Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Reproduction and Behaviour of Local and Non-local Distribution

Bachelor's Thesis

Fabian Auf der Maur

auffabia@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Prof. Dr. Roger Wattenhofer
Yuyi Wang, Ye Wang

June 14, 2021

Acknowledgements

I would like to thank firstly the Distributed Computing Group at ETH under Professor Wattenhofer and to my two supervisors for introducing me to this worthwhile and exciting, but also challenging topic. My thanks go also to my supervisors Yuyi Wang and Ye Wang for helping me with this subject.
Additionally, I'd like to thank Tamás Kriváchy for explaining to me the set-up and background of the here used Fritz distribution.
Another thanks go to my family for their support and motivation. Furthermore, I am grateful to my cat for not destroying my thesis by lying on the keyboard, it really saved me hours of work.

Abstract

Quantum distributions cannot be reproduced classically. This sentence has already been proven with a neural network in [1] to be true, leading to the question of how else can such a distribution been reproduced by a computer. In order to achieve that, we have implemented a way of quantum calculation in the code. This way the computer just has to figure out what he has to use as a basis for the quantum distribution in order to reproduce it. Thanks to this successfully implemented ansatz, we could not just reproduce a quantum distribution, but also observe the circumstance that different basis can lead to the exact same distribution.
Furthermore, we have used a neural network (a modified version from [1]) to approximate a reduced quantum distribution, which is known to be on the brink of being nonlocal. After we have successfully reproduced this distribution, we investigated its proximity to the nonlocal area. By introducing some noise we could show how close this distribution is on the border and that already a small part of noise can set it into the nonlocal area and thus cannot be reproduced by a neural network. Furthermore, we have observed that a small network can approximate the distribution much better than a big network, even when we added noise to the original distribution. Lastly, we have tested the robustness of this dispersion by adding some complete random noise and let this new distribution being reproduced by the neural network. Doing this, we have observed that the neural network can approximate the dispersion, consisting of random noise, well and thus is local. Following that, only the above introduced specific noise can stir the distribution to the nonlocal area.

Contents

Acknowledgements i

Abstract ii

1 Introduction 1

1.1 Motivation . . . 1

1.2 Related Work . . . 2

2 Background 3

2.1 Quantum Distribution Calculation . . . 3

2.1.1 Fritz Distribution . . . 3

2.1.2 LLL Distribution . . . 4

2.2 Neural Network . . . 5

3 Implementation and Reproduction of the Fritz Distribution 8

3.1 Quantum Calculation by Python . . . 8

3.1.1 Condition of Alice, Bob and Charlie . . . 8

3.1.2 Calculation of a Measurement . . . 9

3.1.3 Setting up the Distribution . . . 10

3.2 Reproduction . . . 11

3.3 Training Python to do Quantum Math by Himself . . . 14

4 Reproduction and Investigation of the LLL Distribution 16

4.1 Noise at the Input . . . 17

4.2 Random Noise . . . 20

5 Conclusion & Future Extensions 22

5.1 Conclusion . . . 22

5.2 Future Extensions . . . 23

5.2.1 Elaboration of the Random Noise for the LLL Distribution 24
5.2.2 Train The Computer To Perform Quantum Math 24
5.2.3 New Distributions and Networks 24

Bibliography **26**

Chapter 1

Introduction

1.1 Motivation

With the further development of quantum computer and consequently with quantum computation, cryptography and distributions, these fields are getting interesting in different directions. From how to entangle several atoms up to the implementation of a quantum algorithm, the area of current investigation in these fields is great. In addition to current progress, more interesting benefits of quantum physics in computer science has been developed and even more are in the pipeline. Then no one can doubt that the next technological change will include quantum computers and can bring us, not just in science, but also in the daily routine several benefits.
One of these fields is the handling of quantum distributions. As every distribution is a quantum dispersion build-up from different sources in a network and contains all possible conditions these sources can have. However, contrary to classical distribution, the quantum ones cannot be reproduced by classical sources using hidden variables, similar to the Bell scenario. Even if we use a neural network in order to learn a computer to reproduce such a distribution, it does, as proven in [1], not work.
Nevertheless, if we want to reproduce such a given distribution by a computer, then we have to use another ansatz. In this thesis, we used the idea of teaching the computer how to perform a quantum calculation, which is necessary in order to calculate such a dispersion. Due to this ansatz, the computer must only figure out the different conditions, which the sources in the network have used for the calculation of the quantum distribution.
In order to implement this ansatz, we firstly used the fact that these conditions are linked together. After we have figured out these connections between these conditions, we used several loops and recursion to firstly reduce the remaining possible conditions and then figure out a final possible combination and arrangement of these conditions.
Additionally, we observed another distribution, which is on the brink of being local and can thus be reproduced by a classical neural network. In order to show the distributions proximity to the border between locality and nonlocality, we

implement two different kinds of noises. With the first kind of noise we can stir this distribution towards the nonlocal area, where it cannot be approximated well by the neural network. This can be viewed in Figure 4.3 and Figure 4.5, where both the target distribution and the reproduced distribution with and without noise are shown.
With the other kind of noise we have shown that the distribution is quite robust, meaning that it is not easy to set this dispersion in the nonlocal area, even though this distribution is near the border between locality and nonlocality. This have we observed by adding complete random noise to the distribution, which does not set this distribution, contrary to the previous kind of noise, in the nonlocal part.

1.2 Related Work

The idea of reproducing a quantum distribution was already investigated in the paper from Kriváchy [1]. In this paper, they have observed the behaviour of different quantum distributions, which are known or even proven to be non-local. By introducing two different kinds of noises, one at the source and one at the detector, they have set these distributions in the area of locality.
In order to show the locality/nonlocality, they used a neural network, based on so-called hidden variables, which try to reproduce a given distribution. If it can be reproduced by the machine afterwards, then this dispersion is local, otherwise, it is non-local and thus a quantum distribution. After the non-local distribution consists of around 22 per cent of noise, the distribution can be reproduced by the neural network and is therefore local.
In this paper, they used different distributions for the purpose of testing out their method under several circumstances. Due to this noise, which is mainly one single continuous parameter in the range between zero and one, the authors of this paper can observe these distributions not just in the local and non-local area, but also in the transition between these two fields.
As a start, we were able to use the corresponding code ([2]) of this paper. Based on this we started to investigate these distributions, mainly focused on the Fritz ([3]) distribution. In this bachelor thesis we used a different ansatz without a neural network, in order to successfully reproduce also the non-local part of this distribution (without any noise). Additionally, we modified their neural network in order to reproduce the classical Lovász-Local-Lemma distribution ([4]). Furthermore, we implemented different kind of noises, but contrary to the paper from Kriváchy, we used it to set the LLL distribution in the nonlocal area.

