UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Posgrado en Ciencias Físicas

Entanglement Detection Using Neural Networks

T E S I S

que para optar por el grado de

Maestro en Ciencias

PRESENTA:
Diego Alberto Olvera Millán

Tutor Principal:
Dr. Pablo Barberis Blostein, Instituto de Investigaciones en Matemáticas Aplicadas y Sistemas
Comité tutor:
Dr. Alejandro Pérez Riascos, Instituto de Física
Dr. Carlos Francisco Pineda Zorrilla, Instituto de Física

México,CDMX. (Septiembre) 2021

«Experience is simply the name we give our mistakes.»

Oscar Wilde

Abstract

Entanglement is an important resource for quantum technologies, but its detection and classification cannot be performed efficiently across different kinds of quantum states. In particular, quantum mixed states require computationally demanding methods, such as those of convex roof constructions. In this work, I train an artificial neural network (ANN) to perform the classification between entangled and separable two qubit states(mixed and pure), using expectation values of products of Pauli matrices as the entries of the feature vector, $\vec{x}$. The training is performed using only random pure states sampled from the invariant Haar measure. It is found that, using the 15 linearly independent products of Pauli matrices, an accuracy of 98% is achieved for states drawn from the same distribution, and the accuracy for states drawn from the Bures distribution can reach up to 80% (after applying regularization to the model). Using 4 non-orthogonal products of the Pauli matrices, an accuracy of 91% is achieved for states sampled from the Haar distribution, and, when dealing with states sampled from the Bures distribution with purity and concurrence higher than 0.7, an accuracy of up to 84% is achieved.

Acknowledgments

I would like to thank my mom, my dad, my sister, my wife, my friends, my collegues, my teachers and my tutors, thanks to everyone. But seriously, first and foremost I would like to thank my parents, Cristina and Alberto, for all of their support. My wife, Rebeca, for being a source of inspiration and calm. My tutor, Pablo, for his patience and all of the discussions that have enriched this work and my knowledge in physics in general. I would also like to thank my tutoring committee, formed by Alejandro Pérez and Carlos Pineda, for their input on my work. Many of their suggestions made this a better work. Also, I would like to thank my jury for patiently reading this thesis and giving me helpful comments. These helped me to better convey the important facts. Finally, I would like to thank PAPIIT project IG101421 for the financial support. Thank you all.

Contents

Motivation

In the field of quantum information, entanglement has been identified as an important resource in quantum algorithms and other technologies. Small quantum processors are now available and continuously evolving. Characterizing such processors efficiently so one can say for certain that it works quantum mechanically has become an important task. The question of whether the statistics of a given quantum state produced by the processor arise from quantum entanglement or whether it can be explained classically becomes central in this characterization. Performing full state tomography on these quantum states is impractical, one has to perform many measurements in order to get an estimation of the expectation values of a chosen basis of operators. Thus, the search for efficient algorithms that use only partial information about a quantum state in order to classify whether it's entangled or not is important and has experimental and practical relevance.

Introduction

Entanglement is a hallmark feature of quantum mechanics. Correlations are measured between the local observables of multipartite quantum systems that cannot be simulated using a local and real classical theory [1]. This feature has found many applications in quantum technologies [2–4], but its efficient detection and classification (entangled/separable) remain an open problem in the field. The problem of correctly classifying a given state as entangled or separable is known as the separability problem [5].

Research in entanglement detection has produced many useful results. The celebrated Positive Partial Transpose (PPT) criterion [6, 7], for instance, allows us to correctly determine if there is entanglement for all states of low dimensional Hilbert spaces, specifically those of two qubits, 2×2, and of a qubit and a qutrit, 2×3, but it is only a necessary condition for separability in higher dimensions, there are PPT entangled states. Also, to use such a criterion, the full density matrix of a quantum state must be known [8]. This introduces an additional difficulty in entanglement detection. There are a number of proposals to try to avoid this difficulty [9–12], but only work when certain special conditions are met.

In general, it is impossible to completely isolate a qubit state from its surroundings, thus, instead of having qubit pure states, mixed states are found in nature. It has been shown that the correct classification of these states is an NP-hard problem [13]. There exist Hermitian operators called entanglement witness (EW) [14], which can detect entanglement, but finding an EW that detects the entanglement of an arbitrary state is equivalent to the separability problem [15, 16].

Machine learning has been applied to physics problems in recent years, in particular in quantum information. Steerability detection [17], quantum topology identification [18], fidelity estimation [19], among other applications [20] have been studied. For the problem of entanglement detection, neural networks have been used to optimize parameters of a CHSH type inequality [21], and bootstrap aggregating was used to aid in classification with a convex hull approximation [22]. Quantum neural networks have been shown to detect entanglement [23].

In this thesis, I propose the use of an Artificial Neural Network (ANN) to perform entanglement detection of two qubit states. Operators of the form

$$O_{ij} := \sigma_i \otimes \sigma_j, \tag{1a}$$

$$N_{ij} = \sigma_i \otimes \left(\frac{\sigma_j \pm \sigma_i}{\sqrt{2}} \right), \tag{1b}$$

where $\sigma_0 = \mathbb{1}$, and the σ_i's $(i, j = 1, 2, 3)$ are the Pauli matrices, are used to calculate the expectation values

$$\langle O_{ij} \rangle_\rho = \langle \sigma_i \otimes \sigma_j \rangle_\rho, \tag{2a}$$

$$\langle N_{ij} \rangle_\rho = \left\langle \sigma_i \otimes \left(\frac{\sigma_j \pm \sigma_i}{\sqrt{2}} \right) \right\rangle_\rho . \tag{2b}$$

These numbers are used as the elements of the feature vector, $\mathbf{x} = \mathbf{x}(\rho)$, to be used as input for an ANN. I show that using only 4 particular operators of type (2b) a high accuracy classification is achieved. Also, when using the 15 linearly independent operators of the form (2a) and only pure states in the training phase, we may achieve partially correct classification of mixed states and high accuracy classification for highly entangled mixed states. This is an interesting result, since it is a very computationally demanding task to sample and classify mixed entangled states in higher dimensions.

The thesis is organized as follows. Chapter 1 starts with a reminder of some concepts in entanglement theory: The definition of entanglement, separability criteria, entanglement witnesses, etc. Chapter 2 explores the basics of artificial neural networks. Next, in chapter 3, there is an explanation of how data is built and classified to be used in training and testing of the different ANN models along with a visualization of the data. How the inputs to the ANN's are constructed, the different ANN architectures that were trained and tested is treated in in chapter 4. The main results are presented and discussed in chapter 5 . The thesis ends with a summary of the main results and the conclusions in chapter 6.

1 Quantum entanglement

Let us recall some basic concepts of quantum mechanics [1]. First we state some notation. We use $\mathcal{H}$ to denote a Hilbert space, where the state vectors, $|\psi\rangle$, of a quantum system live. These vectors are also refered to as *kets*. The vector space of operators that act on this Hilbert space is denoted by $\mathcal{L}(\mathcal{H})$.

1.1 Qubits and entanglement

A *qubit* is a state in any two-dimensional Hilbert space. A basis for this space is $\{|0\rangle, |1\rangle\}$, called the *computational basis*. Once the basis is fixed, vectors and operators acting on the Hilbert space may be represented with matrices. A general vector has the form of a column vector

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}. \tag{1.1}$$

These vectors are known as *kets*, and represent *pure states*. The Hermitian conjugate of a ket is called a *bra*, represented by a row vector

$$|\psi\rangle^\dagger = \langle\psi| = \alpha^*\,\langle 0| + \beta^*\,\langle 1| = \alpha^* \begin{bmatrix} 1 & 0 \end{bmatrix} + \beta^* \begin{bmatrix} 0 & 1 \end{bmatrix} = \begin{bmatrix} \alpha^* & \beta^* \end{bmatrix}. \tag{1.2}$$

The inner product of the Hilbert space is given by

$$\langle\psi|\phi\rangle = \begin{bmatrix} \alpha^* & \beta^* \end{bmatrix} \begin{bmatrix} \gamma \\ \delta \end{bmatrix} = \alpha^*\gamma + \beta^*\delta. \tag{1.3}$$

For any two vectors in a Hilbert space, this results in a complex number. The norm of a vector is a positive number defined as

$$\||\psi\rangle\| = \sqrt{\langle\psi|\psi\rangle}. \tag{1.4}$$

Vectors in the Hilbert state describing physical states must be normalized, that is $\||\psi\rangle\| = 1$. Two vectors are said to be *orthogonal* if $\langle\psi|\phi\rangle = 0$. A ket and a bra can be brought together in a different way called a *dyadic product*. Given two vectors $|\psi\rangle, |\phi\rangle \in \mathcal{H}$, their dyadic product, $|\psi\rangle\langle\phi|$, is an operator in $\mathcal{L}(\mathcal{H})$ defined by its action on another vector, $|\Psi\rangle$,

$$(|\psi\rangle\langle\phi|)\,|\Psi\rangle = \langle\phi|\Psi\rangle\,|\psi\rangle. \tag{1.5}$$

Given the Hilbert space of a qubit, $\mathcal{H}$, the basis for the vector space $\mathcal{L}(\mathcal{H})$ is given by $\{|0\rangle\langle 0|, |0\rangle\langle 1|, |1\rangle\langle 0|, |1\rangle\langle 1|\}$. A general operator in $\mathcal{L}(\mathcal{H})$ is represented by a 2 by 2 square matrix

$$\begin{aligned} O &= a\,|0\rangle\langle 0| + b\,|1\rangle\langle 0| + c\,|0\rangle\langle 1| + d\,|1\rangle\langle 1| \\ &= a\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix} + b\begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix} + c\begin{bmatrix}0 & 0\\ 1 & 0\end{bmatrix} + d\begin{bmatrix}0 & 0\\ 0 & 1\end{bmatrix} = \begin{bmatrix}a & b\\ c & d\end{bmatrix}. \end{aligned} \tag{1.6}$$

The *matrix elements* of an operator, O, in a given basis $\{|\psi_i\rangle\}_{i=1}^{d}$ are given by

$$O_{ij} = \langle\psi_i|\,O\,|\psi_j\rangle\,, \tag{1.7}$$

with $i, j = 1, 2, \ldots, d$. From equation (1.7) we see that the elements of the complex conjugate of an operator, O, are $[O^\dagger]_{ij} = O^*_{ji}$. The *expectation value* of an operator with respect to a given state $|\psi\rangle$ is defined as

$$\langle O\rangle_\psi = \langle\psi|\,O\,|\psi\rangle \tag{1.8}$$

Operators that are Hermitian, those such that $H = H^\dagger$, are called *observables* in quantum mechanics. These operators represent quantities that may be measured in a laboratory [24].

There is a set of very important operators in quantum information called Pauli matrices (or Pauli operators). They are denoted by σ_x, σ_y and σ_z (sometimes we replace x, y, z by $1, 2, 3$, respectively). Their matrix elements are

$$\begin{aligned} \sigma_x &= \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} \\ \sigma_y &= \begin{bmatrix}0 & -i\\ i & 0\end{bmatrix} \\ \sigma_z &= \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}. \end{aligned} \tag{1.9}$$

It is common to use σ_0 to denote the identity operator, $\mathbb{1}$. These four matrices form an orthonormal basis for $\mathcal{L}(\mathcal{H})$ [24].

1.2 Density operator

There is an alternative description for quantum state. Given a pure state, $|\psi\rangle$, the operator

$$\rho := |\psi\rangle\langle\psi| \tag{1.10}$$

is called the density matrix of the state. This operator contains all the information that the state vector does. Indeed, one can compute expectation values as

$$\langle O\rangle_\rho = \mathrm{Tr}(O\rho). \tag{1.11}$$

The normalization condition is given by

$$\mathrm{Tr}(\rho) = 1. \tag{1.12}$$

1.3 Mixed states

Quantum *mixed states* are defined as the convex sum of pure states

$$\rho = \sum_{i=1}^{N} p_i \left|\psi\right\rangle_i \left\langle\psi\right|_i , \tag{1.13}$$

where the $p_i's$ satisfy $0 \leq p_i \leq 1 \ \forall i$ and $\sum_{i=1}^{N} p_i = 1$. N can be any natural number. The $p_i's$ are interpreted as the probability that the mixture is found to be in state $|\psi_i\rangle$. The *purity* of a general state is defined as

$$P(\rho) = \mathrm{Tr}(\rho^2). \tag{1.14}$$

It is a measure of how close a state is to being a pure state. For pure states $P(\rho) = 1$, and for mixed states we have $P(\rho) < 1$. Mixed states arise naturally in physical settings.

1.4 Multipartite systems and entanglement

It is an axiom of quantum mechanics [24] that composite systems can be described by vectors in a Hilbert space that is the tensor product of the Hilbert spaces of its constituents. For bipartite systems this reads $\mathcal{H}_{AB} = \mathcal{H}_A \otimes \mathcal{H}_B$, where $H_{A,B}$ describe the Hilbert space of part A and B of the system, respectively. In general, for n subsystems we have $\mathcal{H} = \bigotimes_{I=1}^{n} \mathcal{H}_I$. Given a basis for a Hilbert space of dimension d_A, H_A, $\{\left|i^A\right\rangle\}_{i=1}^{d_A}$ and a basis for a Hilbert space of dimension d_B, H_B, $\{\left|j^B\right\rangle\}_{j=1}^{d_B}$, a basis for the product space is given by $\{\left|i^A\right\rangle \otimes \left|j^B\right\rangle\}_{i,j=1}^{d_A,d_B}$, and thus, the space has dimension $d_A d_B$. For qubits, we have $d_A = d_B = 2$ and the dimension of the product space is 4. The most general 2 qubit pure state is given by

$$|\psi\rangle = \sum_{i,j=1}^{2} c_{ij} |i^A\rangle \otimes |j^B\rangle = \sum_{i,j=1}^{2} c_{ij} \left|i\right\rangle \left|j\right\rangle = \sum_{i,j=1}^{2} c_{ij} \left|ij\right\rangle , \tag{1.15}$$

with $\{|i\rangle\}_{i=0}^{1}$ the computational basis of $\mathcal{H}_I$ $(I = A, B)$. The super-indices A and B are used to emphasise that the left state belongs to the first Hilbert space and the right state to the second. It may be omitted when there is no risk of confusion, keeping in mind that the position of the kets is what matters. A quantum pure state is called a *separable state* if there are state vectors $\left|\phi^A\right\rangle \in \mathcal{H}_A$ and $\left|\phi^B\right\rangle \in \mathcal{H}_B$ such that

$$|\psi\rangle = \left|\psi^A\right\rangle \otimes \left|\psi^B\right\rangle . \tag{1.16}$$

If no such vectors exists, the state is called *entangled*. This means that it is not possible to assign a single state vector to any of the subsystems. The description of the dynamics of the quantum system cannot be separated into the dynamics of its constituents; they are linked. This gives rise to many interesting and non-classical phenomena [1].

A mixed 2-qubit state is written as

$$\rho = \sum_{i=1}^{N} p_i |\phi_i\rangle\langle\phi_i|, \tag{1.17}$$

where $|\phi_i\rangle \in \mathcal{H}_{AB}$ and $0 \leq p_i \leq 1$ such that $\sum_{i=1}^{N} p_i = 1$. A mixed state, ρ_P, is called a *product state* if there are $\rho^A \in \mathcal{L}(\mathcal{H}_A)$ and $\rho^B \in \mathcal{L}(\mathcal{H}_B)$ such that $\rho_P = \rho^A \otimes \rho^B$. A mixed state is called *separable* if it is a convex combination of such product states

$$\rho = \sum_i p_i \rho_i^A \otimes \rho_i^B, \tag{1.18}$$

and it is called an *entangled* state if it may not be expressed in this way.

1.5 Entanglement detection

The problem then arises to determine whether a given state is entangled. Since the definition of entanglement is negative, this is a hard problem. In fact it has been shown to be NP-hard [13]. There are many criteria to determine if a state is entangled; in this work we use the PPT criterion [6, 7], since it's easy to implement. Given the full density matrix, it completely characterizes whether a 2 qubit state is entangled or not. Given a general mixed state of any dimension

$$\rho = \sum_{i,j}^{N} \sum_{k,l}^{M} \rho_{ij,kl} |i^A, k^B\rangle \otimes \langle j^A, l^B| \equiv \sum_{i,j}^{N} \sum_{k,l}^{M} \rho_{ij,kl} |i\rangle\langle j| \otimes |k\rangle\langle l|, \tag{1.19}$$

the *partial transpose* is given by

$$\rho^{T_B} := (\mathbb{1}_A \otimes T_B)(\rho) = \sum_{i,j}^{N} \sum_{k,l}^{M} \rho_{ji,kl} |i^A, k^B\rangle \otimes \langle j^A, l^B| \equiv \sum_{i,j}^{N} \sum_{k,l}^{M} \rho_{ji,kl} |i\rangle\langle j| \otimes |k\rangle\langle l|. \tag{1.20}$$

The sign of the smallest eigenvalue of ρ^{T_A} determines if a state is entangled [1]. If it is negative, then it is entangled, else it is separable. The usefulness of the PPT criterion is limited by the fact that the full density matrix of the state must be known. This means that measurements have to be performed on many identical copies of the state of interest to recreate the density matrix, i.e., quantum tomography must be performed [8]. It is convenient, then, to find methods that can distinguish entanglement using only partial information. A commonly used technique is that of entanglement witnesses, introduced by Terhal [25]. An *entanglement witness* is a Hermitian operator, $\mathcal{W}$, such that

$$\begin{aligned} &\mathrm{Tr}(\mathcal{W}\rho_s) \geq 0 \text{ for all separable states,} \\ &\mathrm{Tr}(\mathcal{W}\rho_e) < 0 \text{ for at least one entangled state.} \end{aligned} \tag{1.21}$$

The use of witnesses has the advantage that only partial information of the quantum state is needed, namely the sign of the expected value for such state, but it is

difficult to construct an entanglement witness for a given arbitrary quantum state [26]. Also, any one witness is not enough to cover a large amount of the set of entangled states, so most entangled states are not detected.

