UPPSALA UNIVERSITET

Master Program in Engineering Physics

Project in applied physics

Machine Learning Approach to Classify Quantum Correlations in Quantum States

Authors

Abrahamsson, Tim
Dahlberg, Julia
Rudengren, Isabella

Supervisors

Azimi Mousolou, Vahid
Singh, Prashant
Mattsson, Per

Uppsala

January 15, 2024

Abstract

Quantum entanglement is an integral part of the field of quantum technology. However, it is still difficult to detect whether or not a given state is entangled, unless one is dealing with a pure state, which is rare in experiments. Luckily, there have been many great advances in the field of machine learning recently which has led to an interest in implementing it when detecting entanglement. This project aims to investigate some ways this could be implemented through the use of neural networks. The neural networks are firstly implemented in a standard fashion and later forced to connect to physical concepts. The implementation of the neural network led to a high accuracy model. However, this model lost a large part of its accuracy when trying to connect it to the Bell-inequality. Later, the Bloch inequality was implemented to more generally represent the density matrices of the 2 qubit states used. Through investigation of the regions of inputs used in this representation that led to entanglement, a clear distinction between entangled and separable states was found. This paper shows the potential of using machine learning in the classification of quantum states. However, further research is needed for this to be practically implemented.

Table of contents

1 Introduction

In quantum information theory, the concept of entanglement, which is a property of quantum states, is of utmost importance since it is what allows quantum computers to act differently from, and solve some problems more efficiently than classical computers. However, to actually use entanglement in quantum computers one will need to be able to identify when a pair of quantum bits, or qubits, are entangled. In general, there are two kinds of quantum states, mixed states and pure states, and it is currently much easier to identify entanglement in pure states but experiments rarely produce a pure state. In addition to this, for more complex systems than two- or three-qubit systems, it is very difficult to identify entanglement.[1][2]

As the field of quantum computing has advanced, so has the field of machine learning, which has led to an interest in how they can be connected. Experiments have been performed investigating how this can be done and this project further analyzes how neural networks can be useful when classifying quantum states as entangled or separable. The project can be divided into three different parts. The first part is an analysis of how well a neural network could classify entangled states given a certain input. The second part is an attempt to connect this neural network to a physical classification to make sure that the classifications make physical sense. The final part is an investigation of parameter values in a more general representation of two-qubit systems to see how they impact the entanglement of the state.[3]

2 Implementation using a shallow neural network

The first part of the project is based on the article ”Entanglement detection using artificial neural networks” by Naema Asif et.al [3]. In this article machine learning models defined as neural networks with different number of nodes are trained and evaluated to be able to classify bipartite (two-qubit) quantum states as either entangled or separable (disentangled). The results are then compared with entanglement witnesses, which functions as theoretical expressions for classifying quantum states, to prove the machine learning improvement.

In this part of the report, shallow neural networks with different number of nodes are created, trained and evaluated based on the article. The models should be able to classify bipartite quantum states as either entangled or separable as accurate as possible. Thus, this is binary classification problem. The models described here are similar to those in the article, however, some slight changes have been done to improve performance. The results are then compared with results from the previous mentioned article. In this report there is also a small analysis of comparing two different input features to the trained models, which is not done in the article.

2.1 Theory and Method

Figure 1 shows the procedure of work in this part of the project. Firstly the data is generated in Matlab and then transferred to Python where the machine learning (ML) models are defined, trained and evaluated. The individual parts of the workflow will be more thoroughly explained further in the article in chronological order.

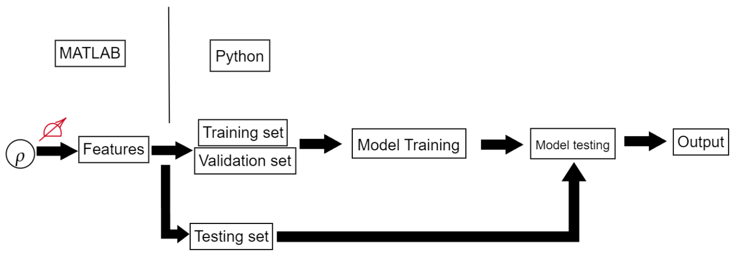


Figure 1: Workflow of the process from the generation of data to the analysis of the performance of the machine learning.