Chapter 2

Background

2.1 Quantum Distribution Calculation

In order to understand a given quantum network, we observed the possible outputs of such a network and their corresponding probability. By summarising all combinations of the different possible outputs, we got a distribution, which can be a classical or a quantum ones. Here we observed possible distribution in the so-called 'triangle' network (Figure 2.1), which is the simplest tripartite topology. This triangle scenario has been proven in [3] to be the smallest network, in which a non-classical correlation exists. Even though it is a simple scenario, the numerically and theoretically complexity should not be underestimated.
In this network, we use the binary or quaternary outputs of each of the three parties, leading to 8, respective 64 possible states. Each of these states has its own, specific probability, which then can be concentrated in a probability distribution. The resulting distribution has been later used to be reproduced by the machine. *If you are interested in the here used quantum computation (set up, calculation and background), please read [5].*

2.1.1 Fritz Distribution

One of these distribution is introduced by Fritz [3]. It is based on the Bell scenario ([6]), modified for the triangle scenario and its non-classicality is therefore well understood. Here we want Alice and Bob to share one of the four Bell states, i.e., $|\Psi_{AB}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. Charlie meanwhile share with Alice either a classically correlated state, i.e. $|\Psi_{AC}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, or a maximally entangled state (like the Bell states). It does not depend which one is used, nor which Bell state is used.
The purpose of this connection is straightforward. Alice and Bob first measure the system they share in the computational state $(|0\rangle, |1\rangle)$. Depending on the random binary outcome, Alice measures either in the $|\Phi_0\rangle$ or in the $|\Phi_1\rangle$ basis. Parallel to Alice, Bob measures according to his outcome which Charlie, in the

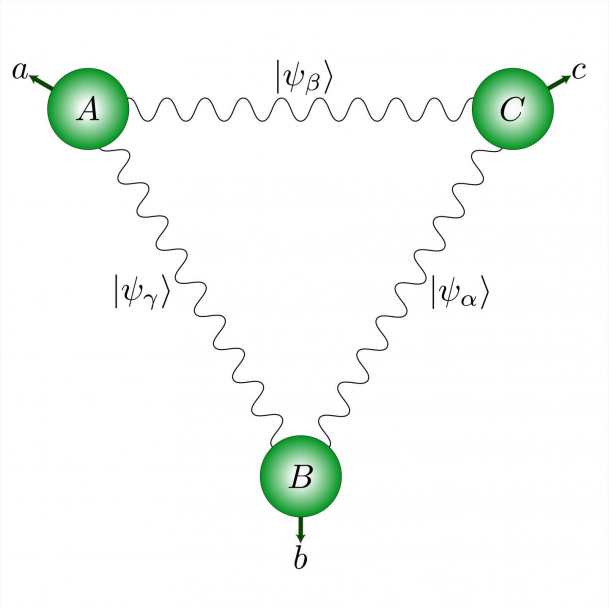

Figure 2.1: Triangle network configuration, where later A, B and C are named as Alice, Bob and Charlie

$|\Theta_0\rangle$ or $|\Theta_1\rangle$ basis. The outcome of both measurements is announced by Alice and Bob as their total output. Charlie measures his state, similarly to Alice and Bob, in the $\{|0\rangle, |1\rangle\}$-basis and announce it as his output. Due to this measurement, Charlie is aware of the measurement setting Alice and Bob used.
Overall, each of the three parties can have four different outputs, leading to 64 various combinations. Nevertheless, not all of the 64 combinations are possible. If Alice or Bob (or both) are currently not measuring in the same base as Charlie, then this combination of states is illegal and have thus a probability of zero. Finally, every single combination with its own probability to reach can be put together in a probability distribution *p(Alice, Bob, Charlie)*.
In conclusion, such a possible distribution can be viewed in Figure 3.1. As seen in the dispersion, just a few possible combinations of states of Alice, Bob and Charlie have a probability higher than zero. For the entire calculation of a possible distribution, please read section 3.1.

2.1.2 LLL Distribution

The LLL distribution was introduced in [4] and is based on the Lovász-Local-Lemma, which was proven in [7]. The idea behind this lemma is to find conditions under which one kind of output events in a probability space can get rid of them simultaneously. This kind of output is called “bad events” and is defined as the output “0”.

This distribution is known to be local, but close to the border of the nonlocal area. Still, it can be reproduced by classical sources, even so, a pretty small amount of specific noise can steer it in the nonlocal field, which will be proven in section 4.1. This distribution is also set in the triangle configuration, but it does not use quaternary outputs of the three parties as in the Fritz distribution. Instead, it uses binary outputs and is thus just a 2-by-2-by-2 distribution, which is set up in the following way:
Each of the parties Alice and Bob have the output value $a(X,Z) = X$, $b(Y,Z) = Y$ with $X, Y \in \{0,1\}$. The variable Z is here meaningless, but because of the fact that in triangle configuration exists distributions where a and b depends of Z, we left it here for consistency. Meanwhile Charlie's output is defined by $c(X,Y) = (1-X)(1-Y)$.
The probability of the possible values of X and Y are similarly. For $X = Y = 0$ is the probability $P(X=0) = P(Y=0) = \frac{-1+\sqrt{5}}{2}$ and for $X = Y = 1$ it is $P(X=1) = P(Y=1) = \frac{3-\sqrt{5}}{2}$. Overall, this leads to the following probability distribution p(Alice, Bob, Charlie):

$$\begin{array}{cccc} p(0,0,0) = 0 & p(0,0,1) = 0.381966 & p(0,1,0) = 0.236068 & p(0,1,1) = 0 \\ p(1,0,0) = 0.236068 & p(1,0,1) = 0 & p(1,1,0) = 0.145898 & p(1,1,1) = 0 \end{array} \tag{2.1}$$

This distribution is normalised, thus the sum over all probabilities leads to one. Due to the eponymous Lovász-Local-Lemma, another equation, called symmetric probabilities, must be hold. According to it must $p(0,0,1) = 0.381966$ be equal to the addition of $p(1,0,0) = 0.236068$ and $p(1,1,0) = 0.145898$, which is here satisfied to.

2.2 Neural Network

Since the first idea of connecting adjustable points together ([8]), neural networks are getting a successful tool for solving a great variety of issues in recent years. The idea behind a neural network is quite intuitive. It consists of an input layer, where it pre-process the incoming data, some hidden layers and an output layer. Each of the layers can consist of several adjustable points, which are connected to his neighbour layers, as it can be viewed in Figure 2.2. These points have a special activation function, which can be modified in order to customise the neural network.
For training the neural network, we gave it some samples consisting of an input and an output. The neural network will now use these samples to adjust their points until it can reproduce the corresponding output from an input. Additionally, we gave the neural network some validation samples, so it can check if it is on the right way. Furthermore, we can test it with some test samples, so we see

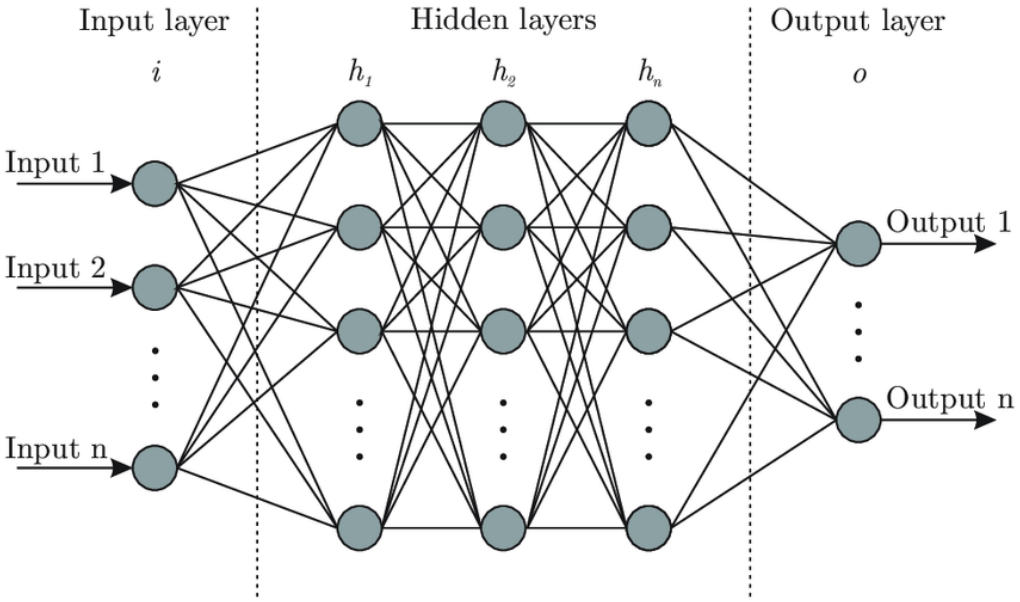

Figure 2.2: A sample neural network with the input layer, some hidden layers and the output layer. The number of layers and even the connection between the layers can be customised so that it can tackle the given problem best. While the output layers are mainly for merging the values from the neural network into the final output value type and the input layers are for preparing the data, the main training part is placed by the hidden layers in the middle section.