1.6 Entanglement measures

First we define *local unitary transformations*. An operator, $U \in \mathcal{L}(\mathcal{H})$ is called *unitary* if $UU^\dagger = \mathbb{1}$. An operator $U \in \mathcal{L}(\mathcal{H}_{AB})$ is called *local unitary tranformation* if there are operators $U_A \in \mathcal{L}(\mathcal{H}_A)$ and $U_B \in \mathcal{L}(\mathcal{H}_B)$ such that

$$U = U_A \otimes U_B \tag{1.22}$$

Local operations and classical communication (LOCC) is a method in quantum information theory where a local operation is performed on part of the system, and where the result of that operation is communicated classically to another part.

An entanglement measure, $E(\rho)$, is a function that quantifies the amount of entanglement that a state has. There are many ways to quantify it, but there are some desirable properties that these functions must have [1].

1. $E(\rho)$ goes to zero when ρ is separable.

2. It should be invariant under a local operations

$$E(\rho) = E(U_A \otimes U_B \rho U_A^\dagger \otimes U_B^\dagger).$$

3. Entanglement may not be created under LOCC, so it is reasonable to demand that $E(\rho)$ does not increase under such operations. That is, if Λ^{LOCC} is a positive map that may be implemented by LOCC, then

$$E(\Lambda^{\mathrm{LOCC}}(\rho)) \leq E(\rho).$$

4. An entanglement measure must be convex, that is, entanglement must be reduced when mixing states

$$E\left(\sum_k p_k \rho_k\right) \leq \sum_k p_k E(\rho_k).$$

In this work we use the entanglement measure of concurrence [5], defined for a 2 qubit pure states, $|\psi\rangle \in \mathcal{H}$, as

$$C(|\psi\rangle) = \sqrt{2\left(1 - \mathrm{Tr}\left[(\rho^{T_A})^2\right]\right)}. \tag{1.23}$$

The extension to mixed states is given by

$$C(\rho) = \max\left\{0, \lambda_1 - \lambda_2 - \lambda_3 - \lambda_4\right\}, \tag{1.24}$$

where the λ_i's are the eigenvalues of the matrix $X(\rho) = \sqrt{\sqrt{\rho}\tilde{\rho}\sqrt{\rho}}$ and $\tilde{\rho} = (\sigma_y \otimes \sigma_y)\, \rho^* \, (\sigma_y \otimes \sigma_y)$, and ρ^* is a matrix whose elements are the complex conjugate of the matrix ρ. This measure is used due to having a closed form extension for mixed states which is easy to implement in code.

The set of all 2 qubit states is usually represented schematically as in figure 1.1. This picture shows the convex nature of the set of all states and that of the set of all separable states. Also, given an entanglement witness, $\mathcal{W}$, the set of all states such that $Tr(\mathcal{W}\rho) = 0$ defines a hyperplane in this space. The hyperplane divides the space in two areas, on one side we have all entangled states detected by $\mathcal{W}$ and on the other we have all separable states and all entangled states not detected by the witness. This is also shown on figure 1.1.

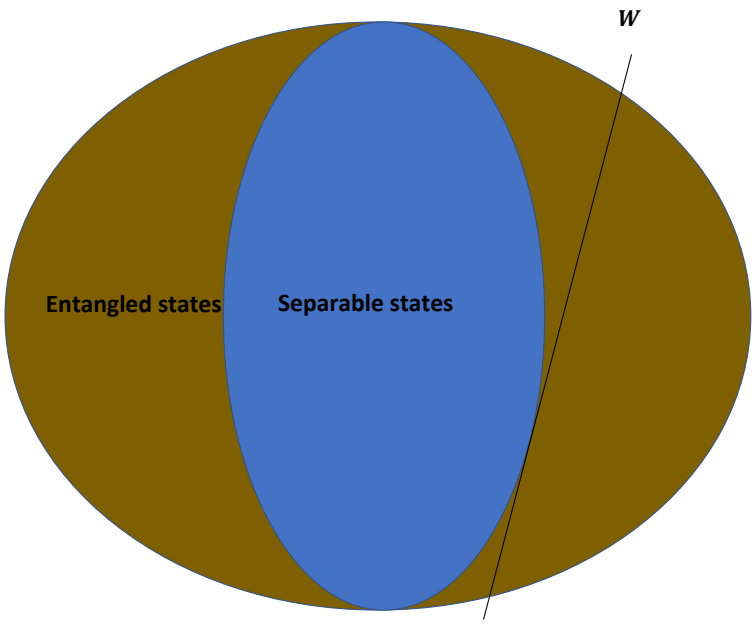

Figure 1.1: Schematic representation of the space of quantum states. The set of separable states is convex, as is the space of all states. An entanglement Witness defines a hyperplane in this space.

2 Machine Learning

Machine learning (ML) is a subfield of artificial intelligence with the goal of developing algorithms capable of *learning*, finding patterns and being able to make a decision for the output given an input, from data automatically. Artificial neural networks (ANN) are ML algorithms that aimed to be a computational model of biological learning. They are a nonlinear model for supervised learning, a type of learning in which the user provides the program with many examples. Nowadays it is more common to refer to ANN's as deep learning algorithms, since any ANN with more than 3 layers is considered to be a deep network. The treatment in this chapter follows [27] and [28].

2.1 Origins of neural networks

The earliest predecessors of modern learning were simple linear models designed to take a set of n inputs, $x_1, \ldots, x_n$ and a set of *weights* $w_1, \ldots, w_n$ to compute the output $f(\mathbf{x}, \mathbf{w}) = x_1 w_1 + \cdots + x_n w_n = y$. The McCulloch-Pitts neuron [29] was an early model of brain function. This model could distinguish between to classes of inputs using the sign of the output of $f(\mathbf{x}, \mathbf{w})$. The weights were set by a human operator. Later, in the 50's, the Rosenblatt perceptron [30] became the first model that could *learn* the weights that defined the classes using examples provided by a human operator. Learning in this context means that the program automatically adjusts the weights in order to comply with the examples provided. The adaptive linear element (ADALINE) [31] returned the value of $f(\mathbf{x})$ to predict a real number and could learn to predict these numbers from data.

These learning algorithms greatly influenced the modern approach of ML. The training algorithm used to adapt the weights of the ADALINE was a special case of an algorithm called *stochastic gradient descent*. Slightly modified versions of this algorithm remain the dominant training algorithm for deep learning models today. The neural networks used in this work are for binary classification, so the following sections deal only with this particular case. In this case the labels of the data sets are 0 and 1, which can represent anything. In this work 0 represents that the state is entangled, while 1 represents a separable state.

2.2 Supervised learning algorithms

Supervised learning is a class of deep learning algorithms. The term supervised comes from the fact that the algorithm is provided with a labeled dataset. The labelling may be done by another program or human operators. The data consists of a *feature vector*, $\vec{x}_i$, an array of numbers that represent the data of interest, and its corresponding labels, y_i,

$$\mathcal{D} = \{(\mathbf{x}_i, y_i) | i = 1, \ldots, N\}. \tag{2.1}$$

Each vector of this set is referred to as an *example*. The features for the set of examples is usually encoded with a *feature matrix*, $\mathbf{X}$, that is made up of the feature vectors as its columns,

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \ldots & \mathbf{x}_N \end{bmatrix}. \tag{2.2}$$

The neural network (for binary classification) is a function that takes as its input the feature vector of an example and outputs a number that is to be interpreted as the probability that the particular training example belongs to a class. This is referred to as the *model*, $f(\mathbf{x}; \theta) \in [0, 1]$. It is common to use the notation $f(\mathbf{X}; \theta)$. This is to be understood as

$$f(\mathbf{X}; \theta) = \begin{bmatrix} f(\mathbf{x}_1; \theta) & f(\mathbf{x}_2; \theta) & \ldots & f(\mathbf{x}_N; \theta) \end{bmatrix}. \tag{2.3}$$

This model's parameters, θ, have to be adjusted in such a way that it produces the correct *predictions*, y. This process is known as *training*, and is done automatically by the program in some algorithmic manner. The performance of such a model is measured using a function, $\mathcal{C}(y, f(\mathbf{X}; \theta))$, called the *cost function*, that allows us to judge how closely the model predicts the correct labels. The value of the parameters that minimizes the cost function will be denoted by θ^*. There is a standard recipe used to find the parameters that minimize the cost function. The first step is to randomly divide the data set into two mutually exclusive groups, $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$, called *training set* and *test set*, with the test set typically having around 90% of the data points. The model is then fit to the data using only the data from the training set, done by minimizing the cost function. Finally, the performance of the model is evaluated by computing the cost function using the test set. This gives a measure of the predictive power of the model by testing it on previously unseen data.

2.3 Neural network architecture

The basic unit of an artificial neural network is called a *neuron*, which can be seen in figure 2.1, and can be regarded as a function that takes a vector, $\mathbf{a}^{[0]}$, of n_0 entries as input and outputs the number

$$a^{[1]} = g^{[1]}(\mathbf{w}^{[1]} \cdot \mathbf{a}^{[0]} + b^{[1]}). \tag{2.4}$$

The elements of the vector $\mathbf{w}^{[1]}$ are the weights and $b^{[1]}$ is called the bias. The function $g^{[1]}$ is called *activation function* and its argument is usually expressed as

$z^{[1]} = \mathbf{w}^{[1]} \cdot \mathbf{a}^{[0]} + b^{[1]}$. An ANN is made up of many neurons (also called units) arranged in a layered network, such as in figure 2.2. In such an arrangement, equation (2.4) changes to

$$\mathbf{a}^{[l]} = g^{[l]}(\mathbf{W}^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}), \tag{2.5}$$

where $\mathbf{W}^{[l]}$ is now a matrix of weights that connects neurons from layer $l-1$ to neurons in layer l, $\mathbf{b}^{[l]}$ is now a vector and the superscripts represent the layer that each element belongs to. The number of neurons in each layer is denoted by $n^{[l]}$. The input the model is the feature vector $\mathbf{x} = \mathbf{a}^{[0]}$. For a binary classification problem, there is one output neuron. The output is denoted as $\tilde{y} \in [0, 1]$. As discussed in the previous section, this number is meant to represent the probability that the input is classified as 0 or 1. The weights and biases are the parameters (θ from the previous section) that are to be optimized.

Now we introduce a little bit of new notation. Given that our data consists on many examples (training examples from the train set and test examples from the test set), it is common to add a superscript to clarify to which example one is referring to. For instance, the activations in layer l of the i-th training example are written as the vector $\mathbf{z}^{[l](i)}$, while the k-th entry of the vector is written as $z_k^{[l](i)}$. Whenever there is no risk of confusion, the notation of equation (2.2) may be used, where the subscripts of each feature vector refers to the example the vector belongs to.

2.4 Activation functions

Activation functions are used to introduce a non-linearity to the model. If all activation functions were linear, the resulting model would always be equivalent to one with a single hidden layer and many hidden units. There are many activation functions used in modern ML, but in this work only two are used: The *ReLU* (Rectified Linear Unit) activation and the *sigmoid* activation. The ReLU function is defined as

$$ReLU(x) = \max(0, x), \tag{2.6}$$

while the sigmoid is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.7}$$

These functions are defined on real numbers, so when a vector is used as argument, it is to be understood as an element-wise operation. That is

$$g(\mathbf{z}) = \begin{bmatrix} g(z_1) \\ g(z_2) \\ \vdots \\ g(z_n). \end{bmatrix}. \tag{2.8}$$

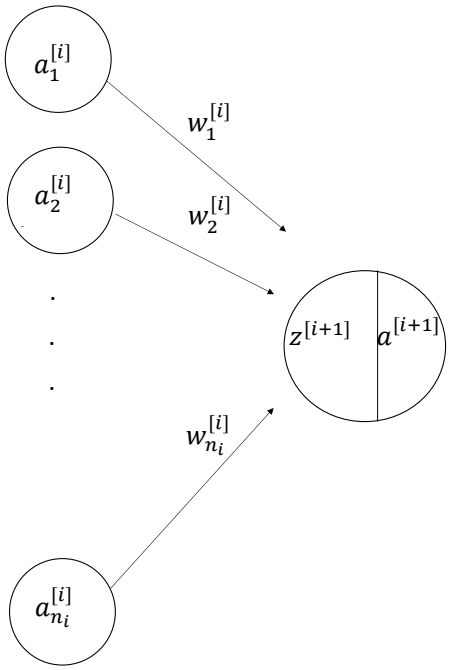

Figure 2.1: A neuron can be regarded as a function. Here, the superscript refers to the layer where a neuron might be and the subscript refers to the j-th element of a vector.

.

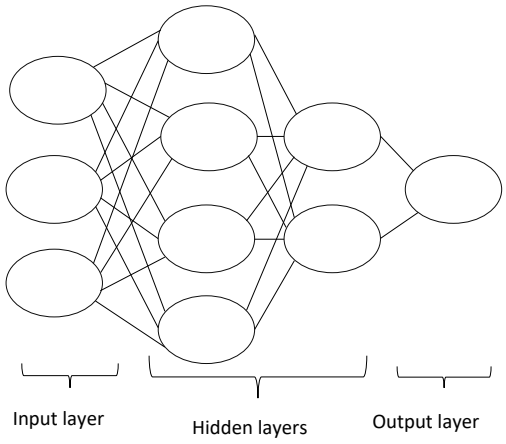

Figure 2.2: An artificial neural network is a collection of neurons. Each neuron is referred to as a unit, units in the hidden layers are called hidden units.

.

2.5 Training

The process of *training* a neural network refers to the process of adjusting the weights of the model, $\mathbf{W}$, in order to minimize the cost function. There are many algorithms used to perform this training process. One of the most popular is *gradient descent*. This is an iterative algorithm, where the values of θ are varied in the direction opposite to that of the gradient, see figure 2.3. The number of iterations is often called *epochs*. The value of the parameters for the epoch $t+1$ is updated according to the rule

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta \mathcal{C}(y, f(\mathbf{X}; \theta_t)), \tag{2.9}$$

where ∇_θ is the gradient with respect to the parameters and the term η_t is called the *learning rate* for epoch t. This method will converge to a local minimum of the cost function if the learning rate is small enough; however, choosing a small learning rate means that the number of steps required to reach the minimum increases. On the other hand, choosing a large learning rate means you can overshoot the minimum and the algorithm becomes unstable.

The problem then becomes how to compute gradients. In practice this is done with an automatic differentiation algorithm called *backpropagation*. The reader is referred to chapter 6.5 of [27] for a complete description of the algorithm. This algorithm is implemented in most machine learning libraries, such as the one used in this work.

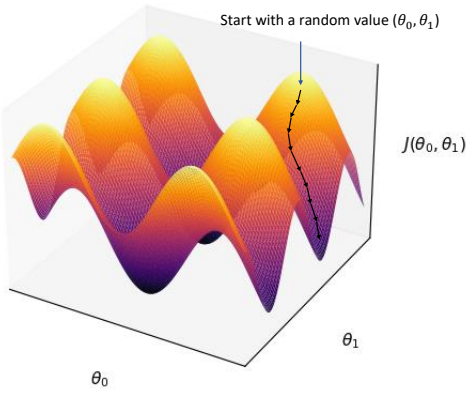

Figure 2.3: The gradient descent algorithm. A small step is taken in the opposite direction of the gradient in order to get to a local minimum. In this picture the cost function is represented as having two parameters, in actual applications the number of parameters in much larger than that.

2.6 Cost function

There are many functions that can be used as cost functions. A common choice, and the one used in this work, is to use the binary cross entropy, defined for a single

training example, $i \in \{1, 2, .., N\}$, as

$$H_i(\mathbf{x}_i, y_i) = -y_i \log\left(f(\mathbf{x}_i, \theta)\right) - (1 - y_i) \log\left(1 - f(\mathbf{x}_i, \theta)\right). \tag{2.10}$$

This function compares the predicted probability to the actual label (0 or 1). It penalizes a bad prediction based on the distance from the correct value.

There are various options on how to define the cost function. One of them is to define it as the average of (2.10) over all training examples

$$C(\mathbf{y}, f(\mathbf{X}, \theta)) = \frac{1}{N} \sum_{i=1}^{N} H_i(\mathbf{x}_i, y_i). \tag{2.11}$$

In practice, this is a very inefficient way to compute the cost. One can divide the training set into *mini-batches* of a given size and then compute the cost at each step of gradient descent as

$$C_{mb}(\mathbf{y}, f(\mathbf{X}, \theta)) = \frac{1}{N_{\text{mb}}} \sum_{i=kN_{\text{mb}}}^{(k+1)N_{\text{mb}}} H_i(\mathbf{x}_i, y_i), \tag{2.12}$$

where k is a random positive number sampled such that $(k+1)N_{\text{mb}} < N$. This is called *mini-batch gradient descent* (as opposed to batch gradient descent, when the whole batch of training examples is used to compute the cost function). Usual choices for N_{mb} are $32, 64, 128$, etc, but if $N_{\text{mb}} = 1$ the algorithm is called *stochastic gradient descent*. Clearly, at each step of mini-batch gradient descent, a different function is being minimized, but it reaches local minima of (2.11). Using mini-batch gradient descent is like taking fast steps down a hill, even if each step is not in the direction of greatest slope, while using batch gradient descent is like taking very slow, very deliberate steps downhill (see figure 2.4).

There are many variations of the gradient descent algorithm. Chapter 4 of reference [28] explores the limitations and variations of this particular algorithm. In this work, the ADAM [32] variation is used for training.