2.1.1 Data generation

Firstly a large data set was generated in Matlab using the same methods as in article [3], i.e through the usage of the QETLAB package (a Matlab package used for quantum physical calculations). The large data set consists of seven smaller data sets which were created from different bipartite (two-qubit) quantum state families to get a diverse set of data points, and hence a more general solution. Each smaller data set consists of 50,000 data points which were put together to produce one big data set of in total, 350,000 data points. A data set only used for testing consisting of 50,000 data points was also created.

For each data point a density matrix, ρ, was created that represents a bipartite quantum state. From this state, ρ, measurements were performed to extract features which were used as input to our machine learning models. The data is also labeled. The method to produce each of the seven smaller data sets is described below. Description of the generation of these data sets require the use of Dirac notation which represents quantum states as state vectors in a complex vector space using what is known as kets ($|\alpha\rangle$)

and their conjugate transpose known as bras ($\langle\alpha|$) [4]. Later on in the article these state vectors will be rewritten to their vector form for Matlab calculations. While these state vectors can only represent pure states, the density matrices are able to represent both pure and mixed states.

Generating density matrices The first data set, containing 50 000 data points, was generated from the density matrix

$$\rho_{\theta,\phi} = p\,|\psi_{\theta,\phi}\rangle\,\langle\psi_{\theta,\phi}| + (1-p)\frac{I}{4}. \tag{1}$$

Here, p is the noise factor given by a random value between 0 and 1, and was obtained in Matlab by using the function `rand`, and I is a 2x2 identity matrix. $|\psi_{\theta,\phi}\rangle$ is a quantum state represented by

$$|\psi_{\theta,\phi}\rangle = cos\left(\frac{\theta}{2}\right)|01\rangle + e^{i\phi}sin\left(\frac{\theta}{2}\right)|10\rangle \tag{2}$$

where θ is a random value between 0 and π, and ϕ is a random value between 0 and 2π, both implemented by using the `rand` function. Furthermore, $|10\rangle$ and $|01\rangle$ were implemented as their matrix representations on the form $[0\,1\,0\,0]^T$ and $[0\,0\,1\,0]^T$, respectively. The second data set, containing another 50 000 data points, was also created using the density matrix in Eq(1), however, $|\psi_{\theta,\phi}\rangle$ is for this data set given by

$$|\psi_{\theta,\phi}\rangle = cos\left(\frac{\theta}{2}\right)|00\rangle + e^{i\phi}sin\left(\frac{\theta}{2}\right)|11\rangle \tag{3}$$

where $|00\rangle$ and $|11\rangle$ were implemented as their matrix representations on the form $[1\,0\,0\,0]^T$ and $[0\,0\,0\,1]^T$.

The third and fourth data sets exclusively contains entangled and separable quantum states, respectively. From the QETLAB package, random density matrices were produced by using the `RandomDensityMatrix` function. To generate 50,000 entangled states for the third data set, each density matrix was checked in a while loop using the PPT criterion through the `IsPPT` function. This criterion was also used for labeling the data sets and is explained in the paragraph *Labeling data.* Moreover, to create 50,000 separable states for the fourth data set, the product of two different random density matrices was calculated, which ensures non-entanglement for the states by definition.

For the fifth, sixth and seventh data set, each consisting of 50,000 quantum states, the expression in equation (1) was used again to produce the density matrices for the quantum states. For the fifth data set the state vector, $|\psi_{\theta,\phi}\rangle$, is a pure state acquired from the function `RandomStateVector` in the QETLAB package. For the sixth and seventh data sets, the $|\psi_{\theta,\phi}\rangle$ is given by the two different Bell states

$$|\psi_+\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) \tag{4}$$

$$|\phi_+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), \tag{5}$$

respectively.

Another bipartite data set consisting of 50,000 data points only used for testing, was also generated from equation 1. Here $|\psi\rangle$ is the Bell state defined in equation 6 and p is the noise factor defined as $p \in (0,1)$. A quantum state of this definition is defined as entangled if $p > \frac{1}{3}$ and separable if $p < \frac{1}{3}$ [3]. When generating this data set in Matlab, the values of p are evenly spaced values between 0 and 1 (excluding 0, 1 and $\frac{1}{3}$).