how well the neural network perform. Finally, if the neural network has learned enough, we can use it to predict the corresponding output from any input.
As for our needs, we took the neural network from Kriváchy [2] and modified it for our purpose. It is still a classical neural network and not like in [9] or general in [10] a quantum one. The "feedforward" neural network is based on three hidden sources, which are like in a directed acyclic graph connected together. While the input (three hidden variables) consist of uniformly drawn random numbers, the outputs are the conditional probabilities. This output is computed with a Monte Carlo approximation, in order to describe a joint probability distribution, which leads us to the reproduced quantum distribution.
Due to the fact that not just the input is restricted, but also the whole neural network, it will only be able to reconstruct local models. This limiting architecture is built upon only three not fully connected part, where each of them consists of several layers of magnitude equal to four. For the reproduction of the LLL distribution we only used one "Dense" layer per part, which used a "tanh" activation function. Together with a "Lambda" layer, for the pre-processing of the input data, and a final "Dense" layer, which used a "softmax" activation function, we just had 66 trainable parameters in the neural network.
As an optimizer we implemented a large variety of several different ones, like

```
Parameter:  0.0
['000: 0.0 ' '001: 0.381966 ' '010: 0.236068 ' '011: 0.0 ']
['100: 0.236068 ' '101: 0.0 ' '110: 0.145898 ' '111: 0.0 ']

Round 1 of 1, with model triangle and LLL distribution of param v = 0.0. Normalization (sum): 1

10000/10000 [==============================] - 26s 3ms/step - loss: 8.2605e-05 - val_loss: 9.7809e-06
10/10 [==============================] - 0s 1ms/step - loss: 8.6516e-06
Loss (with test samples):  8.651613825350069e-06
Distance calculated:  0.01684553017065297
Distribution: LLL, normalization (sum): 1

Parameter:  0.0
['000: 0.0048685 ' '001: 0.3853791 ' '010: 0.2222161 ' '011: 0.0013936 ']
['100: 0.2423987 ' '101: 0.0013948 ' '110: 0.1423513 ' '111: 0.0 ']
Distance:  0.01684553017065297
```

Figure 2.3: A sample training session of a single iteration. In the upper part of the picture is the target distribution, which the neural network tried to reproduce in a single training iteration with the below-mentioned configuration. At the bottom is the approximated distribution with, according to the target dispersion, the discrepancy. Additionally is the final loss value from the training, validation and test section and the current value of the parameter (please read section 4.1) visible.

"FTRL", "RMSprop", "Adam", "Adadelta" and so on. For the loss function, we have implemented as well a huge selection of options, reaching from the standard "mean squared error" function up to particular loss functions as "Log-Cosh error" or the "Kullback-Leibler divergence". Further the implemented loss function "Euclidean distance" is used at the end of the neural network for the visualisation of the discrepancy between the predicted and the target distribution, rather than using it as a loss function. Such a training procedure with input and the predicted output can be viewed in Figure 2.3
Here we founded the best configuration for the optimizer and loss function to be the combination of "$Log - Cosh\ error''$ and the "$adam''$-optimizer with a learning rate of 0.01. With a batch size of 6,000 and 10,000 training, 1,000 validation and ten test samples we got a discrepancy ("Euclidean distance") of down to 0.005 for the LLL distribution, which satisfied our goal of a maximum distance of 0.01.

Chapter 3

Implementation and Reproduction of the Fritz Distribution

3.1 Quantum Calculation by Python

As mentioned in the introduction, no classical attempts based on hidden variables can reproduce a given quantum distribution. Thus, we have to use a different ansatz in which we trained the computer to calculate quantum computations, which are the foundation for a quantum dispersion. In a first step, we wanted to calculate a Fritz distribution (see subsection 2.1.1) by using defined conditions for Alice, Bob and Charlie.
First of all, we had to implement the whole calculation for quantum computation, namely for a joint measurement of two basis. Here, these basis are built up from the three Pauli matrices. Having done that we can set up the four conditions of Alice, Bob and Charlie, according to the previously mentioned setup (subsection 2.1.1) of the Fritz distribution. Following these rules, we started the calculations of the probabilities according to every possible arrangement of these conditions, which is the foundation of our dispersion. Having done that, we wrote all of the 4^3 possible dispositions in a 4-by-4-by-4 tensor. In that way we receive our nonlocal Fritz distribution, which is based on the by us defined conditions of the three parties.
Concretely these three steps, beginning with the possible conditions, looks like:

3.1.1 Condition of Alice, Bob and Charlie

Alice, Bob and Charlie each have two different parameters, while each of them has two possible states. Concluding they have four possible conditions, which can be written in a 1-by-4 matrix. Alice's and Bob's first parameter specify the basis, which they use for the measurement ($|\Phi\rangle/|\Theta\rangle$), and the second parameter define the outcome $|0\rangle/|1\rangle$. Charlie's parameters meanwhile consist only of the

by Alice and Bob used measurement basis. Thus they are the same ones, but can be in a different arrangement as by Alice's and Bob's first parameter.
Therefore for example, the 1-by-4 matrices of Alice, Bob and Charlie can look like the following:

$$Alice = \begin{pmatrix} X+Y,0 \\ X-Y,1 \\ X-Y,0 \\ X+Y,1 \end{pmatrix} Bob = \begin{pmatrix} X,1 \\ Y,0 \\ X,0 \\ Y,1 \end{pmatrix} Charlie = \begin{pmatrix} X-Y,X \\ X+Y,Y \\ X-Y,Y \\ X+Y,X \end{pmatrix} \tag{3.1}$$

The 'X' and 'Y' represent the Pauli matrices, which are used for the joint measurement

Note that it does not depend on which condition stands at which position in the matrix, only the above mentioned "rules" must be followed.

3.1.2 Calculation of a Measurement

The outcome of the probability $p(x)$ through a (joint) measurement can be calculated with:

$$p(x) = \langle\Psi|M|\Psi\rangle = tr(|\Psi\rangle\langle\Psi| M) \tag{3.2}$$

While here is Ψ one of the four bell states, written as a density vector, like:

$$\Psi = \frac{|00\rangle + |11\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

The measurement operator M consists here of two components. One of them is the measurement operator (M_A) defined by Alice's first parameter (see in the previous section), whereas the other one (M_B) is defined by Bob's first parameter. If for example the Pauli matrix 'X' is used, then we first compute the corresponding eigenvectors $\vec{e}_1 = \begin{pmatrix} -1 \\ 1 \end{pmatrix} / \vec{e}_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$.
The two measurement operators are then calculated by the outer product of each of the eigenvectors. This two matrices must satisfy the norm: $\mathbb{1} = M_0 + M_1$. Depending on the outcome, "0" or "1" (second parameter of Alice and Bob), we take one of the matrices as our measurement operator M_A:

$$M_{A,0} = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} / M_{A,1} = \frac{1}{2} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

Together with the other measurement operator (which can be in another basis), the measurement operator M is then calculated by the outer product (or Kronecker product).

$$M = M_A \otimes M_B \tag{3.3}$$

Overall, the probability of a single state is computed like:

$$p(x) = tr(|\Psi\rangle \langle\Psi| M) = tr(\frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 1 \end{pmatrix} M_{A,\{0,1\}} \otimes M_{B,\{0,1\}})$$

3.1.3 Setting up the Distribution

In the end, a distribution is set up by the probability of each possible state. So here, it would consist of $4^3 = 64$ possible states, where each of them has a certain (normalised) probability to get to this specific state (like the state '210' has the probability a and the state '032' has the probability b).
The probability is calculated by the above-mentioned measurement, where the measurement operator is chosen by the current conditions of Alice, Bob and Charlie. For example below we used the example states mentioned in Equation 3.1, likewise for the example distribution in Figure 3.1

Example 1, state '210':
-Charlie has the condition '0', so his parameters are $'X'$ and $'X - Y'$.
-Alice has the condition '2', so her parameters are $'X'$ and 0.
-Bob has the condition '1', so his parameters are $'X - Y'$ and 1.
This probability is unequal to zero because the parameters Alice and Bob use are the given states from Charlie. So for the measurement, the first measurement operator M_A is $M_{X,0}$ and the second one is $M_B = M_{X-Y,1}$. So their probability is computed now by:

$$p(x) = \langle\Psi|M_A \otimes M_B|\Psi\rangle /4 \tag{3.4}$$

The '/4' comes from the four possible conditions Charlie can have (each with a equal probability).