2.7 Bias and variance

The value of the cost function for the best fit model on the training set is called the *in-sample error*, and the value of the cost function on the test set is called the *out-sample error*

$$E_{\text{in}} = \mathcal{C}\left(\mathbf{y}_{\text{train}}, f\left(\mathbf{X}_{\text{train}}; \theta\right)\right) \tag{2.13a}$$

$$E_{\text{out}} = \mathcal{C}\left(\mathbf{y}_{\text{test}}, f\left(\mathbf{X}_{\text{test}}; \theta\right)\right). \tag{2.13b}$$

It is always the case in ML that the model that best fits the training set is not the model that best generalizes the test set. The difference between E_{in} and E_{out} is called *variance*. The difference between the prediction of the values by the ML model

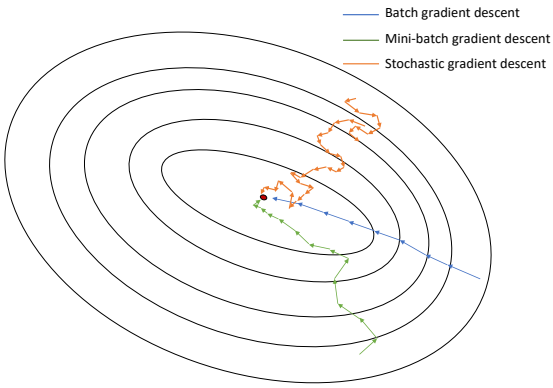

Figure 2.4: Schematic comparison of the gradient descent algorithms. Batch gradient descent always reaches a local minimum, but takes a lot of time. Batch gradient descent and stochastic gradient descent are much faster, but may not reach the minimum.

and the correct value is called *bias*. This leads to the problem of *overfitting* and *underfitting*. Overfitting occurs when the model performs very well on the training set but the performance on the test set is much lower. In these cases we say that there is *high variance*. Underfitting occurs when the model does not fit the train set well. In this cases there is high bias. The challenge is to find a model with the right complexity such that it reduces both bias and variance. A schematic representation of bias and variance may be seen in figure 2.5.

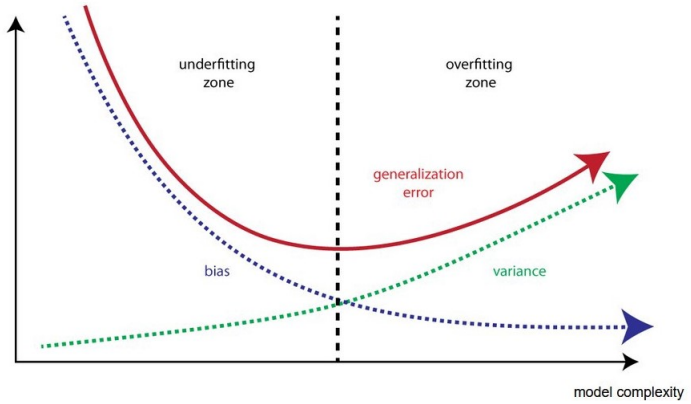

Figure 2.5: The vertical axis represents the model's prediction error. The red curve represents the average error on the test data. The middle line shows where the minimum of E_{out} is.

2.8 Regularization

There are many strategies designed to reduce the test set error, even at the expense of the training set error. These strategies are known as *regularization*, and there are many forms available. Some put extra constraints on a ML model, such as adding restrictions on the parameter values. Some add extra terms in the cost function that make the parameter values tend to lower values. If chosen carefully, these extra constraints and penalties can lead to better performance on the test set. Other times, these constraints and penalties are designed to express a generic preference for a simpler model class that does better at generalization. In the context of neural networks, most regularization methods aim to trade increased bias for reduced variance. In this work two forms of regularization are used: Batch normalization and dropout.

2.9 Dropout

The basic idea of dropout is to prevent overfitting by reducing spurious correlations between neurons of the network by temporarily removing some neurons and their connections (along with the trainable parameters) in each layer at each step of training. This is why it is called dropout, some neurons are dropped at each training step, see figure 2.6.

In practice, one adds a dropout layer before each layer of the neural network. This layer goes neuron by neuron and removes them with a probability p.

2.10 Batch normalization

Batch normalization is based on the observation that training in neural networks works best when the inputs are centered around zero with respect to the bias. The reason for this is that it prevents neurons from saturating and gradients from vanishing in deep neural networks. In the absence of such centering, changes in the parameters in lower layers can give rise to saturation effects in in higher layers. To avoid this, one introduces additional layers called *BatchNorm* layers that standardize the inputs by the mean and variance of the mini-batch.

Consider a layer, l, which has d units and whose inputs for the i-th example are $[z_1^{[l](i)}, z_2^{[l](i)}, {}_d^{[l](i)}]$. We standardize each dimension as

$$z_k^{[l](i)} \to \hat{z}_k^{[l](i)} = \frac{z_k^{[l](i)} - \mu_k^{[l]}}{\sigma_k^{[l]}}, \tag{2.14}$$

where $\mu_k^{[l]} = \frac{1}{N_{\text{mb}}} \sum_{i=1}^{n^{[l]}} z_k^{[l](i)}$ is the mean of the k-th entry over all examples in the mini-batch, and $\sigma_k^{[l]} = \frac{1}{N_{\text{mb}}} \sum_{i=1}^{n^{[l]}} (z_k^{[l](i)} - \mu_k^{[l]})^2$ is the variance of the k-th entry over all examples in the mini-batch. Then we compute the value

$$\tilde{z}_k^{[l](i)} = \gamma_k^{[l]} \hat{z}_k^{[l](i)} + \beta_k^{[l]}. \tag{2.15}$$

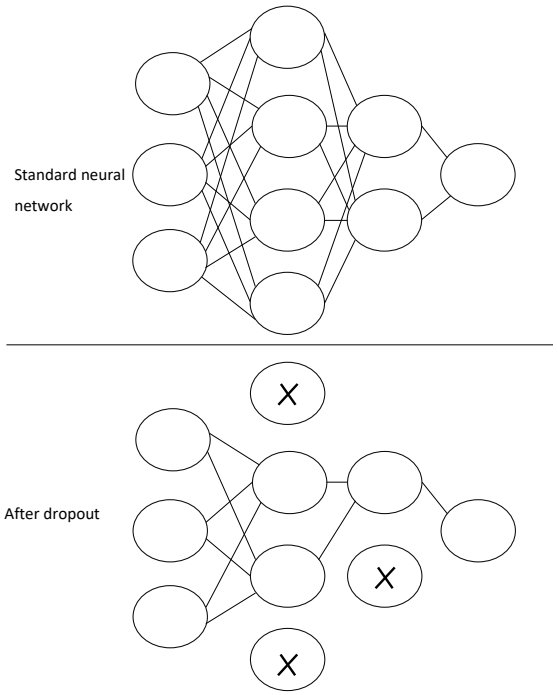

Figure 2.6: Some neurons and its connections are removed during each step of the training procedure with some probability, p. This prevents overfitting by reducing correlations between neurons.

We add a layer that computes the values (2.15) before calculating the activation function. The parameters $\gamma_k^{[l]}$ and $\beta_k^{[l]}$ become learnable parameters of the model which can be updated when performing gradient descent. In practice, batch normalization serves as a regularizer, although it is not fully understood why (see section 9.4 of [28]). One plausible explanation is that in batch normalization, the gradient for a sample depends not only on the sample itself but also on all the properties of the mini-batch. Since a single sample can occur in different mini-batches, this introduces additional randomness into the training procedure which seems to help regularize training.

2.11 Single number evaluation metric

Once a model is trained, we wish to have some performance metrics. For a binary classifier there exists a number of single number evaluation metrics. First we define the confusion matrix for a given ANN model. The *confusion matrix* is a 2 by 2 table that reports the number of true positives(TP), false positives(FP), true negatives(TN) and false negatives(FN) that a model outputs on a certain test set, see table 2.1. The single number evaluation metrics used in this work are precision, recall and the F_1

score, which are calculated from the confusion matrix. In this work, positive refers to separable (1) and negative to entangled (0).

Actual class \ Predicted class	P	N
P	TP	FN
N	FP	TN

Table 2.1: Confusion matrix. The diagonal contains the correctly labeled inputs of the model.

Precision is defined as the fraction of positive predictions reported by the model that were correct, and we denote it with a $\mathcal{P}$. *Recall* is the fraction of positive examples that were correctly classified, and we denote it with an $\mathcal{R}$,

$$\mathcal{P} = \frac{TP}{TP + FP} \tag{2.16}$$

$$\mathcal{R} = \frac{TP}{P} = \frac{TP}{TP + FN}. \tag{2.17}$$

The F_1 score is then defined as the harmonic mean of the precision and recall, and it is a measure of the model's accuracy.

$$F_1 = \frac{2TP}{TP + TN + FP + FN}. \tag{2.18}$$

Finally, the accuracy is defined as the ratio of correctly classified examples

$$A = \frac{n_{\text{correct}}}{n_{\text{total}}} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{2.19}$$

3 Data generation and visualization

The data of interest in this work consist of quantum states and a label that tells us if the state is separable or not, so the data generating process is that of sampling 2 qubit states. These states should be spread out in all space, so that we may correctly classify entangled states. A uniform sampling of pure states may be done, but such a uniform sampling is not possible for mixed states [33].

3.1 State and operator sampling

To create random states, three sampling schemes are used. First, pure states are sampled by generating two random unitary matrices [34], U_1 and U_2, according to the Haar measure on $U(2)$. The *Harr measure* is defined as a non zero measure over a group, G, $\mu : G \to [0, \infty]$, such that for all $S \subset G$ and $g \in G$, $\mu(gS) = \mu(Sg) = \mu(g)$. The state

$$|\psi_{\text{sep}}\rangle = U_1 \otimes U_2 |0, 0\rangle \tag{3.1}$$

is a pure random separable state. A pure random state is then generated by making a superposition of a random number of states of the form (3.1).

$$|\psi\rangle = \sum_{i=1}^{M} |\psi_{sep}^{(i)}\rangle, \tag{3.2}$$

where M can be any natural number. In this work that number may be between 1 and 4, as it is found that the distribution in purity and concurrence of the sampled states does not vary much for $M > 4$. These states are called Haar distributed.

For mixed states, the methods proposed in [33] are used, which result in the Hilbert-Schmidt(HS) and Bures(B) induced distributions. A random state from the HS distribution is given by

$$\rho_{HS} = \frac{GG^\dagger}{Tr(GG^\dagger)}, \tag{3.3}$$

where G is a random 4 by 4 matrix pertaining to the Ginibre ensemble [35], which are matrices with Gaussian a distribution for their elements. A random state from the Bures distribution is given by

$$\rho_B = \frac{(\mathbb{1} + U)GG^\dagger(\mathbb{1} + U)}{Tr[(\mathbb{1} + U)GG^\dagger(\mathbb{1} + U)]}, \tag{3.4}$$

where U is a random unitary matrix distributed according to the Haar measure on $U(4)$. Figure 3.1 shows a histogram of the purity of states generated through the two latter methods. It is seen that the HS distribution results in a vast majority of states with low purity and no pure states. In contrast, the Bures distribution produces states with a higher average purity. The first method produces only states with purity equal to 1. A histogram of the concurrence is presented in figure 3.2. Here it is seen that the Haar distributed states are more uniform, the HS distribution produces mostly weakly entangled states, and the Bures distribution produces states with a higher concurrence on average.

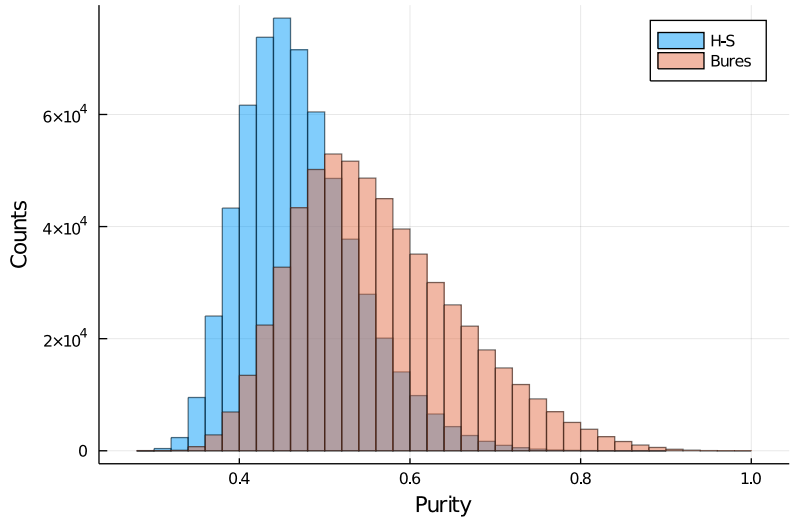

Figure 3.1: Distribution of purity of the random states. Both of them, Bures and HS, result in a majority of very mixed states and very few purer states. In particular, the H-S distribution produces very few states with a higher purity. This figure uses 600,000 states sampled from each distribution.

3.2 Visualizing the data

The task of the neural network is to classify a state given the inputs defined by (2a) and (2b). In order to get a sense of what the neural networks are doing, images that show the way that the data is distributed are presented.

3.3 The 15 linearly independent O_{ij} operators

First consider the 15 linearly independent operators of the form (1a). The numbers (2a) can be considered axes of a plot for the data points. The states used to calculate the expected values (2a) may be pure or mixed. By taking pairs of these numbers as axes for 2D plots, we get images like that of figure 3.3 for pure states.

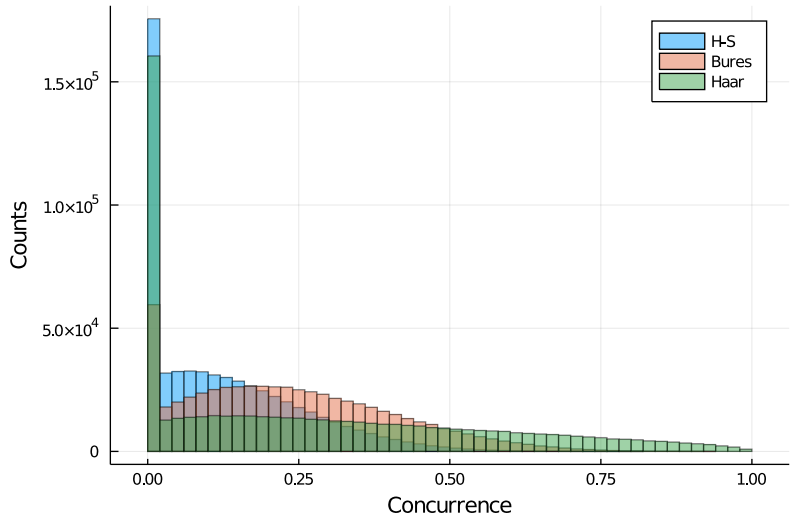

Figure 3.2: Distribution of concurrence of the random states. This distribution is shown for the three sampling schemes. The most uniform among them is the one induced by the Haar measure, but this produces only pure random states. The HS distribution produces mostly weakly entangled states and the Bures distribution produces very few strongly entangled states. 600,000 states from each distribution were used.

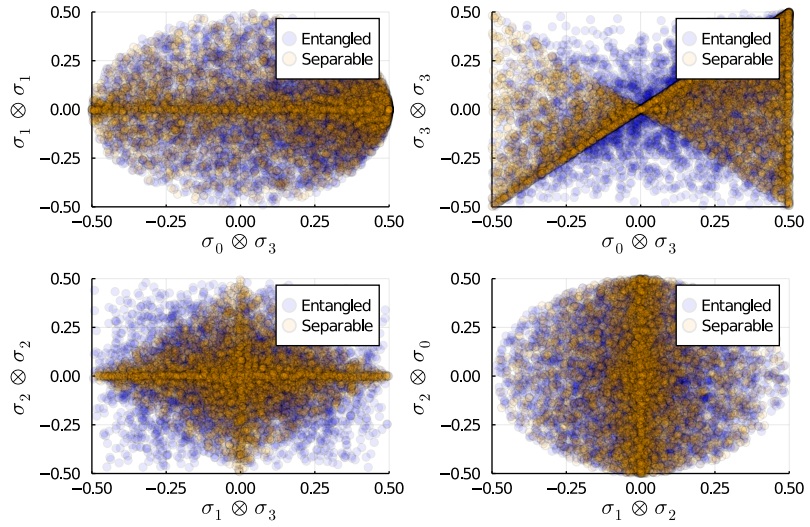

Figure 3.3: Data visualization for pure states using the Pauli matrices products in the axes. These are some of the figures that appear, many different axes produce similar images. All figures were produced using 6,000 states.

Patterns are seen to emerge and there are some representative shapes. Separable states tend to cluster together and form shapes. For instance, for certain axes, separable states form rhombi and cluster in its diagonals. In figure 3.4 only one of the shapes is shown and a much richer structure is seen by showing the different "layers" of concurrence in the image. As entanglement grows, the data spreads more and moves away from the clustering of separable states. In figure 3.5 it is seen that

the states that most closely align with the separable states are those that are weakly entangled, and as entanglement increases, the structure found for separable states is lost. This suggests that weakly entangled states will be the most difficult to categorize, since they may be confused for separable states. This is to be expected, since these are the states closest to the border between separable and entangled states.

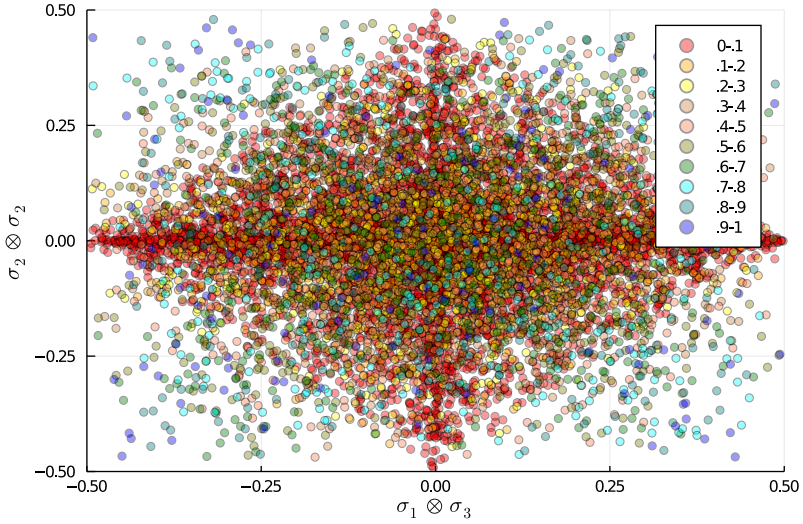

Figure 3.4: A closer look at the data visualization. Here the colors indicate a range of possible values of the concurrence. The red states are separable while the blue ones are the most entangled.