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{6}$$

Feature extraction After the density matrices were created, features was calculated for each density matrix to be used as an input feature in the machine learning algorithm. Two different features were used and then compared. The first one, also used in the article by Naema Asif et.al. [3], is the relative entropy of coherence of a quantum state. The second one is the eigenvalues to a matrix U created from the density matrices. More details about these features and how they were extracted will be explained below.

The relative entropy of coherence of a quantum state ρ is a base dependent quantity defined as in equation 7 below. $S(\rho)$ is the von-Neumann entropy of the quantum state ρ defined as $S(\rho) = -\mathrm{tr}(\rho \log \rho)$ and ρ^d is the diagonal state of ρ defined as $\rho^d = \sum_i \langle i| \rho |i\rangle |i\rangle \langle i|$ where $|i\rangle$ is a reference basis. The basis sets used are the same as in the article and they are defined in equation 8 where $\phi = \pi$. [3] The first set of input features is therefore: $\{C_r(QS, \rho), C_r(RS, \rho), C_r(RT, \rho), C_r(QT, \rho)\}$.

$$C_r(\rho) = S(\rho^d) - S(\rho) \tag{7}$$

$$\begin{aligned} &Q : \{|0\rangle, |1\rangle\} \\ &R : \{|R_+\rangle, |R_-\rangle\} = \left\{ \tfrac{1}{\sqrt{2}}(|0\rangle + ie^{i\phi}|1\rangle), \tfrac{1}{\sqrt{2}}(|0\rangle - ie^{i\phi}|1\rangle) \right\} \\ &S : \{|S_+\rangle, |S_-\rangle\} = \left\{ \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \right\} \\ &T : \{|0\rangle, |1\rangle\} \end{aligned} \tag{8}$$

This feature extraction are based on the Bell-inequality for the relative entropy of coherence, defined in equation 9. It is an entanglement witness that can classify states as entangled or separable by analyzing if the inequality is violated or not for quantum states. The inequality is based on a thought experiment concocted by John Bell, where he creates and inequality which is solely based on the presumption of local realism, which can be boiled down to the assumption that the results of one measurement is unaffected by operations on a separate system.[5] In classical physics this equality always holds, but when one also accounts for quantum mechanics the inequality is, on average, broken. Generally, entangled states violates the inequality and separable satisfies it, however there exists many exceptions [3]. This Bell-inequality is therefore used as an comparison to our machine learning model.

$$C_r(QS, \rho_{AB}) + C_r(RS, \rho_{AB}) + C_r(RT, \rho_{AB}) - C_r(QT, \rho_{AB}) \leq 4 \tag{9}$$

The second feature is the three eigenvalues to the matrix U defined by equation 11, where T_ρ is defined in equation 10. Here σ_n and σ_m represent the Pauli matrices $\sigma_1 = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, $\sigma_2 = \left(\begin{smallmatrix} 0 & -i \\ i & 0 \end{smallmatrix}\right)$ and $\sigma_3 = \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$, T_ρ^T is the matrix transpose of T_ρ and ρ is the density matrices generated in paragraph *Generating density matrices*. The second set of input features is therefore: $\{u_1, u_2, u_3\}$ where u_i is an eigenvalue to matrix U. [1]

$$T_\rho = [t_{nm}] = [Tr(\rho \sigma_n \otimes \sigma_m)] \tag{10}$$

$$U = T_\rho^T T_\rho \tag{11}$$

This feature extraction is based on the Bell-CHSH inequality together with the Bloch representation of the density matrices which can be seen in equation 12. The lower right matrix consisting of the T-elements in this representation is referred to as the correlation matrix. [6]

$$\rho = \frac{1}{4}\left(I \otimes I + \vec{x} \cdot \vec{\sigma} \otimes I + I \otimes \vec{y} \cdot \vec{\sigma} + \sum_{n,m=1}^{3} T_{nm} \sigma_n \otimes \sigma_m \right) \tag{12}$$