Example 1, state '032':
-Charlie has the condition '2', so his parameters are $'Y'$ and $'X - Y'$.
-Alice has the condition '0', so her parameters are $'X'$ and 1.
-Bob has the condition '3', so his parameters are $'X + Y'$ and 1.
This probability is equal to zero, as Alice and Bob use parameters which are not in the given conditions from Charlie. Only when Alice has as her first parameter $'Y'$ and Bob has $'X - Y'$ too is this state possible and thus the probability is higher then zero.

```
['000: 0.0 ' '001: 0.0 ' '002: 0.0 ' '003: 0.018305826 ']
['010: 0.0 ' '011: 0.018305826 ' '012: 0.0 ' '013: 0.0 ']
['020: 0.0 ' '021: 0.0 ' '022: 0.0 ' '023: 0.106694174 ']
['030: 0.0 ' '031: 0.106694174 ' '032: 0.0 ' '033: 0.0 ']
['100: 0.106694174 ' '101: 0.0 ' '102: 0.0 ' '103: 0.0 ']
['110: 0.0 ' '111: 0.0 ' '112: 0.018305826 ' '113: 0.0 ']
['120: 0.018305826 ' '121: 0.0 ' '122: 0.0 ' '123: 0.0 ']
['130: 0.0 ' '131: 0.0 ' '132: 0.106694174 ' '133: 0.0 ']
['200: 0.018305826 ' '201: 0.0 ' '202: 0.0 ' '203: 0.0 ']
['210: 0.0 ' '211: 0.0 ' '212: 0.106694174 ' '213: 0.0 ']
['220: 0.106694174 ' '221: 0.0 ' '222: 0.0 ' '223: 0.0 ']
['230: 0.0 ' '231: 0.0 ' '232: 0.018305826 ' '233: 0.0 ']
['300: 0.0 ' '301: 0.0 ' '302: 0.0 ' '303: 0.106694174 ']
['310: 0.0 ' '311: 0.106694174 ' '312: 0.0 ' '313: 0.0 ']
['320: 0.0 ' '321: 0.0 ' '322: 0.0 ' '323: 0.018305826 ']
['330: 0.0 ' '331: 0.018305826 ' '332: 0.0 ' '333: 0.0 ']
```

Figure 3.1: Possible distribution, where the state '210' refers to states: Alice = '2', Bob = '1', Charlie = '0'

3.2 Reproduction

After we calculated from the given conditions of Alice, Bob and Charlie a quantum distribution, we wanted now to let the machine do the opposite. Doing so and in order to reproduce the given distribution, we gave a quantum dispersion to the machine and let it find out the possible conditions. This turned out to be more challenging than it seemed at first. Even though the 'rules' (from section 3.1) of how these conditions must be adjusted in order to get the target distribution are clear and simple to implement, they are closely linked to each other. Meaning that, whenever we changed a condition while trying out to reproduce a single given probability of a specific state, it changed more or less the whole distribution. This happens due to the fact that a single condition from for example Charlie influence the distribution in 16 states.
Further problem was that changing different conditions could lead to the same result. Therefore, if you want to swap for example the output of the measurement operator of Alice's first and fourth condition in order to change a specific probability, then you probably can do the same with two of Bob's conditions, leading to the same result. The same effect can be seen in Figure 3.5, where different conditions lead to the same distribution. This leads to the problem that you never know if the just swapped conditions are a progress in the right direction or not.
So a simply recursion or loop can not be successful. If we would have done that, we would have to record every single change the program had done, in order to know which changes are not yet tried out. In addition, testing out every possible combination in a simple loop would take too long, because of the huge amount (up to $6 * 10^6$) of possible combinations. This number can be increased significantly

```
[0.         0.         0.         0.01830583 0.         0.01830583
 0.         0.         0.         0.         0.         0.10669417
 0.         0.10669417 0.         0.         0.10669417 0.
 0.         0.         0.         0.         0.01830583 0.
 0.01830583 0.         0.         0.         0.         0.
 0.10669417 0.         0.01830583 0.         0.         0.
 0.         0.         0.10669417 0.         0.10669417 0.
 0.         0.         0.         0.         0.01830583 0.
 0.         0.         0.         0.10669417 0.         0.10669417
 0.         0.         0.         0.         0.         0.01830583
 0.         0.01830583 0.         0.        ]
```

Figure 3.2: The high percentage of states with a probability equal to zero can be seen in this distribution directly.

if were using additionally measurement operators among to the yet used Pauli matrices and their addition/subtraction. Already a doubling of the measurement operators would lead to an 18-times greater number of possible combinations, impossible to try out every assembly of conditions.
For the purpose of resolving this problem, we started looking at how the conditions of the three parties are connected to which one. Meaning that we wanted to know which combination of conditions has a probability equal to zero and which one has not. Due to the fact that just a quarter of all probabilities in the distribution are not equal to zero, we got a close overview of the linked conditions of Alice, Bob and Charlie. Only when each measurement operator of Alice and Bob is on the same basis as Charlie's measurement operators, it leads to a probability unequal to zero. This is only the case in 16 out of 64 different conditions, as it can be seen in Figure 3.2.
Thanks to that, we can reduce the possible conditions for the target distribution significantly (down to around $3 * 10^5$). As next, we just have to test every measurement basis, which Alice, Bob or Charlie can have used for the calculations. There are still hundred thousand of possibilities on which basis each of them have been used and how they can be arranged. However, several different arrangements lead to the same distribution and some basis lead to values that are completely different from the distribution. Knowing that we can sort out the majority of different basis and look just at a 'few' disposals.
Finally, out of around 50 different arrangements possibilities, we only have to figure out the correct one. This is just a small number which we have to try out. In order to increase the speed and efficiency of the program, we used a short recursion and some loops, where we continually observed every point of this distribution. If one point has another probability as in the target distribution, we started looking at the conditions of Alice, Bob and Charlie and swapped them, until we found better arrangements for these conditions. In the end, this leads us to a possible legal order of conditions for Alice, Bob and Charlie in order to

```
alice:
 [['Z' '+' 'Y' '0']
 ['Z' '-' 'Y' '1']
 ['Z' '-' 'Y' '0']
 ['Z' '+' 'Y' '1']]
bob:
 [['Z' 'N' 'N' '1']
 ['Y' 'N' 'N' '0']
 ['Z' 'N' 'N' '0']
 ['Y' 'N' 'N' '1']]
charlie:
 [['Z' '-' 'Y' 'Z' 'N' 'N']
 ['Z' '+' 'Y' 'Y' 'N' 'N']
 ['Z' '-' 'Y' 'Y' 'N' 'N']
 ['Z' '+' 'Y' 'Z' 'N' 'N']]
```

Figure 3.3: Target conditions for Alice, Bob and Charlie.

```
Founded possible states. alice:
 [['Y' 'N' 'N' '0']
 ['X' 'N' 'N' '0']
 ['X' 'N' 'N' '1']
 ['Y' 'N' 'N' '1']]
bob:
 [['X' '+' 'Y' '0']
 ['X' '-' 'Y' '1']
 ['X' '+' 'Y' '1']
 ['X' '-' 'Y' '0']]
charlie:
 [['X' 'N' 'N' 'X' '+' 'Y']
 ['Y' 'N' 'N' 'X' '-' 'Y']
 ['X' 'N' 'N' 'X' '-' 'Y']
 ['Y' 'N' 'N' 'X' '+' 'Y']]
```