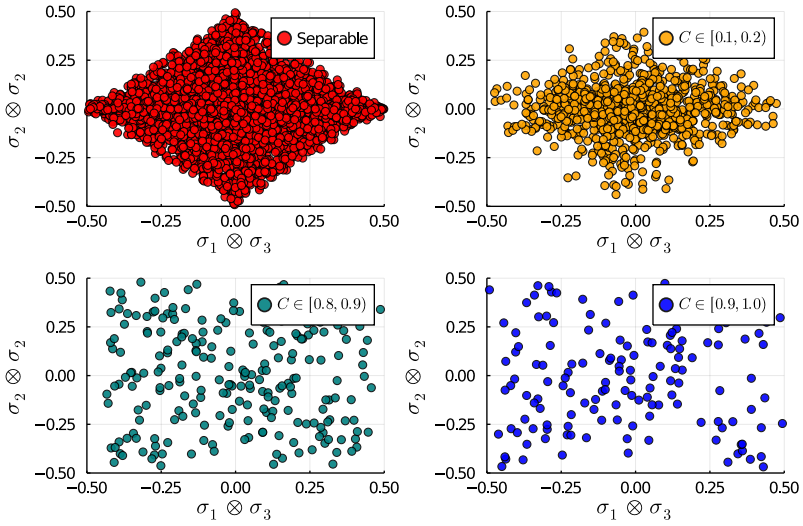

Figure 3.5: Layers of concurrence for pure states using Pauli operators products in the axes. Strongly entangled states are more likely to be found outside of the cluster of separable states, while weakly entangled states mostly overlap in this area with separable states.

Another interesting detail found in this images is the fact that there are some zones where we may exclude separable states. Namely, in figure 3.4 there are 4 zones

in which no separable states may be found. It seems reasonable to speculate that the ANN model learns to identify the areas where no entanglement is present by using all the expected values as input.

Mixed states produce very different images. Using mixed states sampled from the Bures ensemble to calculate the expected values results in figures such as in 3.6, in which a similar structure is seen, but the decision boundaries are not as clearly defined. In fact, there are axes that do not seem to show such a boundary. There are more mixed separable states that spread out. At the same time, there aren't many highly entangled states produced in this sampling scheme.

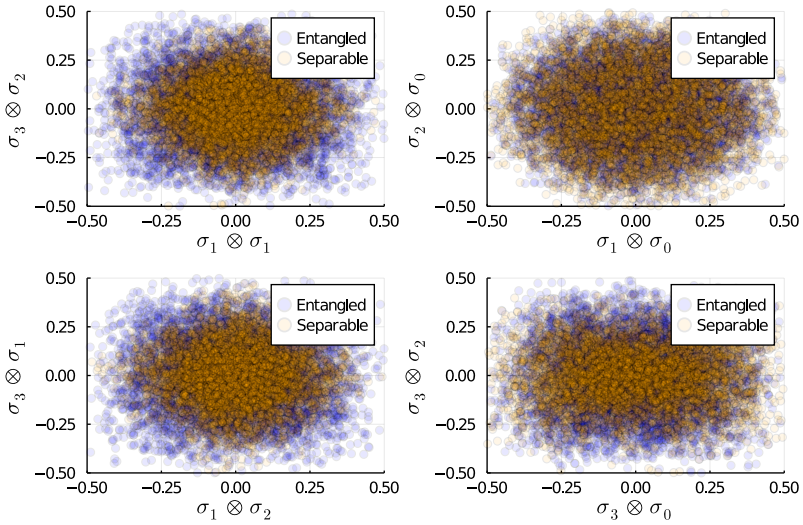

Figure 3.6: Various data distributions using Pauli operators and mixed states sampled from the Bures ensemble. The boundaries where separable states may be excluded are less clear, and for some axes do not appear to exist. 6,000 states where sampled to make this figures.

Looking at figures in layers of concurrence (figure 3.7) it is found that the separable states tend to cluster together, weakly entangled states distribute themselves quite near the separable states and strongly entangled states spread out and lose structure, as in the separable case, but in this case there appear to be much smaller decision boundaries.

A 3D plot shows how the algorithm may find separable states. As seen in figure 3.8, the zones where no separable states may be found are large. The exclusion zones in the higher dimensional case may be large enough so that a very accurate classification can be performed. Also, figure 3.9 suggests that a classification for mixed states is possible, since there are many axes that show outlying separable states. Again, in a higher dimensional space, better boundaries may be found.

3.4 Sets of N_{ij} operators

The operators (1a) are orthogonal, but separability criteria based on Bell's inequalities use non orthogonal operators, such as those of the form (1b). In this work, these

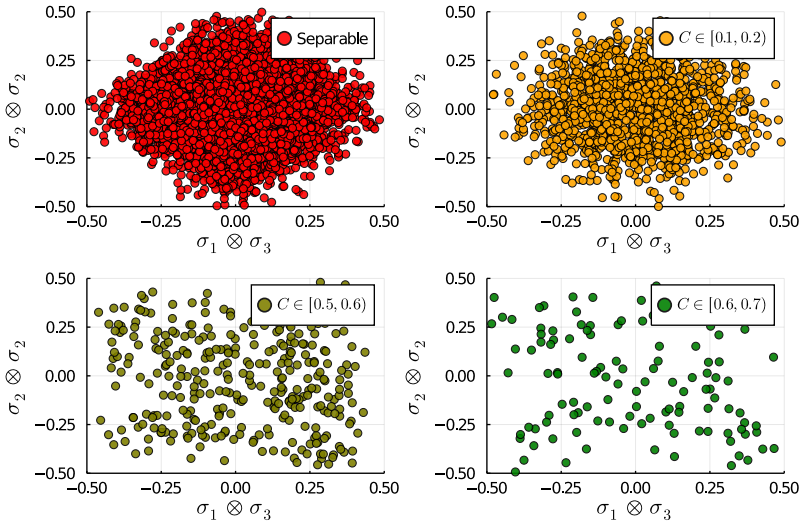

Figure 3.7: Layers of concurrence for a particular set of axis using random mixed states. We observe the loss of structure and spreading of states. A lower concurrence interval is used, since there are very few strongly entangled states.

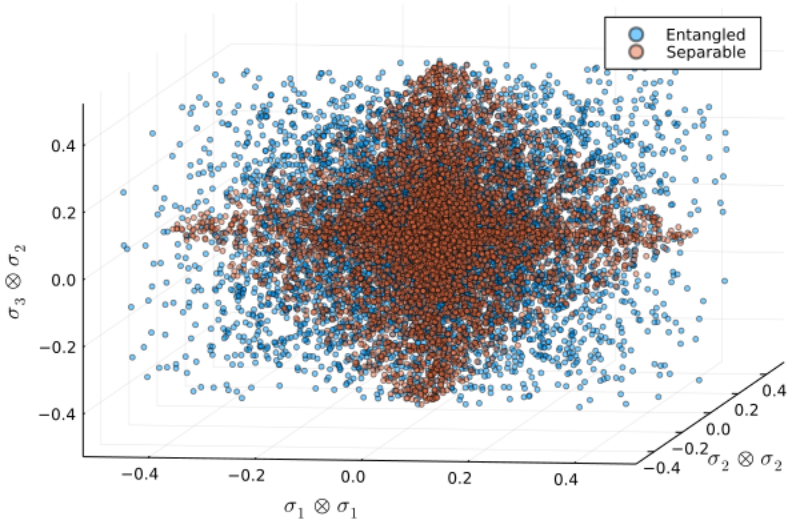

Figure 3.8: A 3D plot for pure states. Decision boundaries are quite clear and suggest that they become more apparent in more dimensions. The axis used were chosen randomly from the O_{ij} operators.

operators are sampled randomly through the following procedure:

- Sample 3 numbers between 0 and 3 without replacement, call them i, j, k.
- Sample between $+1$ and -1, call it s.
- Add the operator $\sigma_i \otimes \frac{\sigma_j + s\sigma_k}{\sqrt{2}}$ to the list.
- With probability 0.5, add the operator $\sigma_i \otimes \frac{\sigma_j - s\sigma_k}{\sqrt{2}}$ to the list.

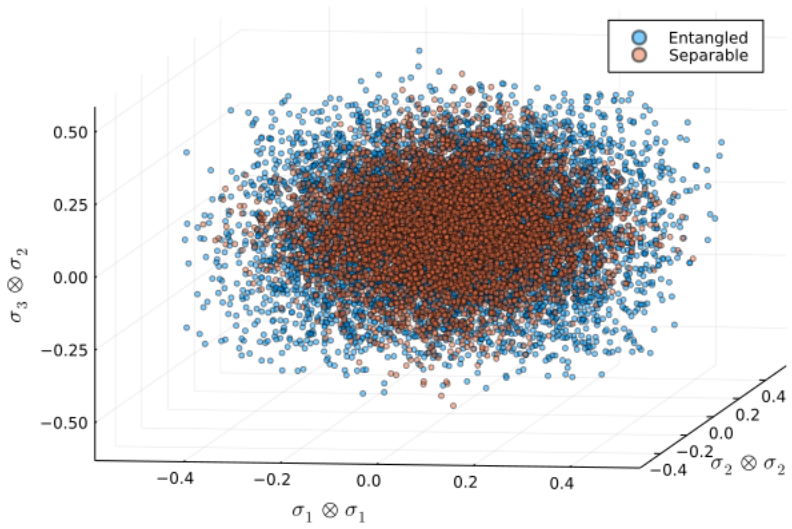

Figure 3.9: A 3D plot for mixed states sampled form the Bures ensemble. The decision boundaries are not as clear, there appear to be separable states everywhere in space, but most separable states cluster at the center and many entangled states are spread out. The axis used are chosen randomly from the O_{ij} operators.

- Repeat 10 times.
- Keep only the unique operators that result.

In this way, we may characterize an operator of the form (1b) with the numbers $[i, j, k, s]$, where s is the sign and may be $+$ or $-$. One such realization of this procedure yielded the numbers

$$\begin{aligned}
[0, 1, 3, -1] \\
[3, 0, 2, 1] \\
[3, 0, 2, -1] \\
[1, 3, 0, -1] \\
[1, 3, 0, 1] \\
[1, 0, 2, 1] \\
[3, 2, 0, 1] \\
[3, 1, 0, 1] \\
[2, 0, 3, 1].
\end{aligned} \tag{3.5}$$

These numbers then determine the 10 operators used to construct the feature vector (2b). Some 2d plots that arise from these axes are shown in figure 3.10. We notice similar patterns as in the case of orthogonal operators, some exclusion zones, where only entangled states are found, and clustering of the separable states. In figure 3.11 the layers of concurrence are shown for a particular set of axes. Again, highly entangled states are found far from the central clustering of separable states.

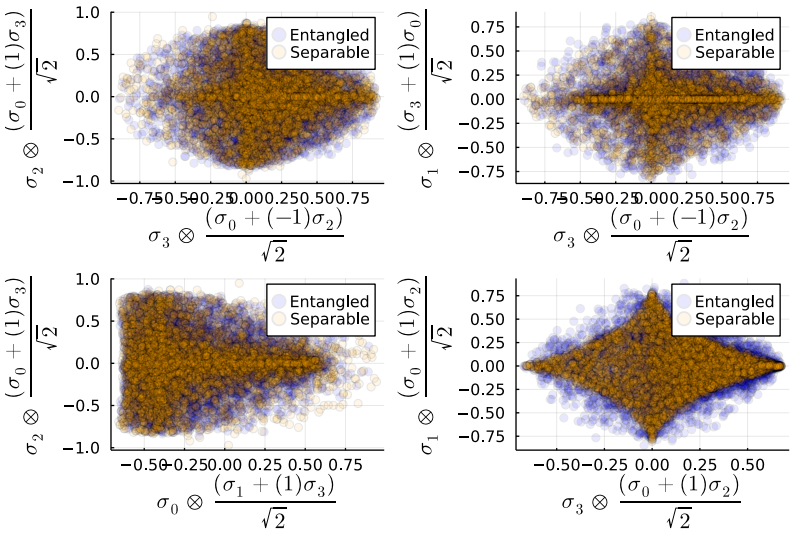

Figure 3.10: Data visualization using non orthogonal products of Pauli matrices. The boundaries are no longer lines, but rather curves, owing to the non orthogonality. Some zones of where only entangled states may be found are apparent. 6000 states were used for all plots.

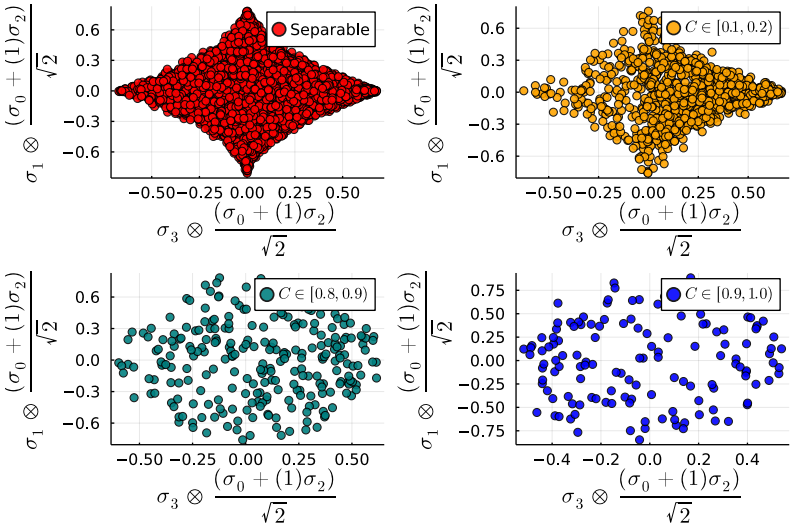

Figure 3.11: Layers of concurrence for the N_{ij} operators. We again observe de-clustering with increasing concurrence.

There is a set of operators that results in a clear separation between pure entangled

and separable states. The operators

$$\begin{aligned} &\sigma_2 \otimes \frac{(\sigma_0 - \sigma_1)}{\sqrt{2}} \\ &\sigma_2 \otimes \frac{(\sigma_0 + \sigma_1)}{\sqrt{2}} \\ &\sigma_0 \otimes \frac{(\sigma_0 - \sigma_1)}{\sqrt{2}} \\ &\sigma_0 \otimes \frac{(\sigma_0 + \sigma_1)}{\sqrt{2}} \end{aligned} \tag{3.6}$$

produce the plots seen in figure 3.12. These images do not show much, aside from very large zones were only entangled states reside, but the 3D graph shown from two perspectives in figures 3.13 and 3.14 reveals a drastic separation of the entangled and separable states.

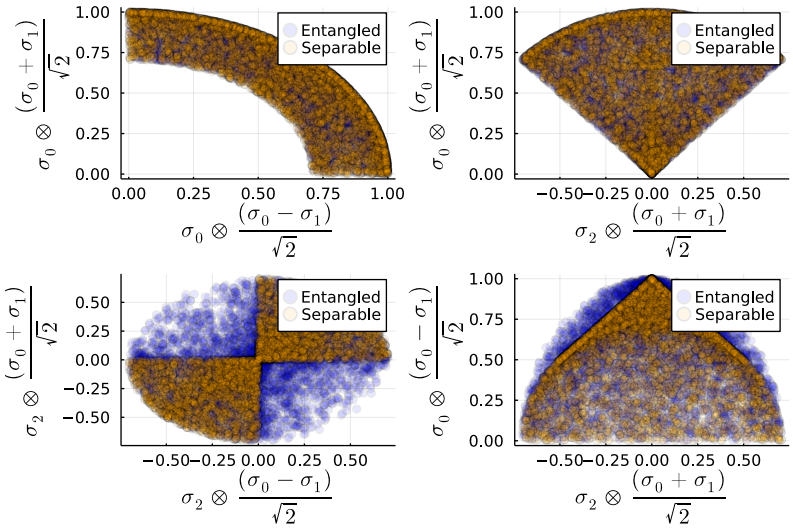

Figure 3.12: Data visualization for the set of operators (3.6). Some of the graphs show exclusion zones, while in others, separable and entangled states seem to overlap.

In this 3D plot, a surface of separable states is formed. No separable states reside outside of this surface. For this reason, the task of classifying a state as separable or entangled amounts to determining if the state is in the surface of separable states or not. This is a tractable task for a machine learning algorithm, and leads to an algorithm for determining the separability of a state using only four expected values, as opposed to the 15 needed for the PPT criterion.

An entanglement witness would require only 1 expected value, but would not work with every state sampled from the Haar induced distribution. In contrast, this method appears to work for any state sampled from this distribution.

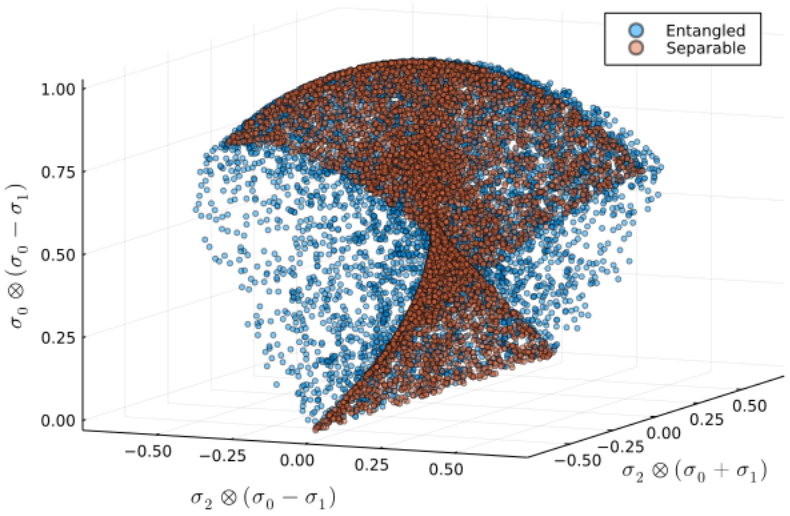

Figure 3.13: 3D visualization for the first three operators in equation (3.6). A sharp separation in entangled and separable pure states can be seen, there is a surface where only separable states are found. No separable states reside outside this surface.