For a two-qubit system, the Bell-CHSH inequality is given by[6]

$$|Tr(\rho\, \mathcal{B}_{CHSH})| \leq 2 \tag{13}$$

According to the Horodecki theorem the maximum value for the Bell-CHSH operator for a state ρ is given by

$$\max_{\mathcal{B}_{CHSH}} Tr(\rho\, \mathcal{B}_{CHSH}) = 2\sqrt{M(\rho)} \tag{14}$$

where $M(\rho) = \max\limits_{j<k} \{h_j + h_k\} \leq 2$, and h_j are the eigenvalues of the matrix $U = T^T T$. This gives us the condition that for $M(\rho) > 1$ the Bell-CHSH inequality is violated and therefore the state is non-local, which represents action at a distance between states is possible. Since we now have a correlation between the eigenvalues of the U-matrix and whether the state is local or not, we can use these eigenvalues as input parameters for the training and evaluation of a machine learning model.

Labeling data To label the data a criterion called Positive Partial Transpose (PPT) criterion or Peres-Horodecki criterion was used. This is a criterion that can determine if for example two- or three-qubit quantum states are entangled or separable by computing the eigenvalues of the partial transpose of the density matrix of a quantum state, ρ. The state is defined as entangled if any eigenvalues are negative and separable if all eigenvalues are non-negative [7]. The QETLAB function `IsPPT` was used to calculate the positive partial transpose of the density matrices, ρ, in Matlab and the label 1 correspond to an entangled state and the label 0 corresponds to a separable state.

2.1.2 Implementing Machine Learning

When the data sets had been generated with features and labels, these are saved in a csv-file and transferred to Python were the machine learning algorithm is implemented. The machine learning algorithm is a neural network implemented in Python (Google Colab) using the Pytorch package and the Scikit-Learn package. The models were defined to solve a binary classification problem defined as being able to correctly classify bipartite quantum states as either entangled or separable.

Basics Neural Network Before explaining the definition of the machine learning model used here, the basics of the type of neural networks used here will be shortly explained. A neural network is a type of machine learning model consisting of several layers represented by the rows of circles in figure 2. It has one input layer (the layer to the left) and one output layer (rightmost layer). It can also have one or several hidden layers which are between the input and the output layer. Each layer has a number of nodes and they in turn contains weights and biases which are model parameters updated through the training process of the neural network. The output, y from a node is given by the equation 15 (also seen in figure 2) where w_i represents the weights, b_i the biases and x_i the input data. f is an activation function defined by the user.

$$y = f\left(\sum_i w_i x_i + b_i\right) \tag{15}$$

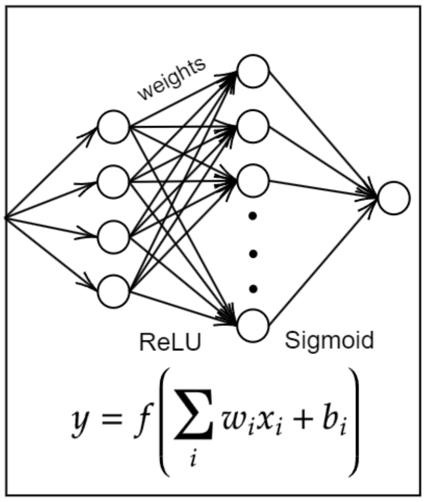
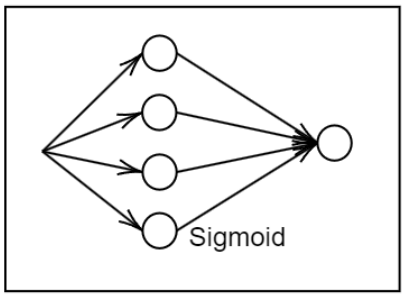


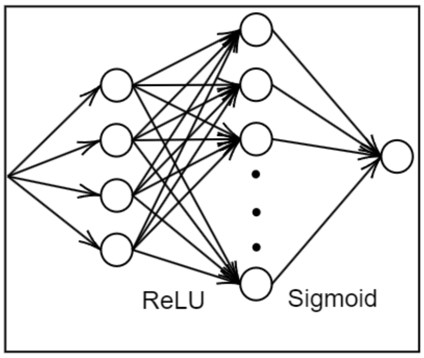
Figure 2: A basic visualization of a neural network with the equation for the output from one node.