Figure 3.4: By the program founded conditions for Alice, Bob and Charlie.

```
Distribution: quantum calculated distribution, normalization (sum): 1
['000: 0.0 ' '001: 0.0 ' '002: 0.0 ' '003: 0.018305826 ']
['010: 0.0 ' '011: 0.018305826 ' '012: 0.0 ' '013: 0.0 ']
['020: 0.0 ' '021: 0.0 ' '022: 0.0 ' '023: 0.106694174 ']
['030: 0.0 ' '031: 0.106694174 ' '032: 0.0 ' '033: 0.0 ']
['100: 0.106694174 ' '101: 0.0 ' '102: 0.0 ' '103: 0.0 ']
['110: 0.0 ' '111: 0.0 ' '112: 0.018305826 ' '113: 0.0 ']
['120: 0.018305826 ' '121: 0.0 ' '122: 0.0 ' '123: 0.0 ']
['130: 0.0 ' '131: 0.0 ' '132: 0.106694174 ' '133: 0.0 ']
['200: 0.018305826 ' '201: 0.0 ' '202: 0.0 ' '203: 0.0 ']
['210: 0.0 ' '211: 0.0 ' '212: 0.106694174 ' '213: 0.0 ']
['220: 0.106694174 ' '221: 0.0 ' '222: 0.0 ' '223: 0.0 ']
['230: 0.0 ' '231: 0.0 ' '232: 0.018305826 ' '233: 0.0 ']
['300: 0.0 ' '301: 0.0 ' '302: 0.0 ' '303: 0.106694174 ']
['310: 0.0 ' '311: 0.106694174 ' '312: 0.0 ' '313: 0.0 ']
['320: 0.0 ' '321: 0.0 ' '322: 0.0 ' '323: 0.018305826 ']
['330: 0.0 ' '331: 0.018305826 ' '332: 0.0 ' '333: 0.0 ']
```

Figure 3.5: While either the defined conditions (Figure 3.3) and the founded conditions (Figure 3.4) are different, they both lead to the exactly same distribution. This distribution is shown here, where the different states (for example '132') are written together with their probability ($p(Alice = 1, Bob = 3, Charlie = 2) = 0.106694174$).

reproduce the given distribution.
In conclusion, we can reproduce the given quantum distribution. Due to the fact that, as above mentioned, several different measurement operators leads to the same result, we can use different conditions for computing the same distribution. This can be seen in Figure 3.5, where the parties and their different conditions for calculating the distribution are not the same ones as the ones founded by the program.

```
Input string: [['Y' '+' 'X' '0' 'Y' 'N' 'N' '1']]
Input preprocessed: [[2 7 4 6 2 3 3 5]]
Output probability: [0.0625]
```

Figure 3.6: The input string consists of two randomly chosen measurement operators. The first three signs specify the first measurement operator, meaning that here ['Y' '+' 'X'] refers to the measurement operator based on the addition of the Pauli matrix "Y" and "X". Similar refers the fifth, sixth and seventh (['Y' 'N' 'N']) characters, while 'N' stands for an unused place, to the second measurement operator, which is here based on the "Y" matrix. Lastly, the remaining two signs specify the output of the two measurement operator. In order to give the neural network a standardized input string, we have to preprocess it, which leads to the next string in the image. Finally, the output consists of a single float, representing the probability from the joint measurement of the two basis.

Based on this fact and not having any additional information's about the used conditions of Alice, Bob and Charlie, it is impossible to discover the exactly used conditions and their arrangements for this distribution. However, this is somewhat meaningless, then our goal was to reproduce a given distribution, which we successfully managed.

3.3 Training Python to do Quantum Math by Himself

In the previous section, we have implemented with several lines of codes the calculation for quantum computations, in order to calculate a quantum distribution. From now on we want that the machine can do it by itself, without any kind of nearly hard-coded implementation. In order to achieve this goal, we set up a neural network with the purpose of learning the quantum calculation of a joint measurement.
As an input, we just used the basis of the measurement operator, which is implemented here as a short string. The neural network accepts input data just in some predefined types, so we have to compile the input string into a tokenized string, where each character is replaced with a specific number, which is shown in the middle string in Figure 3.6. The target output itself is then a single float, which is, according to the input measurement operators, the corresponding probability of the joint measurement calculation. An example of an input and the corresponding output can be seen in Figure 3.6.
For the auxiliary functions we implemented a huge variety. As an optimizer there is a great selection consisting amongst other the "Adam", "Nadam", "FTRL", "SGD" or the "RMSprop". The "FTRL" is a optimization algorithm, which suits best for models with large and sparse feature spaces, whereas the focus of the "Adam" algorithm lies more on deep neural networks.

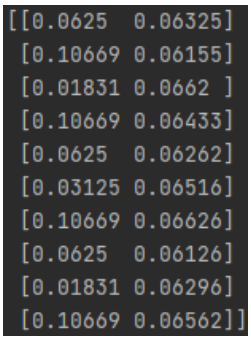

Figure 3.7: On the left side of the tabular are the correct output probabilities, while on the right side are the predicted values written. As we can see, the neural networks prediction stands in no relation to the correct values.

Concerning the loss function, we have started with simple "mean squared error", "mean absolute error" or the "mean absolute percentage error" functions, continuing up to particular functions as the "Log-Cosh error" or the "Kullback-Leibler divergence". While the "mean squared error" evaluate higher distances more than smaller ones, the "Log-Cosh error" has the advantage of not being strongly affected by occasional wildly incorrect prediction.
Only for the metrics we have used either a standard one, namely the "accuracy class" or no one. As a model, we have tried out several possible combinations of different layers, like "Dense", "Dropout", "BatchNormalization" or "Embedding" layer. A varying width of the neural network (number of (trainable) points per a single layer) have we tested too, leading to eight up to several thousands trainable parameters.
When it comes down to the training procedure of the neural network, the problem appeared. Even though the neural network seems to learn from the given samples, he just does not learn right enough to give useful predictions. Even with enough time (until the loss is not getting smaller anymore), it can not give accurate predictions, as it can be seen in Figure 3.7. It seems that the neural network does not use the input data as it should, because with random input and output we got the same useless predictions. Due to lack of time, we could not have tackled this issue, but it can be considered for a future improvement.

Chapter 4

Reproduction and Investigation of the LLL Distribution

The LLL distribution (subsection 2.1.2) is known to be in the local area and thus calculable with hidden variables. Accordingly, we used the given neural network for the reproduction of this dispersion. We modified this neural network in terms of size and add several auxiliary functions, like additional optimizer and loss functions, which can be read in section 2.2. These changes were necessary to customise the neural network for our needs and to achieve acceptable results. In order to evaluate how good the neural network's prediction of the dispersion is, we look at the distance between the predicted distribution and the target ones, which we manually computed according to subsection 2.1.2. We have chosen as a first step a target maximum discrepancy of 0.01, which is calculated by the "Euclidean distance" $d(predicted, target) = \sqrt{\sum_{a,b,c}^{8}(p(a,b,c) - t(a,b,c))}$. This can already be achieved with a pretty restricted network consisting of 66 trainable parameters, which are split up into three different layers, each representing a hidden variable.
As common with neural networks, after a certain point they like to tend using huge amounts of new samples in order to reach just a little better approximation of the target distribution. Here we found this point at a pretty sparse neural network. With just 66 trainable parameters, 10,000 samples, 1,000 validation samples and around ten minutes on a high-end PC we already achieved pretty accurate reproductions of this distribution. With a calculated discrepancy of down to 0.004, we cannot just say that this reproduction is, without doubt, local, but also we observed that this approximation is quite accurate and satisfied our goal of a maximum distance of 0.01.
In order to achieve such a small distance, we let the neural network training several times the same setup and used the shortest distance, which was founded by these attempts, as the final discrepancy between the target and predicted dispersion. We did this since the output distribution of the neural network is not every time the same, nor the founded distances according to it. This difference between several attempts can be seen in Figure 4.1.