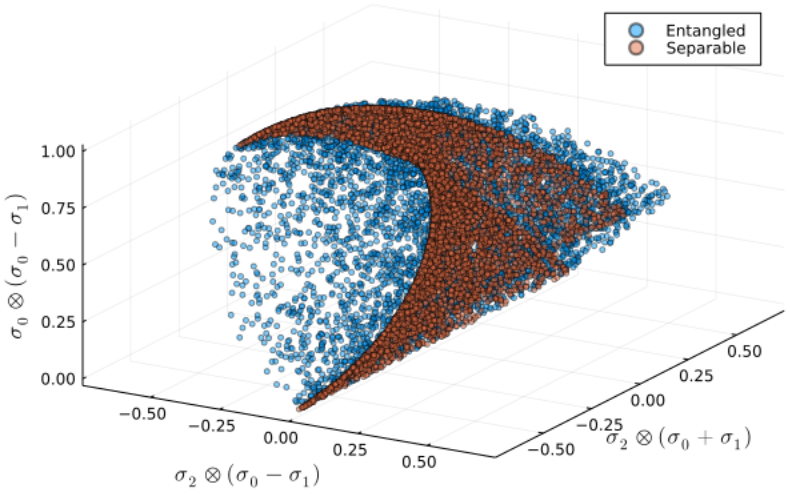

Figure 3.14: Another angle of figure 3.13. Separable states form a surface where no entangled states are found.

4 Implementation

The sets (2.1) were built by sampling 2 qubit states from the distributions using the Julia programming language [36]. The random operators used for the construction of the feature vector are sampled uniformly according to the Haar measure using the Julia package `QuantumInformation.jl` [37]. The main functions can be found in appendix A.

4.1 Main functions

Random unitary operators are sampled with the following parametrization of $U(2)$.

$$U(\alpha,\phi,\psi,\chi) := e^{i\alpha}\begin{pmatrix} e^{i\psi}\cos\phi & e^{i\chi}\mathrm{sen}\phi \\ -e^{-i\chi}\mathrm{sen}\phi & e^{-i\psi}\cos\phi \end{pmatrix}. \tag{4.1}$$

One samples $\alpha,\psi,\chi \in [0,2\pi)$ y $\xi \in [0,1)$ uniformly and then computes $\phi = \arcsin\sqrt{\chi}$ and evaluates (4.1).

As discussed in the previous section, one can then sample random separable states as in equation (3.1). Then pure random states are sampled by first sampling a random number between 1 and 4. This number determines how many random pure separable states are sampled and then the resulting random state is the normalized superposition of these states. Mixed states are sampled in the two ways described in the previous section (see appendix A for the code).

After sampling random states, the feature vectors must be built. This is done for a given state, ρ, by calculating all expected values from the operators O_{ij} or N_{ij} and forming the array

$$\begin{bmatrix} \langle\rho\rangle_{O_{01}} \\ \langle\rho\rangle_{O_{02}} \\ \vdots \\ \langle\rho\rangle_{O_{33}}. \end{bmatrix} \tag{4.2}$$

The operators are changed as needed for each model.

The labels for each state are computed using the PPT criterion.

With these ingredients, the data sets are built using a function that takes the following inputs,

```
function create_set_noW(N,representatives,mixed,Bures)
```

N is the number of separable states that the dataset will have (so the dataset has $2N$ states), `representatives` is the number of operators from the list are going to be used (15, 10 or 4 are the numbers used), `mixed` is a boolean input. If false, then pure random states are sampled, if true, then mixed random states from the Hilbert-Schmidt ensamble are sampeld. `Bures` is another boolean input. If true, mixed states from the Bures ensamble are sampled. States are sampled until N separable states occur. Then, the Y vector is built, which is the vector with the entanglement labels for each state. The function outputs X, the feature matrix, Y, the labels vector, and an array of the corresponding states.

In this work various ANN architectures are used, with varying depths (number of hidden layers) and number of hidden units. We also use input vectors of various dimensions

- 15 numbers of (2a).
- 10 numbers of the form (2b).
- 4 expected values using the operators in (3.6).

All the models are built and trained using the Julia package `Flux.jl` [38]. The hidden units have ReLU activation and the output layer is a single neuron with a sigmoid activation (see equations (2.6) and (2.7)). Training is performed using mini-batch gradient descent [27] with ADAM optimizer. For some models regularization is used and for others it is not in order to test the impact of regularization on the performance. The training is performed over different kinds of training sets and we compare the performance between them. A schematic visualization of these networks can be viewed in figure 4.1.

The number of layers of the models is defined as one of the following lines

```
n_layers = [400,160,80,40,20]; # L = 5
n_layers = [500,400,160,80,40,20]; # L = 6
```

These arrays indicate the number of hidden units in each layer. These two architectures were chosen after trying a wide variety of larger and smaller models. It was found that these number of layers and hidden units are enough to perform well and be trained with low computing time. The `Flux` library allows one to easily define a model and initialize its parameters randomly using the `Chain, Dense, Dropout` and `BatchNorm` functions. The `Chain(f,g)` function performs the composition $f \circ g$, the `Dense(n,m,act)` function creates a layer from n neurons to m neurons with activation function `act` and initializes the parameters randomly. The `Dropout(p)` function creates a dropout layer that removes neurons with probability p. The `BatchNorm` layer creates a batch normalization layer and initializes the parameters given in (2.15). The models for the largest network are defined as follows

```
#Normal model
model = Chain(
```

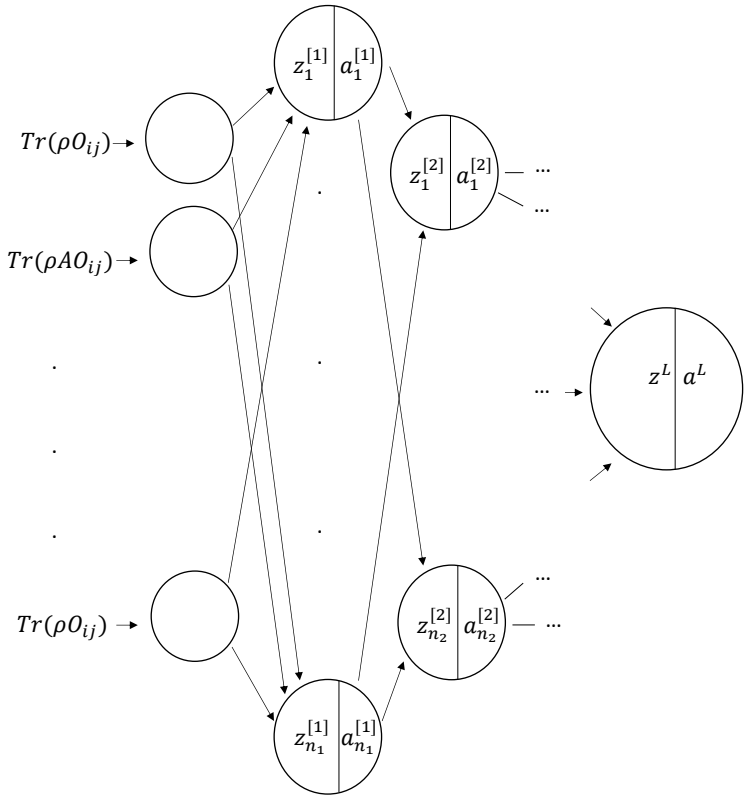

Figure 4.1: Neural network architecture. A variable number of hidden layers, L, is used with varying number of hidden units, n_l. The operators O_{ik} may refer to the products in (1a) or (1a).

```
    Dense(rep1,n_layers[1],relu),
    Dense(n_layers[1],n_layers[2],relu),
    Dense(n_layers[2],n_layers[3],relu),
    Dense(n_layers[3],n_layers[4],relu),
    Dense(n_layers[4],n_layers[5],relu),
    Dense(n_layers[5],n_layers[6],relu),
    Dense(n_layers[6],1,sigmoid)
)
#Regularized model
model2 = Chain(
    Dropout(0.1),
    Dense(rep1,n_layers[1],relu),
    BatchNorm(n_layers[1]),
    Dropout(0.1),
    Dense(n_layers[1],n_layers[2],relu),
    BatchNorm(n_layers[2]),
    Dropout(0.1),
    Dense(n_layers[2],n_layers[3],relu),
```

```
    BatchNorm(n_layers[3]),
    Dropout(0.1),
    Dense(n_layers[3],n_layers[4],relu),
    BatchNorm(n_layers[4]),
    Dropout(0.1),
    Dense(n_layers[4],n_layers[5],relu),
    BatchNorm(n_layers[5]),
    Dropout(0.1),
    Dense(n_layers[5],n_layers[6],relu),
    BatchNorm(n_layers[6]),
    Dense(n_layers[6],1,sigmoid)
)
```

Similar definitions are used for the smaller models by removing the last `Dense`, `Dropout` and `BatchNorm` layers as needed. The model is then trained using the built in function `train!(J,params,D-train,opt)` that performs one step of the `opt` variation of gradient descent (in this work, the ADAM is used) by optimizing the cost function J defined on the test set, `D-train`, and changes the model parameters, `params`. One step of training is defined in the function

```
function training_step!(model,loss,opt,X_train,Y_train)
    Flux.Optimise.train!(loss,Flux.params(model),[(X_train,Y_train)],opt)
end;
```

So, full training in many epochs with mini batch ADAM optimization (and regularization, when the model has the regularization layers) is defined in the following function

```
#=X_train and Y_train must be arrays of mini-batches,
while X_complete and Y_complete is the full data set. =#
function full_training!(model,loss,X_train,Y_train,X_test,Y_test,
X_complete,Y_complete,epochs,interval=500,α₀=0.01,decay_rate=0,α_decay_t=epochs)
    α_N = α₀
    opt=ADAM(α₀)
    L=length(X_train)
    costs=Vector{Real}(undef,epochs)
    @showprogress for i in 1:epochs
        trainmode!(model,true)
        k=rand(1:L)
        training_step!(model,loss,opt,X_train[k],Y_train[k])
        if i%α_decay_t == 0
            α_N = α_N/(1+decay_rate*floor(i/α_decay_t))
            opt=ADAM(α_N)
        end
        testmode!(model,true)
        l = loss(X_complete,Y_complete)
        costs[i] = l
        if i%interval == 0
            println("The cost function is: \$l")
        end
    end

    testmode!(model,true)
    println("Accuracy on the training set is:
    \$(100*accuracy(model,X_complete,Y_complete))")
    println("Accuracy on the test set is:
    \$(100*accuracy(model,X_test,Y_test))")
```

```
    return costs
end;
```

Notice that the cost function that is computed at each step is the true cost function evaluated over all training examples, while the optimization step is taking evaluating the cost function on a mini-batch.

Auxiliary functions and the functions that implement performance metrics can be consulted in appendix A.

4.2 Training and test sets

As mentioned, the labeled sets are constructed by sampling random states from the three mentioned distributions and classifying them according to the PPT criterion (1.20). All of these distributions produce far more entangled states than separable states. If we use a training set constructed by direct sampling, we end up with more entangled states than separable states, and after training we see that the model tends to always predict that a state is entangled, so we must perform sampling until we achieve a 50-50 ratio, such that no bias is introduced with the data. Models were trained and tested using training sets of different dimensions, which are denoted by M and m respectively. This results in various train and test sets according to each distribution, dimension of the input, type of the input (equation (2a) or (2b) and whether it is for training or testing, which may be labelled as $\mathcal{D}^{(M,N)}_{d_{train/test}}$, where M denotes the number of separable (entangled) states, N denotes the dimension of the feature vectors, which, as mentioned, may be 15, 10 or 4, and d denotes the distribution used for sampling. We use HS for the HS distribution, H for the Haar induced distribution and B for the Bures distribution. In order to perform a better comparison between the models, the states used for testing are all the same; the only thing that changes between the test sets are the operators used to calculate the expected values.

5 Results

In this chapter the main results are presented. They are divided by the dimension of the input vector. For each type of input vector two models were trained, one without regularization and one with regularization. At the end of this section, the performance of the models trained on mixed sets is presented.

5.1 Models trained with the 15 O_{ij} inputs

The training set for the first two models was $D^{(9984,15)}_{H_{train}}$ and the architecture used was $n_{layers} = [500, 400, 160, 80, 40, 20]$. These models will be referred to as $\mathcal{M}^{nr}_{15}$ and $\mathcal{M}^{r}_{15}$ (for non regularized and regularized respectively). The evolution of the cost with each step can be seen in figure 5.1. As expected, using the full information of the density matrix means that the model can drive the error down to almost zero. The performance was tested on three different sets, namely $D^{(80000,15)}_{H_{test}}$, $D^{(80000,15)}_{HS_{test}}$ and $D^{(80000,15)}_{B_{test}}$. The confusion matrices that arise for these models and their one number evaluations are shown in tables 5.1-5.4.

Regularization	No				Yes		
Metric / Distribution	$\mathcal{P}$	$\mathcal{R}$	F_1		$\mathcal{P}$	$\mathcal{R}$	F_1
H	0.95	0.99	0.97		0.79	0.99	0.88
HS	0.93	0.04	0.08		0.89	0.40	0.56
B	0.93	0.07	0.14		0.88	0.56	0.69

Table 5.1: Performance metrics of the first models, $\mathcal{M}^{nr}_{15}$ and $\mathcal{M}^{r}_{15}$, both trained with the set $D^{(9984,15)}_{H_{train}}$. The model without regularization does not generalize to mixed states, while the regularized model does better on the sets $D^{(80000,15)}_{HS_{test}}$ and $D^{(80000,15)}_{B_{test}}$.

These models perform well for states from the H distribution. It is interesting to note that regularization increases performance on mixed states, specially for those form the B distribution. Also, regularization helps with recall at the expense of precision. Now the question becomes, on which kinds of states does the model make mistakes?

	Separable	Entangled
Separable	79563	437
Entangled	4102	75898

Separable	Entangled
79989	11
21671	58329

Table 5.2: Confusion matrix for the test set $D^{(80000,15)}_{H_{test}}$ of the not regularized (left) and regularized model (right). Regularization helps the model to not incorrectly classify separable states at the cost of detecting less entangled states.

	Separable	Entangled
Separable	3548	76452
Entangled	268	79732

Separable	Entangled
32392	47608
4045	75955

Table 5.3: Confusion matrix for the test set $D^{(80000,15)}_{HS_{test}}$ of the not regularized (left) and regularized model (right). Regularization helps the model to classify mixed states, even when training was performed on pure states.

	Separable	Entangled
Separable	5941	74059
Entangled	454	79546

Separable	Entangled
45103	34897
6418	73582

Table 5.4: Confusion matrix for the test set $D^{(80000,15)}_{B_{test}}$ of the not regularized (left) and regularized model (right). States from the Bures distribution are classified better because more of them are strongly entangled.

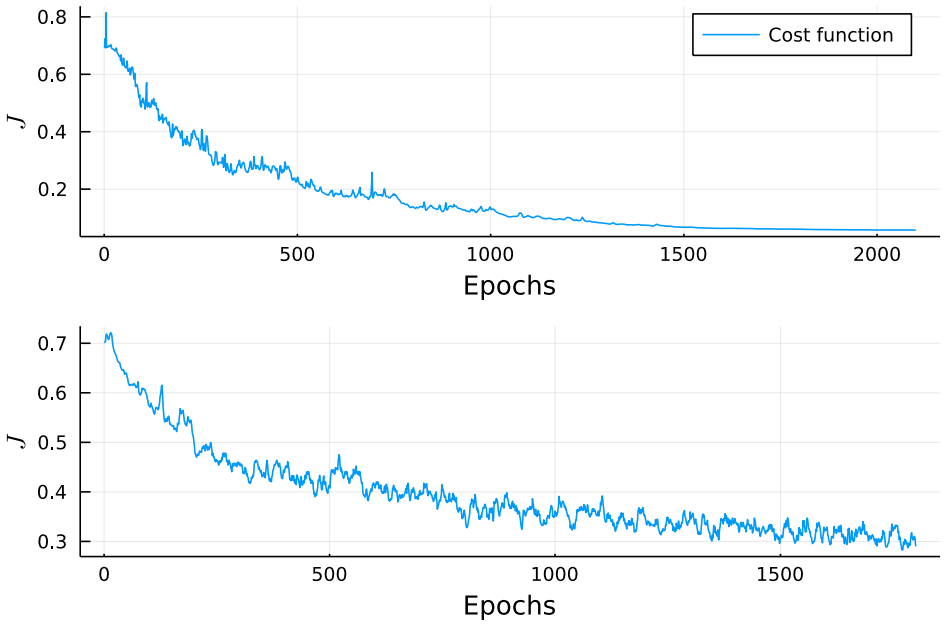

Figure 5.1: Evolution of true cost with each training step. The unregularized model is top and the regularized one bottom. We can see that the cost function eventually goes very close to zero for the unregularized model.

5.1.1 Error analysis of $\mathcal{M}_{15}^{r}$ and $\mathcal{M}_{15}^{nr}$

We can read from table 5.2 that both models incorrectly classify some entangled states (sampled from H) as separable states. It turns out that these states have low values of concurrence. Indeed, the maximum concurrence of the mislabeled entangled states is 0.17 for $\mathcal{M}_{15}^{nr}$ and 0.39 for $\mathcal{M}_{15}^{r}$. A histogram of the concurrence of mislabeled states can be seen in figure 5.2, where it is seen that all entangled states with high concurrence are correctly labeled.