A neural network is trained by feeding it the input features of a training data set. The model produces a predicted output which is then fed into a user-defined loss function which calculate the loss. The loss is here defined as the difference between the predicted output from the model and the correct output given by the labels from the training data set. The loss is then used to calculate gradients which are used to update the model parameters in the neural network (the weights and biases) using an optimizer function (also user-defined). The whole process is then repeated a number of times depending on the number of epochs and batches you have. The number of epochs is defined as the number of times you feed the whole training data set to the model and when using batches one divides the whole training data set into several smaller parts which are then fed to the model one at a time. Using batches increases the number of times the model parameters are updated every epoch. After the training process the model is evaluated and tested.[8]

Machine Learning Models Three different neural network models were implemented in accordance with article [3]: a simple neural network with no hidden layer (0 nodes) and two models with a hidden layer with 10 nodes and 50 nodes respectively. A visualization of these models can be seen in figure 3 where the simplest model is visualized by figure 3a and the other to models by figure 3b.



(a) 0 nodes



(b) 10 or 50 nodes

Figure 3: A visualization of the neural networks used.

All models have an input layer and an output layer. The input layers of all models have a number of nodes corresponding to the size of each type of input feature. When using the data set with the relative entropy of coherence as input feature, the input layer has four nodes, and when using the eigenvalues to the matrix U as input feature, the input layer has 3 nodes. The output layer has one node which corresponds to the model's class prediction (entangled or separable).

The models also have one or several activation functions. The simplest model (0 nodes) have an sigmoid activation function. For the other two models, which contains a hidden layer, two linear activation function were used: first a rectified linear activation function (ReLU) and then a sigmoid activation function. Using ReLU introduces a desired non-linearity in our model which makes the model able to learn more complex correlations. To implement these activation functions in our neural networks, the functions `nn.Sigmoid` and `nn.ReLU` from Pytorch were used.

For the two models with 10 and 50 nodes the weights and biases in the neural network are also initialized in accordance with article [3]. The weights are initialized uniformly (also called He initialization) using the function `nn.init.kaiming_uniform_` from the Scikit-Learn package and the all biases are set to 0.01.

Algorithm In the following part the algorithm for the implementation of the machine learning models is described in accordance with the workflow in figure 1. Firstly, the data generated in Matlab was read from the csv-file. The large data set was then split into a training set, validation set and test set of 70% (245,000 data points), 20% (70,000 data points) and 10% (35,000 data points) of the total data points respectively.

The randomized split was done using Scikit-learn's function `train_test_split` twice. To improve the accuracy results, the input data was then normalized using Scikit-learn's function `MinMaxScaler` (it was not clear whether this was done in article [3]).

In the next part, the training process, the different models were trained using some different training parameters which were tuned over a few runs for each model. These can be seen in table 1 but they are also explained below.

Model	Learning rate	Batch size	Early stopping start	Early stopping limit
0 nodes	0.01	Training size	100	30
10 nodes	0.001	32	100	50
50 nodes	0.001	32	None	None

Table 1: Parameter specifications for final ML models.

The loss function used during training (and evaluation) was binary cross-entropy (BCE), and Root Mean Squared Propagation (RMSProp) was used as an optimizer. They were implemented using the built-in functions `BCELoss()` and `RMSprop()` with default parameters except for the learning rate in the optimizer which was varied according to the table.

In the training process an upper limit of 250 epochs (learning iterations) was set and batches of different sizes were used which can be seen in table 1. When training the models with 10 nodes and 50 nodes the training most often used all epochs, while for the simplest model with 0 nodes the training lasted for approximately 100 epochs. Early stopping and checkpointing was also implemented. Early stopping stops the training process if the model has not become more accurate for a user set limit of epochs and makes sure the model does not train for too long and become overfitted. Checkpointning saves the model with the highest performance (highest accuracy) so the best model can be used for the evaluation phase. To avoid overfitting our models, the loss and accuracy on the training data as well as the accuracies evaluated on the training and validation data respectively were plotted.