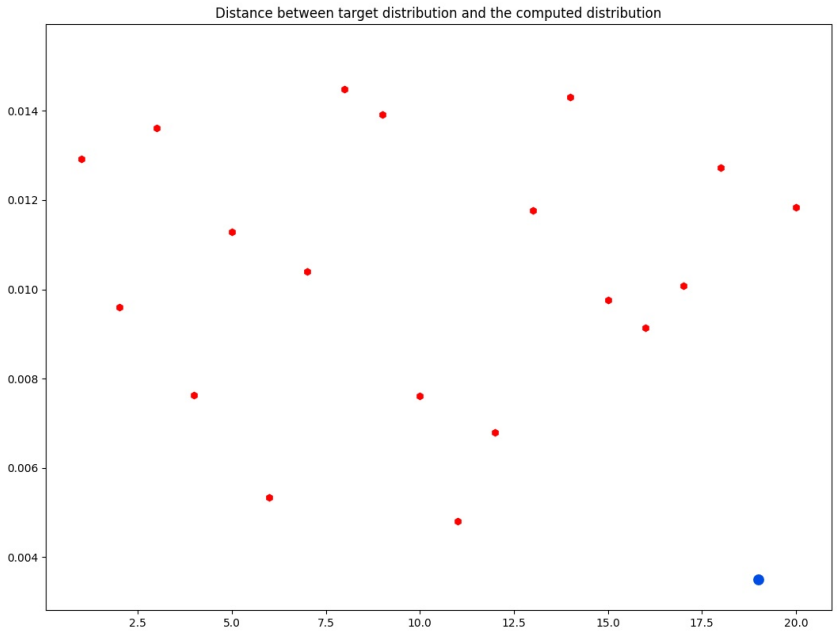

Figure 4.1: The discrepancy between the predicted distribution of the neural network and the target dispersion can reach, due to the behaviour of a neural network, different values with the same starting point. That's why we let the neural network training the same set up several, here 20 times, in order to find and use the shortest distance, which is here represented by the blue point.

After we have tested this dispersion, we started to introduce noise. We implemented two kinds of noises for the purpose of testing the robustness and locality/nonlocality of the distribution. In order to investigate the robustness of this distribution, we added some small random noise, leading to a distribution, which has slightly different values according to the LLL-distribution.

4.1 Noise at the Input

The LLL distribution is known to be on the brink of locality and nonlocality. In order to show this, we implemented the first noise at the initial part of this distribution's set-up. This means concretely that we added this noise just at the four points $(p(a,b,c) = p(0,0,1) = p(0,1,0) = p(1,0,0) = p(1,1,0))$, where each of them already has a probability higher than zero and let the other four points with their probabilities equal to zero. This kind of noise is set up by

```
Parameter:  0.0
['000: 0.0 ' '001: 0.381966 ' '010: 0.236068 ' '011: 0.0 ']
['100: 0.236068 ' '101: 0.0 ' '110: 0.145898 ' '111: 0.0 ']
```

Figure 4.2: LLL distribution without any noise, used as a target distribution for the neural network.

```
Parameter:  0.0
['000: 0.0012424 ' '001: 0.3800286 ' '010: 0.2355342 ' '011: 0.0006087 ']
['100: 0.236204 ' '101: 0.0003892 ' '110: 0.145993 ' '111: 0.0 ']
```

Figure 4.3: Predicted distribution by the neural network according to the target distribution in Figure 4.2. The approximation of the neural network is pretty good, the discrepancy between the target and the predicted distribution is with 0.00247621 quiet small and thus can this distribution be considered as local.

```
Parameter:  0.2
['000: 0.0 ' '001: 0.581966 ' '010: 0.036068 ' '011: 0.0 ']
['100: 0.036068 ' '101: 0.0 ' '110: 0.545898 ' '111: 0.0 ']
```

Figure 4.4: Target distribution, which is build up from the LLL distribution and consisting around 84.7 per cent of noise.

```
Parameter:  0.2
['000: 0.0004232 ' '001: 0.308815 ' '010: 0.1430866 ' '011: 0.0018289 ']
['100: 0.1515231 ' '101: 0.0007352 ' '110: 0.2646353 ' '111: 0.1289527 ']
```

Figure 4.5: Predicted distribution by the neural network according to the target distribution in Figure 4.4. The approximation of the neural network is really bad, the discrepancy between the target and the predicted distribution is with 0.44174192 way to large and thus is this distribution from above nonlocal.

adding a parameter n at the position $p(0,0,1)$, leading to a different probability for this point, according to the value of the parameter. This parameter is a single float, which change the probability of one single point. Furthermore, for the reason of normalization and following the "Lovász-Local-Lemma" ($p(0,0,1) = p(0,1,0) + p(1,1,0)$), we had to adjust the remaining three chances. Concretely these probabilities according to the parameter n looks now like:

$$\begin{aligned} p(0,0,1) &= (\frac{-1+\sqrt{5}}{2})^2 + n \\ p(0,1,0) &= (\frac{-1+\sqrt{5}}{2}) * (\frac{3-\sqrt{5}}{2}) - n \\ p(1,0,0) &= (\frac{-1+\sqrt{5}}{2}) * (\frac{3-\sqrt{5}}{2}) - n \\ p(1,1,0) &= (\frac{3-\sqrt{5}}{2})^2 + 2n \end{aligned} \tag{4.1}$$

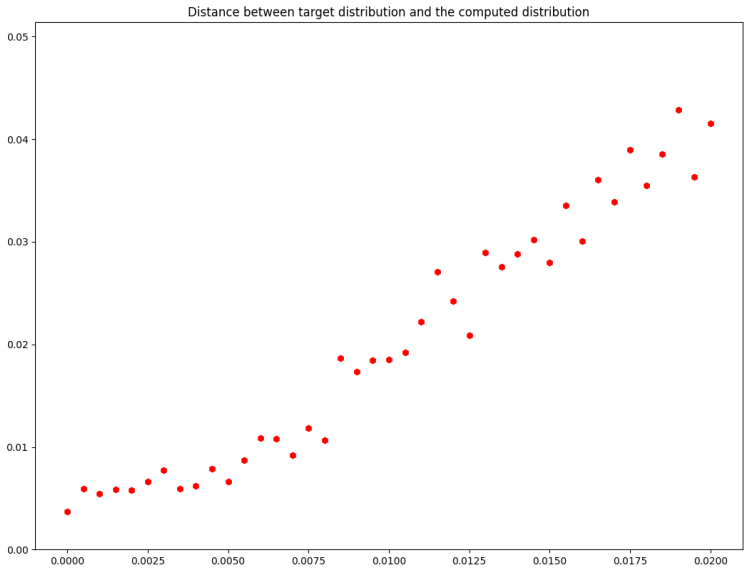

Figure 4.6: The discrepancy between the predicted distribution of the neural network and the target dispersion raise after a short period of consistency according to the increase of the parameter. This raise continue until the parameter reaches its limit at 0.236068.

Thanks to this noise, we can stir the distribution towards the non-local area. Due to the fact that a probability cannot be below zero, we have to limit the parameter in an interval of $n \in [0, 0.236068]$, otherwise, we would have a negative chance in the two points $p(0, 1, 0)$ and $p(1, 0, 0)$. According to the fact that this neural network cannot reproduce any nonlocal dispersion (as it can be seen in Figure 4.3 and Figure 4.5), we can figure out immediately when the distribution reaches the nonlocal field. Moreover, this happens already after a small amount ($n = 0.005$ or 2.12%) of noise is introduced as seen in the Figure 4.6. Due to this nearly immediately increase in the distance between the predicted and the target distribution we can prove that the LLL distribution is still local, but on the brink of being nonlocal.
Additionally, we found an interesting behaviour of the neural network. A small neural network with one hidden layer (with four trainable points on it) per party, where each party represent one of the three hidden variables, leads to the Figure 4.7 above with a continuous rise of the distance. Whereas a bigger network with three hidden layers (with 16 trainable points on it) per party leads to a different Figure 4.8.