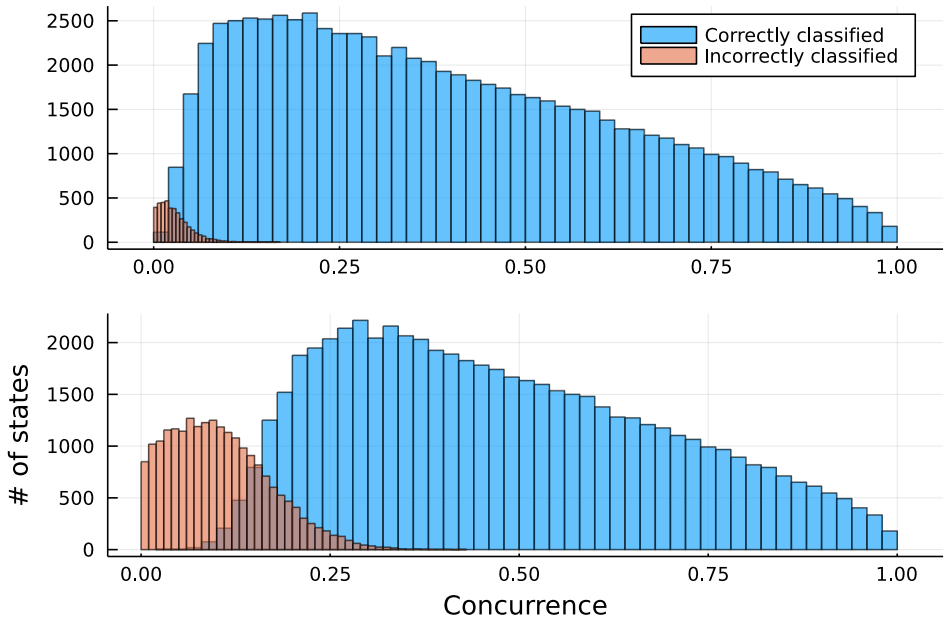

Figure 5.2: Histogram of the predictions for entangled states. Top: $\mathcal{M}_{15}^{nr}$ model. Bottom: $\mathcal{M}_{15}^{r}$ model. All highly entangled states are correctly classified.

The model $\mathcal{M}_{15}^{nr}$ mislabels many separable states sampled from B, as can be seen in table 5.4. These states have lower average purity than the correctly labeled ones. The maximum purity of mislabeled states is .73, while the average purity is 0.43. For the correctly labeled, the average purity is 0.49. This can be seen in figure 5.3. It is known that ML models struggle to generalize to sets that are not drawn from the same distribution as the test set. That is what is seen in this case, but if one restricts the classification to highly entangled, high purity states, the performance on mixed states is adequate, even when training is done only on pure states.

To see this, the performance of the model is studied in intervals of both purity and concurrence. The results are presented in tables 5.5-5.6, where the accuracy (2.19) is presented in intervals of purity and concurrence. This tables are studied for the test set $D_{B_{test}}^{(80000,15)}$. It is worth noting that in table 5.6, the model $\mathcal{M}_{15}^{nr}$ has close to 100% accuracy in most concurrence intervals. This is because the model tends to classify all mixed states as entangled, as may be seen in the confusion matrix 5.4, so this result should not be confused as positive.

Finally, consider the sets $M_1 = \{\rho \in D_{B_{test}}^{(80000,15)} | P(\rho) < 0.7\}$ and $M_2 = \{\rho \in$

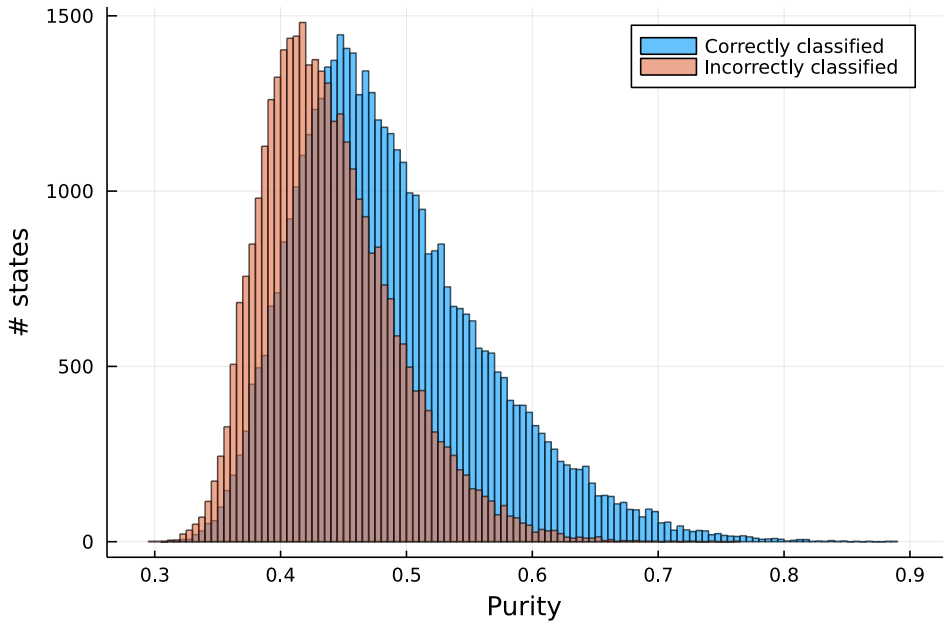

Figure 5.3: Purity of separable states drawn form $D_{B_{test}}^{(80000,15)}$. Most incorrectly labeled states have low purity.

Purity Interval	# of states	Accuracy (%) for $\mathcal{M}_{15}^{nr}$	Accuracy (%) for $\mathcal{M}_{15}^{r}$
0.3-0.4	13134	7.00	34.77
0.4-0.5	65894	33.90	63.99
0.5-0.6	49944	69.05	86.42
0.6-0.7	22094	86.48	92.81
0.7-0.8	7440	96.13	92.85
0.8-0.9	1416	99.44	91.88
0.9-1	77	100	94.81

Table 5.5: Accuracy in intervals of purity for the set $D_{B_{test}}^{(80000,15)}$. The fraction of correctly classified states increases with the purity of the states.

$D_{B_{test}}^{(80000,15)}|\rho$ is entangled and $C(\rho) < 0.6\}\}$. Then, for the set $D_{B_{test}}^{(80000,15)}\backslash(M_1 \cup M_2)$, that is, one filters all states from $D_{B_{test}}^{(80000,15)}$ that have purity larger than 0.7 and concurrence larger than 0.6 (but also keeping the separable states), one ends up with 2245 states and the model $\mathcal{M}_{15}^{r}$ has 99.73% accuracy. This means that this is an acceptable model to classify mixed state entanglement when one knows that the entangled states have high purity and one only cares about highly entangled states. On the other hand, $\mathcal{M}_{15}^{nr}$ achieves 87.97% accuracy.

Concurrence Interval	# of states	Accuracy (%) for $\mathcal{M}_{15}^{nr}$	Accuracy (%) for $\mathcal{M}_{15}^{r}$
0.0-0.1	94512	21.29	58.35
0.1-0.2	18843	99.39	90.70
0.2-0.3	17757	99.97	98.91
0.3-0.4	13662	100	99.97
0.4-0.5	8599	100	100
0.5-0.6	4435	100	100
0.6-0.7	1687	100	100
0.7-0.8	440	100	100
0.8-0.9	64	100	100

Table 5.6: Accuracy in intervals of concurrence for the set $D_{B_{test}}^{(80000,15)}$. The model $\mathcal{M}_{15}^{nr}$ tend to classify most states as entangled, while the regularized model can make more of a distinction.

5.2 Models trained with 10 N_{ij} inputs

The training set for the next two models is $D_{H_{train}}^{(12800,10)}$ and the architecture used was $n_{layers} = [400, 160, 80, 40, 20]$. These models will be referred to as $\mathcal{M}_{10}^{nr}$ and $\mathcal{M}_{10}^{r}$. The evolution of the cost with each step can be seen in figure 5.4. The performance was tested on the sets $D_{H_{test}}^{(80000,10)}, D_{HS_{test}}^{(80000,10)}$ and $D_{B_{test}}^{(80000,10)}$. The confusion matrices that arise for these models and their one number evaluations are shown in tables 5.7-5.10. It is found that the F_1 score measured on states from the H distribution in not as good as in last section. It seems like the missing information plays a role in classifying pure states that the operators constructed from (3.5) cannot make up for.

Regularization	No				Yes		
Metric / Distribution	$\mathcal{P}$	$\mathcal{R}$	F_1		$\mathcal{P}$	$\mathcal{R}$	F_1
H	0.77	0.95	0.85		0.67	0.99	0.80
HS	0.71	0.27	0.39		0.68	0.38	0.49
B	0.72	0.35	0.47		0.69	0.48	0.57

Table 5.7: Performance metrics of $\mathcal{M}_{10}^{nr}$ and $\mathcal{M}_{10}^{r}$. Neither of the models achieve an F_1 score as good as in the case of 15 inputs. Also, the models do not generalize well to $D_{HS_{test}}^{(80000,10)}$ and $D_{B_{test}}^{(80000,10)}$, but regularization does help.

5.2.1 Error analysis of $\mathcal{M}_{10}^{r}$ and $\mathcal{M}_{10}^{nr}$

The same method is applied to these models as in last section. Again, most incorrectly classified entangled states from H have low concurrence, as can be seen in 5.5. This time, though, there are some highly entangled states that are not correctly classified.

	Separable	Entangled
Separable	76220	3780
Entangled	22989	57011

Separable	Entangled
79144	856
37964	42036

Table 5.8: Confusion matrix for the test set $D^{(80000,10)}_{H_{test}}$ of the not regularized (left) and regularized model (right). As before, regularization makes the model make fewer mistakes in labeling separable states.

	Separable	Entangled
Separable	21755	58245
Entangled	8768	71232

Separable	Entangled
30473	49527
14128	65872

Table 5.9: Confusion matrix for the test set $D^{(80000,10)}_{HS_{test}}$ of the not regularized (left) and regularized model (right). This models do not generalize as well as those from last section.

	Separable	Entangled
Separable	28205	51795
Entangled	10456	69544

Separable	Entangled
38122	41878
17142	62858

Table 5.10: Confusion matrix for the test set $D^{(80000,10)}_{B_{test}}$ of the not regularized (left) and regularized model (right).

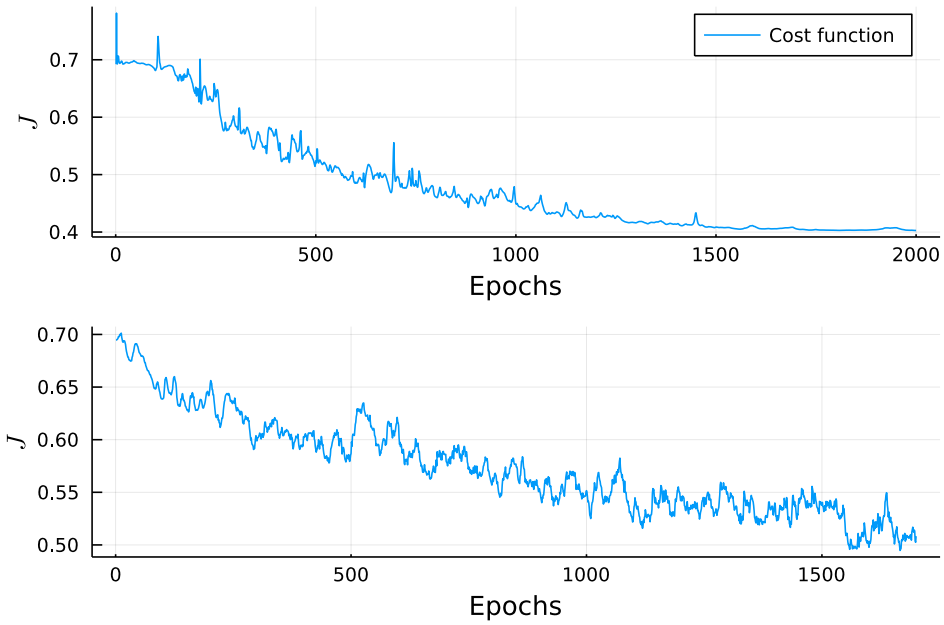

Figure 5.4: Evolution of true cost with each training step. These models do not approach 0 cost. Missing information prevents this.

The maximum concurrence of mislabeled states (from H) is 0.93 for $\mathcal{M}^r_{10}$ and 0.94 for $\mathcal{M}^r_{10}$.

Meanwhile, as can be seen from figure 5.6, most incorrectly labeled separable states

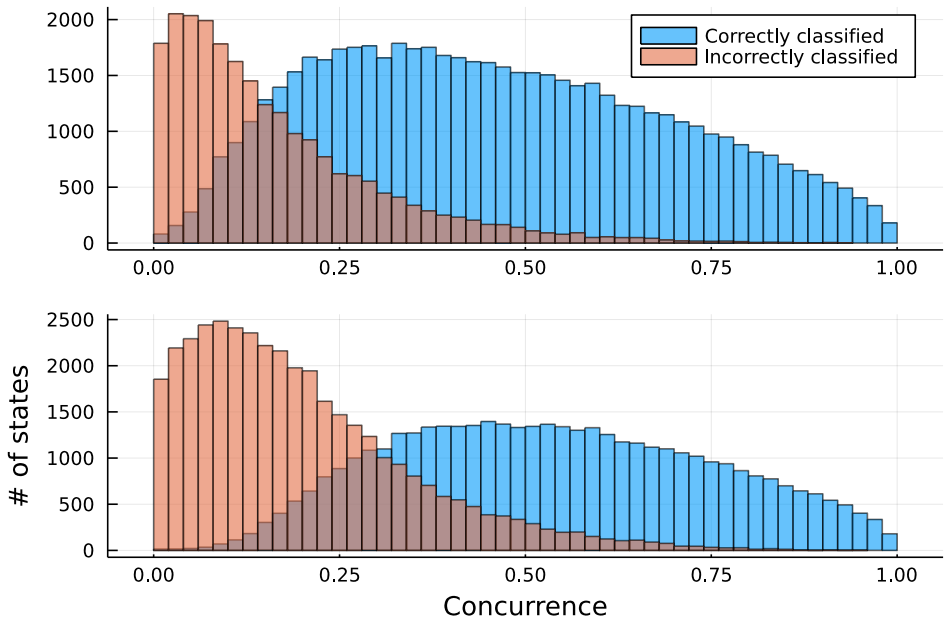

Figure 5.5: Histogram of the predictions for entangled states. Top: $\mathcal{M}_{10}^{nr}$ model. Bottom: $\mathcal{M}_{10}^{r}$ model. This time there are some highly entangled states that are mislabeled.

from B have low purity, but there are some high purity states that are incorrectly labeled.

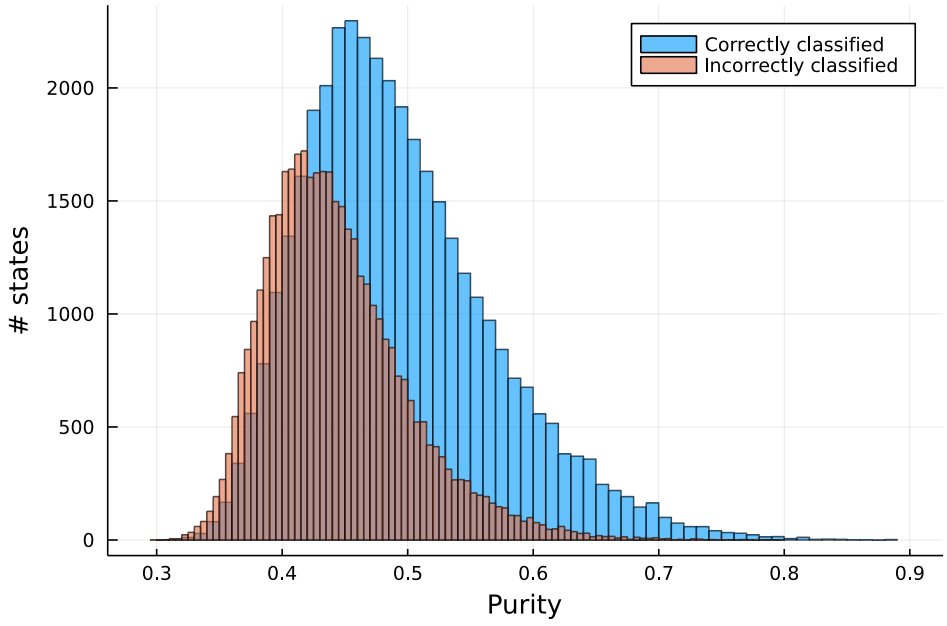

Figure 5.6: Purity of separable states drawn form $D_{B_{test}}^{(80000,10)}$. Most incorrectly labeled states have low purity, as before.

The interval performance is seen in tables 5.11 and 5.12, where we see a similar

behavior as in last section, but less pronounced.

Purity Interval	# of states	Accuracy (%) for $\mathcal{M}_{10}^{nr}$	Accuracy (%) for $\mathcal{M}_{10}^{r}$
0.3-0.4	13134	20.71	26.96
0.4-0.5	65894	47.64	54.25
0.5-0.6	49944	74.92	75.16
0.6-0.7	22094	83.72	78.03
0.7-0.8	7440	86.65	77.75
0.8-0.9	1416	85.24	75.85
0.9-1	77	90.91	71.43

Table 5.11: Accuracy in intervals of purity for the set $D_{B_{test}}^{(80000,10)}$. The fraction of correctly classified states increases with the purity of the states.

Concurrence Interval	# of states	Accuracy (%) for $\mathcal{M}_{10}^{nr}$	Accuracy (%) for $\mathcal{M}_{10}^{r}$
0.0-0.1	94512	40.84	49.27
0.1-0.2	18843	81.19	69.34
0.2-0.3	17757	90.00	81.69
0.3-0.4	13662	94.93	89.78
0.4-0.5	8599	97.18	94.29
0.5-0.6	4435	98.46	97.23
0.6-0.7	1687	98.93	98.10
0.7-0.8	440	100	99.54
0.8-0.9	64	100	100

Table 5.12: Accuracy in intervals of concurrence for the set $D_{B_{test}}^{(80000,10)}$. The higher the concurrence the better the accuracy.

