', '48', '49', '50', '51', '52', '53', '54', '55', '56', '57', '58', '59', '60', '61',
 '62', '63', '64']
numeric_features = df[numeric_feature_names]
tf.convert_to_tensor(numeric_features)

normalizer = tf.keras.layers.Normalization(axis=-1)
```

```
normalizer.adapt(numeric_features)
normalizer(numeric_features.iloc[:3])

def get_basic_model():
  model = tf.keras.Sequential([
    normalizer,
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(1)
  ])

  model.compile(optimizer='adam',
                loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
                metrics=['accuracy'])
  return model

model = get_basic_model()
model.fit(numeric_features, answer, epochs=160, batch_size=BATCH_SIZE)

test_url = "https://raw.githubusercontent.com/krystalyork1696/s_equals_zero/main/znsn_test5.csv"
test_fp = tf.keras.utils.get_file(fname=os.path.basename(test_url), origin=test_url)

names=['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17',
 '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30', '31', '32', '33',
 '34', '35', '36', '37', '38', '39', '40', '41', '42', '43', '44', '45', '46', '47', '48', '49',
 '50', '51', '52', '53', '54', '55', '56', '57', '58', '59', '60', '61', '62', '63', '64', 'answer']

def pack_features_vector(features, labels):
  """Pack the features into a single array."""
  features = tf.stack(list(features.values()), axis=1)
  return features, labels

batch_size = 9
test_dataset = tf.data.experimental.make_csv_dataset(
    test_fp,
    batch_size,
    names,
    label_name='answer',
    num_epochs=9,
    shuffle=False)

test_dataset = test_dataset.map(pack_features_vector)
test_accuracy = tf.keras.metrics.Accuracy()

for (x, y) in test_dataset:
  # training=False is needed only if there are layers with different
  # behavior during training versus inference (e.g. Dropout).
  logits = model(x, training=False)
  prediction = tf.argmax(logits, axis=1, output_type=tf.int32)
  test_accuracy(prediction, y)

print("Test set accuracy: {:.3%}".format(test_accuracy.result()))
print(prediction)
```

Bibliography

[1] R. A. Makin, K. York, S. M. Durbin, N. Senabulya, J. Mathis, R. Clarke, N. Feldberg, P. Miska, C. M. Jones, Z. Deng, L. Williams, E. Kioupakis, and R. J. Reeves, “Alloy-Free Band Gap Tuning across the Visible Spectrum,” *Physical Review Letters*, vol. 122, p. 256403, June 2019.

[2] R. A. Makin, K. York, S. M. Durbin, and R. J. Reeves, “Revisiting semiconductor band gaps through structural motifs: An Ising model perspective,” *Physical Review B*, vol. 102, p. 115202, Sept. 2020.

[3] B. Warren, *X-Ray Diffraction, Dover Books on Physics*. Dover Books on Physics, Dover Publications, 2012.

[4] A. Dey, “Machine Learning Algorithms: A Review,” vol. 7, p. 6, 2016.

[5] K. MOMMA, “VESTA,” 2018.