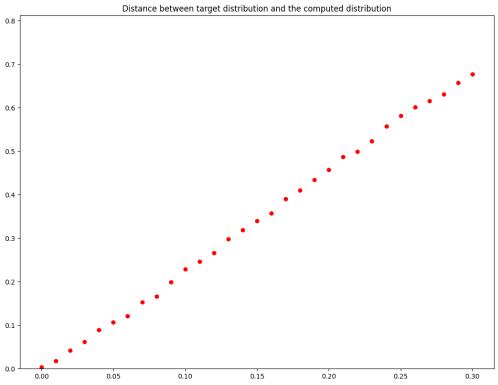

Figure 4.7: The discrepancy between the target and the predicted distributions is increasing steadily according to the increase of the parameter.

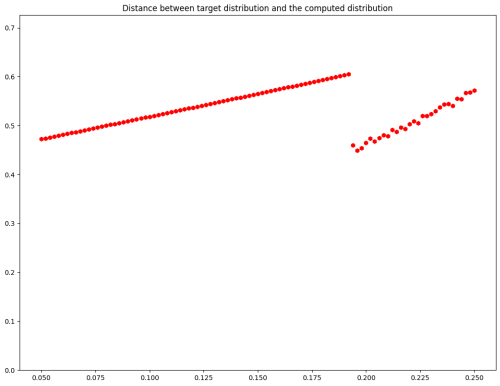

Figure 4.8: The approximation of the neural network is already at the beginning bad and does not change much while the parameter increases.

In this graphic, we can see, that the neural network cannot approximate the same distribution as good as with the smaller network. Due to lack of time, we neither could figure out where the problem lies, nor how to solve it. We did not know if there is a problem with the structure or handling of the bigger neural network, a simple overfit of the neural network or something else. Nevertheless, it can be considered and investigated in future extensions.

4.2 Random Noise

The second type of noise we applied after the calculation of the LLL dispersion. We generated a full random 2-by-2-by-2 distribution and normalised it, hence the sum of the probabilities over all eight points is equal to one. We added this random dispersion together with the original LLL distribution, so we received a new distribution which the neural network can try to reproduce. As a parameter we defined how much of the final distribution consists of the LLL distribution and how much of the randomly generated one. Concretely it looks like:

$$Distribution = (1 - parameter) * LLL + parameter * random \tag{4.2}$$

Even though the random part of the distribution leads to an unforeseen direction of the dispersion, the neural network is able to reproduce this new distribution consisting of random noise (the discrepancy is below 0.01), as it can be observed in 4.9. Thus, this distribution is clearly predictable with this kind of neural network, even with random noise. This shows us that this distribution is, regardless of its proximity to the border, still robust and hard to get in the nonlocal area, meaning that just one special kind of noise leads the distribution towards nonlocality.

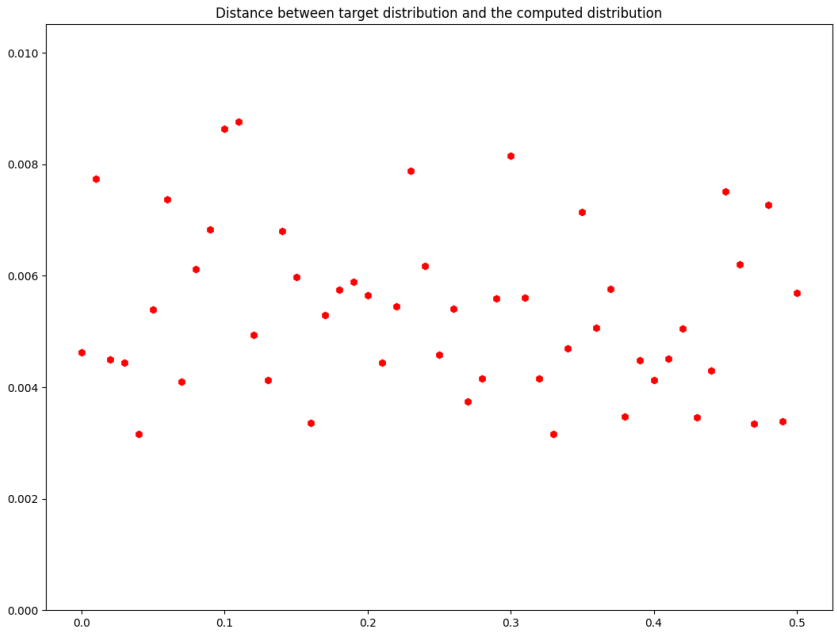

Figure 4.9: The discrepancy between the target distribution and the predicted dispersion by the neural network according to the parameter $p \in [0, 0.5]$. The distance here is with below 0.01 pretty small and can be interpreted as that the neural network can reproduce the distribution, thus even with random noise is the dispersion local.

Chapter 5

Conclusion & Future Extensions

5.1 Conclusion

The behaviour and reproduction of quantum distribution have been investigated in this thesis in different aspects. We were successful with the ansatz of teaching the program how to do quantum calculation. Even though a missing puzzle (neural network for the quantum calculation, see subsection 5.2.2) could not be finished in time, the ansatz is proven to work. Thanks to this ansatz, we were able to figure out the by Alice, Bob and Charlie used conditions, which are the basis for such a quantum dispersion. Following that, we could not only successfully reproduce any given Fritz distribution, but we also have observed that the usage of different measurement operators leads to the same quantum dispersion (Figure 3.5). Even though this ansatz is fitted for the Fritz dispersion, theoretically it can be modified for other quantum distribution as well.
While investigating the LLL distribution, we have observed that a small neural network, based on three hidden variables, can reproduce this probability dispersion. This has proven to us that this distribution is local. The used amount of resources satisfied us too, then with 66 trainable parameters, partitioned in three layers, this neural network is highly restricted.
Furthermore, we have proven by adding noise that this distribution is on the brink of being non-local. We have observed that already a small amount (2.12 %) of noise, implemented with the parameter n, leads to a distribution that cannot be approximated well by the neural network. Even though the distances at this amount ($n = 0.004$) of noise seems still local, the distribution is getting tougher and tougher to be reproduced by the neural network as the noise increase steadily (see Figure 5.1). This shows us, that after this point the distribution is in the nonlocal area and cannot be reproduced with a neural network based on hidden variables.
Additionally, we have shown that this dispersion is still quite robust. While we added some random noise, which leads to a completely unknown direction for the distribution, the neural network has no problem in order to reproduce it (see Figure 4.9). This observation shows us that even though the distribution is near the border of nonlocality, it is not that simple to cross it and just specific noise