Finally, consider the same filtered set as in last section (the test set minus low purity and low concurrence entangled states). Since the states are the same, 2245 such states are again found, and the accuracy of $\mathcal{M}_{10}^{r}$ on this set is 97.15%. This means this is still a valid model for high purity states and high concurrence states. It is also interesting to note that the model $\mathcal{M}_{10}^{nr}$ has an accuracy of 94.92% on this filtered set.

5.3 Models trained with the 4 special N_{ij} inputs

The training set for the next two models is $D_{H_{train}}^{(12800,4)}$ and the architecture used was $n_{layers} = [400, 160, 80, 40, 20]$. These models will be referred to as $\mathcal{M}_{4}^{nr}$ and $\mathcal{M}_{4}^{r}$. The evolution of the cost with each step can be seen in figure 5.7. The performance was tested on the sets $D_{H_{test}}^{(80000,4)}$, $D_{HS_{test}}^{(80000,4)}$ and $D_{B_{test}}^{(80000,4)}$. The confusion matrices that arise

for these models and their one number evaluations are shown in tables 5.13-5.16. It is interesting to note in table 5.13 that the performance on states drawn from H is very good, even if there is very little information about the state available. This was to be expected, as a clear distinction is seen in figures 3.13 and 3.14.

Regularization	No				Yes		
Metric / Distribution	$\mathcal{P}$	$\mathcal{R}$	F_1		$\mathcal{P}$	$\mathcal{R}$	F_1
H	0.85	0.97	0.91		0.63	0.99	0.78
HS	0.61	0.08	0.14		0.59	0.53	0.56
B	0.65	0.11	0.17		0.60	0.56	0.58

Table 5.13: Performance metrics of the first models, $\mathcal{M}_4^{nr}$ and $\mathcal{M}_4^{r}$. Performance for the H set is very good for both models.

	Separable	Entangled
Separable	77532	2468
Entangled	13336	66664

Separable	Entangled
79867	133
46161	33839

Table 5.14: Confusion matrix for the test set $D_{H_{test}}^{(80000,4)}$ of the not regularized (left) and regularized model (right). Good performance is found in both models, but the regularized one misses on a lot of entangled states.

	Separable	Entangled
Separable	6134	73866
Entangled	3856	76144

Separable	Entangled
42553	37447
29695	50305

Table 5.15: Confusion matrix for the test set $D_{HS_{test}}^{(80000,4)}$ of the not regularized (left) and regularized model (right). We can see a trend, models trained with pure states and without regularization tend to classify most mixed states as entangled.

	Separable	Entangled
Separable	7478	72522
Entangled	4176	75824

Separable	Entangled
44846	35154
29650	50350

Table 5.16: Confusion matrix for the test set $D_{B_{test}}^{(80000,4)}$ of the not regularized (left) and regularized model (right).

5.3.1 Error analysis of $\mathcal{M}_4^r$ and $\mathcal{M}_4^{nr}$

Focusing on mislabeled entangled states from H, it can be seen in figure 5.8 that most highly entangled states are correctly classified and most incorrectly classified states

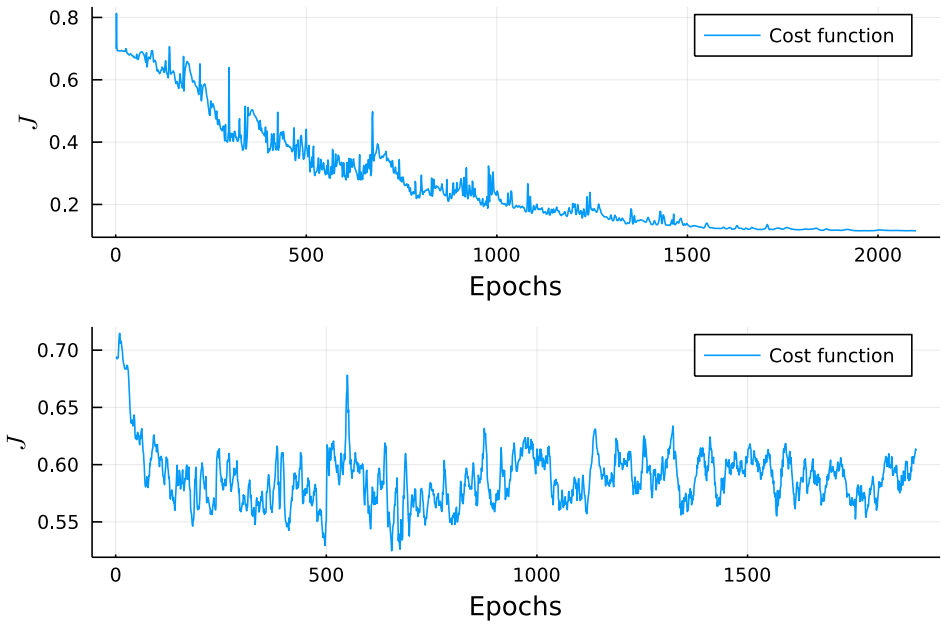

Figure 5.7: Evolution of true cost with each training step. The unregularized model is top and the regularized one bottom. Notice how the final cost function is below that of the 10 input model.

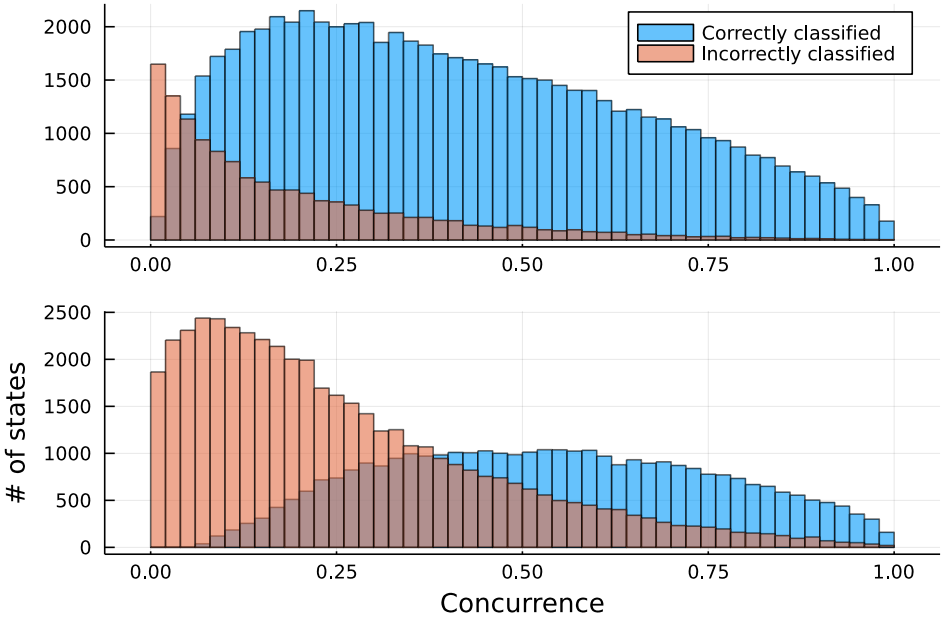

Figure 5.8: Histogram of the predictions for entangled states. Top: $\mathcal{M}_4^{nr}$ model. Bottom: $\mathcal{M}_4^{r}$ model.

have little entanglement, but again there are a few highly entangled states that are mislabeled in both models.

Also, in the case of separable states from B, it is seen in figure 5.9 that purity

does not play such an important role in mislabeling from model $\mathcal{M}_4^r$. Both histograms are very alike, only slightly shifted. Indeed, table 5.17 shows that accuracy does not dramatically increase with purity.

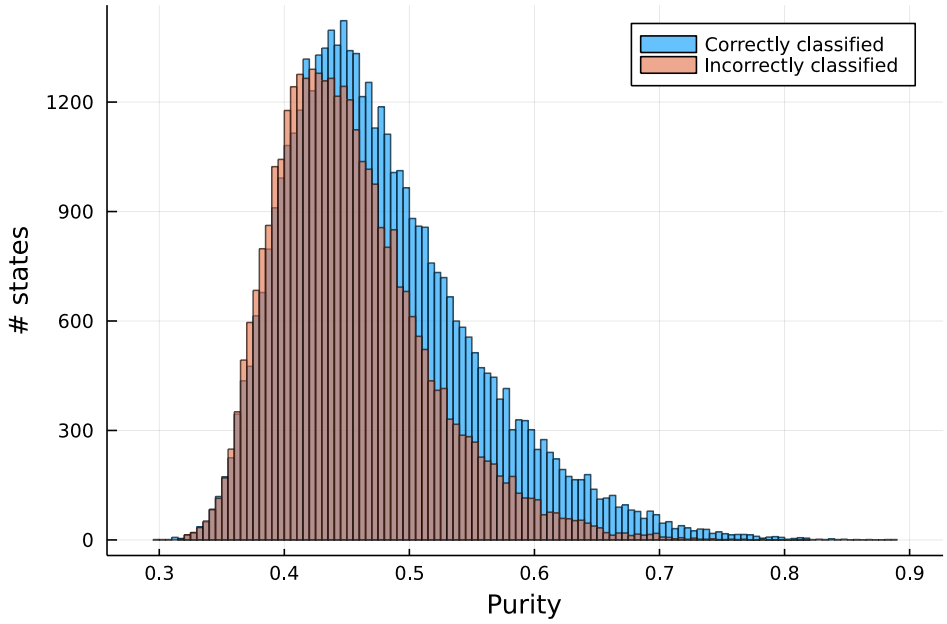

Figure 5.9: Purity of separable states drawn form $D_{B_{test}}^{(80000,4)}$. This time both histograms are very alike.

To see this, the performance of the model is studied in intervals of both purity and concurrence.

Purity Interval	# of states	Accuracy (%) for $\mathcal{M}_4^{nr}$	Accuracy (%) for $\mathcal{M}_4^r$
0.3-0.4	13134	10.14	48.24
0.4-0.5	65894	34.12	55.51
0.5-0.6	49944	66.83	64.09
0.6-0.7	22094	81.68	65.58
0.7-0.8	7440	89.78	64.73
0.8-0.9	1416	92.37	64.48
0.9-1	77	94.81	59.74

Table 5.17: Accuracy in intervals of purity for the set $D_{B_{test}}^{(80000,4)}$. The fraction of correctly classified states does not increase much for the regularized model, the unregularized one does better.

This makes it so that the performance on the filtered set (which has 2245 elements) has an accuracy of only 82.36% for $\mathcal{M}_4^r$ and 84.32% for $\mathcal{M}_4^{nr}$. It seems this model does not benefit as much from regularization.

Concurrence Interval	# of states	Accuracy (%) for $\mathcal{M}_4^{nr}$	Accuracy (%) for $\mathcal{M}_4^{r}$
0.0-0.1	94512	21.98	54.92
0.1-0.2	18843	93.83	55.60
0.2-0.3	17757	95.37	64.41
0.3-0.4	13662	96.12	70.58
0.4-0.5	8599	96.87	74.90
0.5-0.6	4435	97.27	78.85
0.6-0.7	1687	97.21	80.38
0.7-0.8	440	98.63	85.91
0.8-0.9	64	98.87	81.25

Table 5.18: Accuracy in intervals of concurrence for the set $D_{B_{test}}^{(80000,4)}$. In this case the regularized model does not do as well with these states, while the unregularized one tends to classify most states as entangled.

To get a sense of the shortcoming for mixed states in this models, one can look at figure 5.10. Mixed separable states are spread out in all space, so a model that correctly classifies pure separable states and pure entangled states will be confused by the mixed separable states.

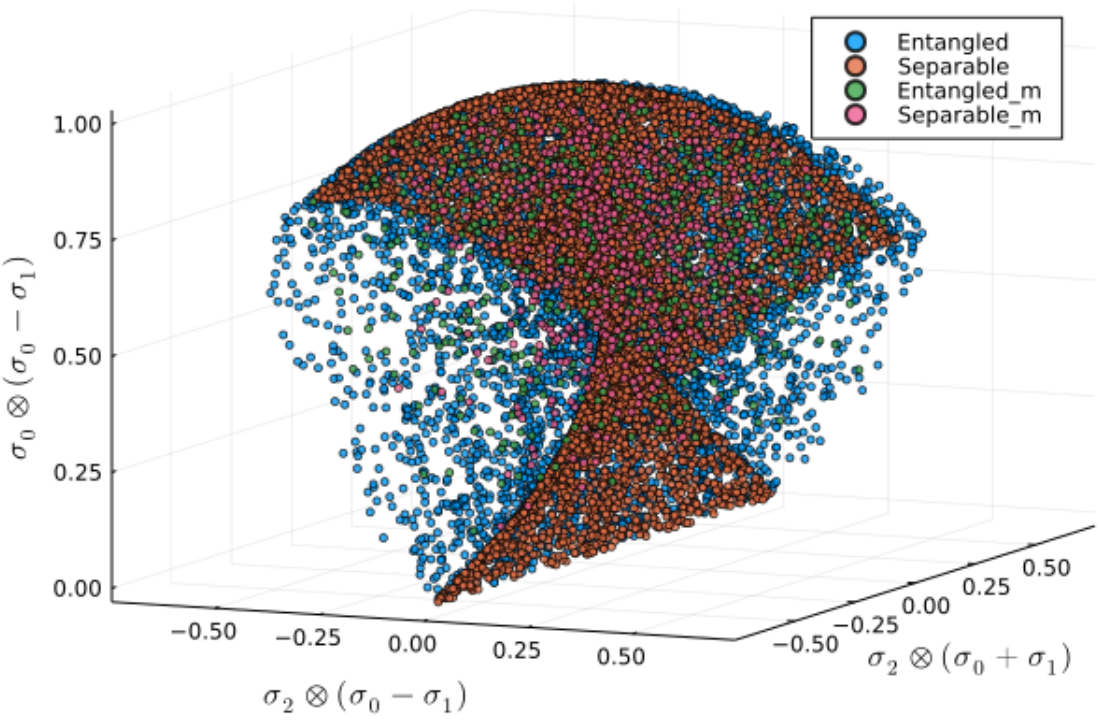

Figure 5.10: Pure and mixed separable states. A model trained to classify the orange states as separable will struggle to classify the pink ones as separable too, since they are outside the orange separable surface.

5.4 Training with mixed states

The models from past sections were trained using pure states because, for higher dimensions, it becomes very computationally demanding to sample and classify mixed states. In the case of two qubits, we have the PPT criterion, so training with mixed states is possible and efficient. If training is performed with $D^{(N,4)}_{B_{train}}$ for a model with 4 inputs, the performance increases with mixed states (as is to be expected) but is severely impacted for pure states. The metrics for this models, which we may call $\mathcal{M}^{nr/r}_{m4}$, are shown in table 5.19

Regularization	No				Yes		
Metric / Distribution	$\mathcal{P}$	$\mathcal{R}$	F_1		$\mathcal{P}$	$\mathcal{R}$	F_1
H	0.60	0.72	0.65		0.59	0.81	0.68
HS	0.57	0.75	0.65		0.57	0.77	0.65
B	0.59	0.73	0.65		0.58	0.76	0.66

Table 5.19: Performance metrics for models trained with mixed states. The F_1 across all distributions and both models becomes almost the same.

For both these models there are a lot of separable states from H that are classified as entangled. This is a problem because there is no way to characterize the states that are incorrectly classified as was done in past sections. These states are pure and separable, that is they have $P = 1$ and $C = 0$, so this is not a useful model in an experimental setup that produces pure states. On the other hand, there are many mixed states that are incorrectly classified, but when characterized as in past sections, it is found that there are many highly entangled states and high purity states that are not correctly classified. In fact, the histograms of purity of correctly and incorrectly classified states overlap, as can be seen in figure 5.11.

The same things occur for models with 10 inputs.

5.5 Testing with states sampled directly from Bures distribution

So far, the test phase has been performed with sets that have a 50-50 ratio of separable and entangled states, since that is how training was performed. The question arises then, how does a model trained like this perform with states sampled directly from the Bures distribution? When states are sampled like this, only around 7.8% of them are separable. The test set of states sampled directly from the Bures distributions will be referred to as $D^{(100000,4)}_{Bures}$, and was built using 100000 states in total, of which 7316 are separable states. Model $\mathcal{M}^r_4$ has an acuracy of 74.97% for this set. The confusion matrix is

After performing the same filtering process with this test set, it is found that 1984 states remain and accuracy rises to 83% for $\mathcal{M}^r_4$ and to 96.32% for $\mathcal{M}^{nr}_4$. So results

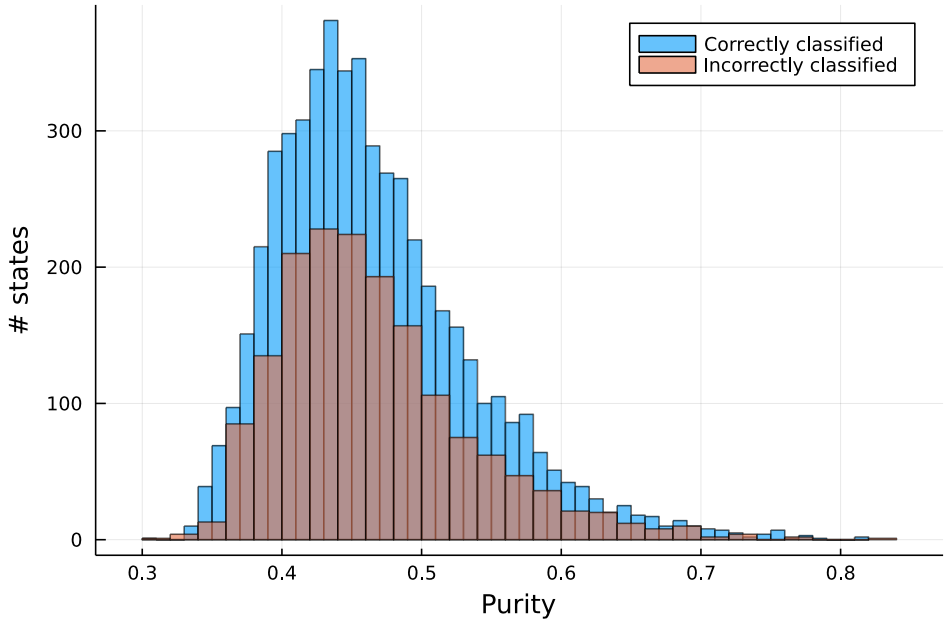

Figure 5.11: Purity of separable states drawn form $D^{(7000)}_{B_{test}}$. As is to be expected, both histograms overlap, the only difference being in the number of counts and not in the distribution. This histogram was done with predictions from $\mathcal{M}^r_{m4}$.

	Separable	Entangled
Separable	3944	3372
Entangled	37112	55572

Table 5.20: Confusion matrix for the test set $D^{(100000,4)}_{Bures}$ for the regularized model. Most states are entangled.

remain similar even when the test set does not have a 50-50 ratio.

6 Conclusions

Our results indicate that a universal classifier is not possible using this method. The main source of error comes form the similarity of weakly entangled states and separable states. This is to be expected, since weakly entangled states are the closest to the set of separable states and machine learning models usually struggle near the borders of different classes.

Other works that have used ML have encountered similar problems. For instance, reference [21] uses a similar approach, but training is performed on mixed states sampled from the Hilbert-Schmidt distribution and tests within the same distribution. The best accuracy reported is for a restricted subset from this distribution, focusing on highly entangled states or highly separable states (as measured by the value of the lowest eigenvalue of the partial transpose of a density matrix). Also, the performance for pure states is not reported. In our work we found that, if training is performed with mixed states, the performance for pure states is rather poor. Reference [22] reports good results on mixed states and has the advantage of only sampling pure states while performing well for mixed states, but full information of the quantum state is required in the algorithm proposed in that work. Finally, reference [39] also trains the model with mixed states and tests within those states; this makes scaling the method difficult.

Still, the neural network approach to the separability problem for two qubits shows promise. Classification of pure states may be achieved using as little as four observables, and the method has proven to be useful in classifying high purity and high concurrence mixed states. This is encouraging, since one is usually interested in these kinds of quantum states for quantum computation tasks.

The task of very accurately classifying all mixed states remains unsolved. Some possible avenues to explore are performing dimensional reduction (using principal component analysis, for instance) when studying mixed states using many operators of the form (1) to find which operators are the most important to classify entanglement. This could lead to a reliable method to classify low purity and low concurrence states using less than the fifteen operators needed for the PPT criterion.

Also, the method proposed in this thesis could be extended to higher dimensional Hilbert spaces, where the PPT criterion is no longer available, since the classification of pure states is known for many Hilbert spaces, and it was shown that training with only pure states yields fair results with high purity and highly entangled mixed states.

A Code

Sampling of random unitary matrices:

```
#= Parametrización de U(2) =#
function U(alpha,psi,chi,xi)
    phi = asin(xi)
    return exp(im*alpha) * [exp(im*psi)*cos(phi) exp(im*chi)sin(phi);
                      -exp(-im*chi)sin(phi) exp(-im*psi)cos(phi)]
end;
```

The functions used to sample random states are the following

```
#= Function to sample M 2 qubit separable states =#
function estados_separables(M::Integer)
    separables = Array{Array{Complex{Float64},1}}(undef,M)
    for i in 1:M
        α1=rand(0:0.01:2*pi);ψ1=rand(0:0.01:2*pi);χ1=rand(0:0.01:2*pi)
        α2=rand(0:0.01:2*pi);ψ2=rand(0:0.01:2*pi);χ2=rand(0:0.01:2*pi)

        ξ1=rand() ;ξ2=rand()

        separables[i] = kron(U(α1,ψ1,χ1,ξ1)*[1,0],U(α2,ψ2,χ2,ξ2)*[1,0])
    end

    return separables
end

#Function to sample pure random states
function estados_puros_aleatorios(M::Integer)
    aleatorios = Array{Array{Complex{Float64},1}}(undef,M)
    for i in 1:M
        n = rand(1:4)
        psis = estados_separables(n)
        pesos = normalize(rand(n))
        aleatorios[i] = normalize(sum(pesos.*psis))
    end
    return aleatorios
end

#= Function to sample random mixed states form
the HS distribution =#
function estados_aleatorios(M::Integer)
    g = HilbertSchmidtStates(4)
    aleatorios = Array{Array{Complex{Float64},2}}(undef,M)
    for i in 1:M
        aleatorios[i] = rand(g)
    end
    return aleatorios
end;
```

```
#= Function to sample random mixed states from
the Bures distribution =#
function estados_aleatorios_Bures(M::Integer)
    g = GinibreEnsemble(4)
    u = CircularEnsemble{2}(4)
    aleatorios = Array{Array{Complex{Float64},2}}(undef,M)
    for i in 1:M
        A = rand(g)
        U = rand(u)
        Id = Diagonal([1,1,1,1])
        matriz = (Id + U)*A*A' *(Id +U')
        aleatorios[i] = Hermitian(matriz / tr(matriz))
    end
    return aleatorios
end
```

The feature vectors are built with the function

```
function feature(state,representatives::Integer)
    f = Array{Float64}(undef,representatives)
    for i in 1:representatives
        f[i] = expected_value(operators[i],state)
    end
    return normalize(f)
end
```

Here, the function `expected_value` (O, ρ) computes the expected value $\langle O \rangle_\rho$, and `operators` is an array that contains the operators to be used, that is, operators of the form (1a) or (1b), which is defined before the training of the ANN.

The PPt criterion is , implemented with the following functions

```
function PPT(ρ::Matrix)
    ρ^TB = partial_transpose(ρ)
    e_vals = eigvals(ρ^TB)
    positives = real.(e_vals) .>= -0.0000001
    #sum(positives) is the number of positive eigenvalues
    if sum(positives) != 4   #=If there is a negative
    eigenvalue => state entangled =#
         0 #entangled
    else
         1 #separable
    end
end

function PPT(ψ::Vector)
    ρ = ψ*ψ'
    PPT(ρ)
end;
```

The data sets are built with the following function

```
#= Function that creates the data sets (train or test) for
2 qubits. It consists of N entangled and N separable states.
If mixed!=true, Bures!=true, then random pure states from H
are used.If mixed=true, mixed random states from HS are used.
If Bures=true, mixed random states from B are used. It returns
X(matrix of feature vectors) and Y (labels) along with an
array of the states that are used.=#
```

```
function create_set_noW(N,representatives,mixed,Bures)
    if mixed == false
        states=[estados_puros_aleatorios(1)[1]]
        Y=[PPT(states[1])]
        #= A counter is set up to track how many separable states have been
        produced.
        Since all distributions produce many more entangled states, m
        any states have to be sampled in order to get a 50-50 ratio. =#.
        separable_counter=Y[]
        entangled_counter=1-separable_counter
        set = feature_noW(states[1],representatives)
        while separable_counter<N
        #Stop adding entangled states once you get to N.
            if entangled_counter<N
                state = estados_puros_aleatorios(1)[1]
                ppt_actual = PPT(state)
                push!(Y,ppt_actual)
                push!(states,state)
                separable_counter += ppt_actual
                entangled_counter += (1-ppt_actual)
                nuevo=feature_noW(state,representatives)
                set=hcat(set,nuevo)
            else
                state = estados_puros_aleatorios(1)[1]
                ppt_actual = PPT(state)
                if ppt_actual == 1
                    push!(Y,ppt_actual)
                    push!(states,state)
                    separable_counter += ppt_actual
                    nuevo=feature_noW(state,representatives)
                    set=hcat(set,nuevo)
                end
            end
        end
        return set,transpose(Y),states
    elseif (mixed == true && Bures == false)
        states=[estados_aleatorios(1)[1]]
        Y=[PPT(states[1])]
        separable_counter=Y[]
        entangled_counter=1-separable_counter
        set = feature_noW(states[1],representatives)
        while separable_counter<N
            if entangled_counter<N
                state = states_aleatorios(1)[1]
                ppt_actual = PPT(state)
                push!(Y,ppt_actual)
                push!(states,state)
                separable_counter += ppt_actual
                entangled_counter += (1-ppt_actual)
                nuevo=feature_noW(state,representatives)
                set=hcat(set,nuevo)
            else
                state = states_aleatorios(1)[1]
                ppt_actual = PPT(state)
                if ppt_actual == 1
                    push!(Y,ppt_actual)
                    push!(states,state)
                    separable_counter += ppt_actual
                    nuevo=feature_noW(state,representatives)
                    set=hcat(set,nuevo)
                end
            end
        end
        return set,transpose(Y),states
    elseif mixed == true && Bures == true
        states=[estados_aleatorios_Bures(1)[1]]
        Y=[PPT(states[1])]
        separable_counter=Y[]
```

```
        entangled_counter=1-separable_counter
        set = feature_noW(states[1],representatives)
        while separable_counter<N
            if entangled_counter<N
                state = states_aleatorios_Bures(1)[1]
                ppt_actual = PPT(state)
                push!(Y,ppt_actual)
                push!(states,state)
                separable_counter += ppt_actual
                entangled_counter += (1-ppt_actual)
                nuevo=feature_noW(state,representatives)
                set=hcat(set,nuevo)
            else
                state = estados_aleatorios_Bures(1)[1]
                ppt_actual = PPT(state)
                if ppt_actual == 1
                    push!(Y,ppt_actual)
                    push!(states,state)
                    separable_counter += ppt_actual
                    nuevo=feature_noW(state,representatives)
                    set=hcat(set,nuevo)
                end
            end
        end
        return set,transpose(Y),states
    end
end
```

Performance metrics are defined in the following functions

```
function accuracy(Modelo,X,Y)
    hits = round.(Modelo(X)) .== Y
    return sum(hits)/length(Y)
end;

#= Here "positive" means being separable
and "negative" means being entangled =#
function confusion_table(tp,fp,tn,fn)
    df = DataFrame(Separable = [tp,fp],Entangled = [fn,tn])
    return df
end

function confusion_matrix(labels,predictions)
    zeros = findall(x->x==0,labels')
    ones = findall(x->x==1,labels')
    ln=length(zeros) ; lp=length(ones)
    positives = predictions[ones] .== labels[ones]
    negatives = predictions[zeros] .== labels[zeros]
    tp = sum(positives) ; fn = lp - tp
    tn = sum(negatives) ; fp = ln - tn
    return confusion_table(tp,fp,tn,fn)
end

function false_negative_rate(confusion)
    tp = confusion[1,1]
    fp = confusion[2,1]
    tn = confusion[2,2]
    fn = confusion[1,2]
    return fn/(fn+tp)
end

function false_positive_rate(confusion)
    tp = confusion[1,1]
    fp = confusion[2,1]
    tn = confusion[2,2]
```

```
    fn = confusion[1,2]
    return fp/(fp+tn)
end

function mismatch_rate(confusion)
   return false_negative_rate(confusion) + false_positive_rate(confusion)
end

function confusion_accuracy(confusion)
    tp = confusion[1,1]
    fp = confusion[2,1]
    tn = confusion[2,2]
    fn = confusion[1,2]
    return (tp+tn)/(tp+tn+fp+fn)
end;

function precision(confusion)
    tp = confusion[1,1]
    fp = confusion[2,1]
    tn = confusion[2,2]
    fn = confusion[1,2]
    return tp/(tp+fp)
end;

function recall(confusion)
    tp = confusion[1,1]
    fp = confusion[2,1]
    tn = confusion[2,2]
    fn = confusion[1,2]
    return tp/(tp+fn)
end;

function F1score(confusion)
    tp = confusion[1,1]
    fp = confusion[2,1]
    tn = confusion[2,2]
    fn = confusion[1,2]
    return 2*tp/(2*tp + fp + fn)
end;
```

Bibliography

[1] R. Horodecki, P. Horodecki, M. Horodecki, and K. Horodecki, "Quantum entanglement," *Reviews of Modern Physics*, vol. 81, pp. 865–942, 6 2009.

[2] A. K. Ekert, "Quantum cryptography based on bell's theorem," *Phys. Rev. Lett.*, vol. 67, pp. 661–663, Aug 1991.

[3] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, "Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels," *Phys. Rev. Lett.*, vol. 70, pp. 1895–1899, Mar 1993.

[4] C. H. Bennett and S. J. Wiesner, "Communication via one- and two-particle operators on einstein-podolsky-rosen states," *Phys. Rev. Lett.*, vol. 69, pp. 2881–2884, Nov 1992.

[5] O. Gühne and G. Tóth, "Entanglement detection," 4 2009.

[6] P. Horodecki, "Separability criterion and inseparable mixed states with positive partial transposition," *Physics Letters A*, vol. 232, no. 5, pp. 333–339, 1997.

[7] A. Peres, "Separability criterion for density matrices," *Phys. Rev. Lett.*, vol. 77, pp. 1413–1415, Aug 1996.

[8] D. F. James, P. G. Kwiat, W. J. Munro, and A. G. White, "Measurement of qubits," *Physical Review A - Atomic, Molecular, and Optical Physics*, vol. 64, p. 15, 2001.

[9] S. P. Walborn, P. H. S. Ribeiro, L. Davidovich, F. Mintert, and A. Buchleitner, "Experimental determination of entanglement with a single measurement," *Nature*, vol. 440, pp. 1022–1024, 4 2006.

[10] O. Gühne, P. Hyllus, D. Bruß, A. Ekert, M. Lewenstein, C. Macchiavello, and A. Sanpera, "Detection of entanglement with few local measurements," *Physical Review A - Atomic, Molecular, and Optical Physics*, vol. 66, p. 5, 2002.

[11] O. Gühne, P. Hyllus, D. Bruss, A. Ekert, M. Lewenstein, C. Macchiavello, and A. Sanpera, "Experimental detection of entanglement via witness operators and local measurements," *Journal of Modern Optics*, vol. 50-6, pp. 1079–1102, 2003.

[12] G. Tóth and O. Gühne, "Detecting genuine multipartite entanglement with two local measurements," *Physical Review Letters*, vol. 94, 2 2005.

[13] S. Gharibian, "Strong np-hardness of the quantum separability problem," 10 2008.

[14] B. M. Terhal, "Bell inequalities and the separability criterion," *Physics Letters A*, vol. 271, pp. 319–326, 2000.

[15] M. Lewenstein, B. Kraus, J. I. Cirac, and P. Horodecki, "Optimization of entanglement witnesses," *Phys. Rev. A*, vol. 62, p. 052310, Oct 2000.

[16] H. S. Park, S. S. B. Lee, H. Kim, S. K. Choi, and H. S. Sim, "Construction of optimal witness for unknown two-qubit entanglement," 6 2010.

[17] C. Ren and C. Chen, "Steerability detection of an arbitrary two-qubit state via machine learning," *Physical Review A*, vol. 100, 8 2019.

[18] Y. Ming, C. T. Lin, S. D. Bartlett, and W. W. Zhang, "Quantum topology identification with deep neural networks and quantum walks," *npj Computational Materials*, vol. 5, 12 2019.

[19] X. Zhang, M. Luo, Z. Wen, Q. Feng, S. Pang, W. Luo, and X. Zhou, "Direct fidelity estimation of quantum states using machine learning," 2 2021.

[20] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," 9 2017.

[21] Y. C. Ma and M. H. Yung, "Transforming bell's inequalities into state classifiers with machine learning," *npj Quantum Information*, vol. 4, 12 2018.

[22] S. Lu, S. Huang, K. Li, J. Li, J. Chen, D. Lu, Z. Ji, Y. Shen, D. Zhou, and B. Zeng, "Separability-entanglement classifier via machine learning," *Physical Review A*, vol. 98, 7 2018.

[23] P. H. Qiu, X. G. Chen, and Y. W. Shi, "Detecting entanglement with deep quantum neural networks," *IEEE Access*, vol. 7, pp. 94310–94320, 2019.

[24] D. J. Griffiths, *Introduction to Quantum Mechanics (2nd Edition)*. Pearson Prentice Hall, 2nd ed., Apr. 2004.

[25] B. M. Terhal, "Detecting quantum entanglement," *Theoretical Computer Science*, vol. 287, pp. 313–335, 2002.

[26] D. Chruściński and G. Sarbicki, "Entanglement witnesses: construction, analysis and classification," 2 2014.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[28] P. Mehta, M. Bukov, C. H. Wang, A. G. Day, C. Richardson, C. K. Fisher, and D. J. Schwab, "A high-bias, low-variance introduction to machine learning for physicists," *Physics Reports*, vol. 810, pp. 1–124, 5 2019.

[29] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, Dec 1943.

[30] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[31] B. Widrow and M. E. Hoff, "Adaptive switching circuits," *1960 IRE WESCON Convention Record*, pp. 96–104, 1960. Reprinted in *Neurocomputing* MIT Press, 1988 .

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 12 2014.

[33] K. Zyczkowski, K. A. Penson, I. Nechita, and B. Collins, "Generating random density matrices," *Journal of Mathematical Physics*, vol. 52, 6 2011.

[34] K. Zyczkowskit and M. Kust, "Random unitary matrices," *J. Phys. A Math. Gen*, vol. 27, pp. 4235–4245, 1994.

[35] J. Ginibre, "Statistical ensembles of complex, quaternion, and real matrices," *Journal of Mathematical Physics*, vol. 6, no. 3, pp. 440–449, 1965.

[36] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, pp. 65–98, 2017.

[37] P. Gawron, D. Kurzyk, and Łukasz Pawela, "Quantuminformation.jl—a julia package for numerical computation in quantum information theory," *PLoS ONE*, vol. 13, 12 2018.

[38] M. Innes, "Flux: Elegant machine learning with julia," *Journal of Open Source Software*, vol. 3, p. 602, 5 2018.

[39] J. Roik, K. Bartkiewicz, A. Černoch, and K. Lemr, "Accuracy of entanglement detection via artificial neural networks and human-designed entanglement witnesses," *Phys. Rev. Applied*, vol. 15, p. 054006, May 2021.