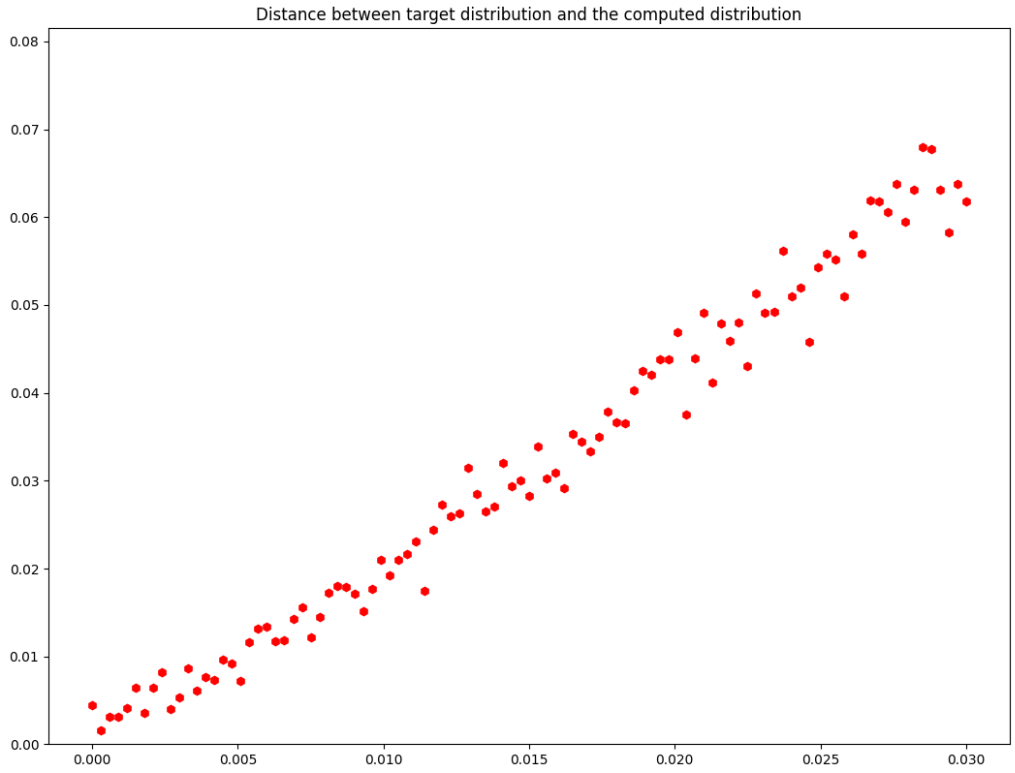

Figure 5.1: The discrepancy between the target and predicted distribution is increasing steadily, which point out that the neural network can just reproduce the original LLL dispersion, but practically nothing of the included noise.

leads to the nonlocal area, where such a classical neural network can approximate it well.
We have uploaded the code on Github, see [11]. Please read the as well uploaded "Read me", in order to work with the program in its full richness.

5.2 Future Extensions

For this paper we had only three month time, consequently, we could not have investigated everything which we wanted. One of these is the founded anomaly in Figure 4.8, where the bigger neural network could not reproduce the distribution as good as a smaller neural network had done. With some more time and resources (like processor power), this question can be revealed to.
Furthermore, this thesis investigated just the surface of this new and fascinating topic. Having a greater amount of time, we could have improved this paper even more and added several nice extensions. Some of them would be:

5.2.1 Elaboration of the Random Noise for the LLL Distribution

While in section 4.2 we have investigated the robustness of the LLL distribution with random noise, which already gave a good overview of how robust this dispersion is. In order to precisely test and determine the local and nonlocal area around the LLL distribution, we need a better and more structured attempt. Mainly we should investigate how good the here used neural network can reproduce a distribution, which is just slightly different according to the original distribution.
Meaning that we have to make a little change of the value of one or more or even all points from the original dispersion and observe if the neural network can reproduce it or not. By doing this and summarising all these changes, we can give a precise definition of which noise leads the distribution to the nonlocal area and which let her in the local area.

5.2.2 Train The Computer To Perform Quantum Math

The idea of training the program to calculate a joint measurement ($p(x) = \langle\Psi|M|\Psi\rangle$) we already followed in 3.3. For this purpose, we wanted to use a neural network. As input, we used a string consisting of the two measurement operators (with their corresponding output) leading to a probability, which we needed as an output of the network. Therefore, we can learn the neural network how to calculate the joint measurement, which we needed not just for the Fritz distribution, but also for other quantum distributions.
The neural network was already set up, as well as the corresponding optimizer, loss function, the calculation for the in-/output and any other auxiliary functions. Due to lack of time and some issues with the learning procedure of the neural network, we could not finish the neural network, however, it can be considered as a starting point for any further improvements.

5.2.3 New Distributions and Networks

In this thesis, we have just looked at one network, the triangle configuration. It was chosen to the fact that it is one of the simplest non-trivial networks. Another simple setup would be the entanglement-swapping between two sources and three parties, as proposed in [12]. Nevertheless, any more complex topology can lead to a worthwhile discovery. Other issues may raise with a more sophisticated reticulation.
One of them may be the used amount of resources, which already cannot be underestimated in a triangle configuration. To tackle this kind of problem new types of algorithm and code structure must be figured out, additionally to an of course improved processing power. Already the re-computation of the LLL distribution, which is definitely one of the smallest dispersions and thus a pretty

easy one to reproduce, needs already at least 10 minutes on a high-end PC. This does not sound like much processing power and time is needed. However, when a 2^3 small distribution needs this amount of resources, how much resources then need a bigger network with for example 4^8 (quantum network with eight parties) possible arrangements?
Similar to the distribution, the amount of possible dispersion is in any kind of network boundless. Even for the triangle configuration, we already know at least three further distributions: the elegant distribution ([13]), the NSI distribution ([14]) and the Renou et al. distribution ([15]). Moreover, with the increase of the model size, the quantity of distributions increases even more. Not only we can find new distribution, but also we can use new kinds of measurement operators, which are the basis of every quantum dispersion. Even though not every kind of distribution is in or around the area of non-locality, there are enough of them which can be observed. Some of them can be reproduced in the here investigated ansatz, for others it is necessary to find a new attempt. Furthermore, every distribution by adding noise can be set in the local or nonlocal area, leading to another worthwhile observing field.

Bibliography

[1] C. D. e. a. Kriváchy T., Cai Y., in *A neural network oracle for quantum nonlocality problems in networks.*, Aug. 2020.

[2] tkrivachy (Tamás Kriváchy), https://github.com/tkrivachy/neural-network-for-nonlocality-in-networks, 2019, source code for "A neural network oracle for quantum nonlocality problems in networks".

[3] T. Fritz, "The triangle scenario c_3," in *Beyond Bell's theorem: correlation scenarios*, Oct. 2012.

[4] X. L. Y. W. M. X. Kun He, Liang Li, in *Variable Version Lovasz Local Lemma: Beyond Shearer's Bound*, Sep. 2017.

[5] I. L. C. Michael A. Nielsen, "Introduction to quantum mechanics," in *Quantum Computation and Quantum Information (10th anniversary edition)*, Sep. 2019.

[6] J. S. Bell, in *On the Einstein Podolsky Rosen paradox*, Nov. 1964.

[7] P. E. László Lovász, in *Problems and results on 3-chromatic Hypergraphs and some related questions*, Jan. 1974.

[8] W. P. Warren McCulloch, "A logical calculus of the ideas immanent in nervous activity." in *Bulletin of Mathematical Biophysics*, Dec. 1943.

[9] D. V. Alexandr A. Ezhov, in *Quantum neural networks*, 2000.

[10] T. F. T. J. O. R. S. D. S. R. W. Kerstin Beer, Dmytro Bondarenko, in *Training deep quantum neural networks*, Feb. 2020.

[11] F. adm (Fabian Auf der Maur), 2021, source code for "Reproduction and Behaviour of Local and Non-local Distribution".

[12] N. G. S. P. Cyril Branciard, Denis Rosset, in *Bilocal versus nonbilocal correlations in entanglement-swapping experiments*, Mar. 2012.

[13] N. Gisin, "The elegant joint measurement on two qubits," in *Entanglement 25 Years after Quantum Teleportation: Testing Joint Measurements in Quantum Networks*, Mar. 2019.

[14] Y. C. P. R. A. T. E. Z. C. S. P. N. B. Nicolas Gisin, Jean-Daniel Bancal, in *Constraints on nonlocality in networks from no-signaling and independence*, May 2020.

[15] S. B. N. B. N. G. Marc-Olivier Renou, Elisa Bäumer and S. Beigi, in *Genuine Quantum Nonlocality in the Triangle Network*, Sep. 2019.