In the evaluation process, following the training process, the trained models are evaluated on the test data from the large data set and the test data set created in paragraph *Generating density matrices* from equation 1 and 6. The test data sets are from now on called test data set 1 and test data set 2. Accuracies as well as the number of truly/falsely entangled/separable states are calculated on both test data sets. The results can be seen in section 2.2.

This whole process described above is executed for each type of neural network for the data set containing the relative entropy of coherence values and for the data set containing the eigenvalues to the matrix U, one at a time. The models trained on the data set containing the relative entropy of coherence values are from now on called *Model_ Cr* and the models trained on the data set containing the eigenvalues to the matrix U are called *Model_ ueig.* The process is also repeated three times for each model with different values on the seed which controls the random number generator in the `train_test_split`. This makes it possible to vary the data points in the training data set and the validation data set. The test data set is, however, always the same. In total there are 18 different neural network models.

2.2 Results

In this section the performance of the different machine learning models will be presented using terms such as accuracy, truly entangled (TE), truly separable (TS), falsely entangled (FE) and falsely separable (FS). Accuracy stands proportion of states that were correctly predicted. TE/TS are the number of states correctly predicted as entangled/separable and FE/FS are the number of states wrongly predicted as entangled/separable.

Firstly the generated data sets are classified using the Bell-type inequality for relative entropy of coherence and the Bell-CHSH inequality in equation 9 and 13 respectively. Using the Bell-type inequality for relative entropy of coherence on our corresponding data set consisting of values of relative entropy of coherence, we get an accuracy of 44.6% and the model solely predicts the states as separable. This indicates that the inequality is not able to detect any entangled states in our data set. Using the Bell-CHSH inequality as an entanglement witness on the data set consisting of eigenvalues to the matrix U gives an accuracy of 59.94% and it manages to correctly predict 53756 out of 193976 entangled states.

The accuracy results from evaluating the machine learning models on the test data set 1 (test data set received from the train-validation-test split) can be seen in table 2 where table 2a and 2b present the results for the models trained on the values of relative entropy of coherence (Model_Cr) and eigenvalues of the matrix U (Model_ueig), respectively. The corresponding number of truly entangled TE, FE, TS and FS states for each set of models can be seen in figure 4 where figures 4a, 4c and 4e show the results for the Model_Cr and figures 4b, 4d and 4f show the results for the Model_ueig.

The results from evaluating the model on the test data set 2 defined in equation 1 and 6 is seen in table 3a with a corresponding graph indicating the relationship between the predicted probability and the noise factor p in figures 5 and 6. As in the previous mentioned tables and figures table 3a and figures 5a, 5b and 5c are results from the Models_Cr and table 3b and figures 6a, 6b and 6c are results from the Models_ueig.

Table 2: Accuracy values for models evaluated on test data set 1.

(a) Models trained on coherence.

Model set\Nodes	0	10	50
Model1_Cr	81.39%	96.91%	97.58%
Model2_Cr	81.46%	96.39%	96.59%
Model3_Cr	81.74%	97.03%	96.91%
Mean	81.53%	96.78%	97.03%
std	0.19%	0.34%	0.51%

(b) Models trained on eigenvalues to the matrix U.

Model\Nodes	0	10	50
Model1_ueig	88.22%	97.72%	98.95%
Model2_ueig	91.92%	98.11%	99.39%
Model3_ueig	91.84%	98.51%	99.26%
Mean	90.66%	98.11%	99.20%
std	2.11%	0.40%	0.23%

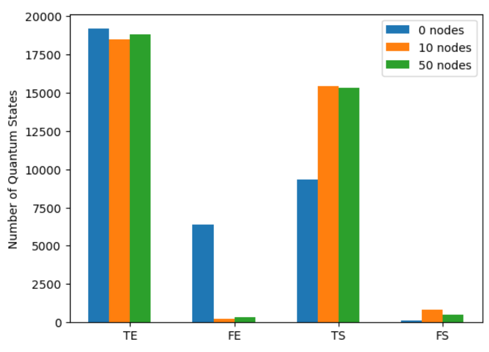
Table 3: Accuracy values for models evaluated on the test data set 2.

(a) Models trained on coherence.

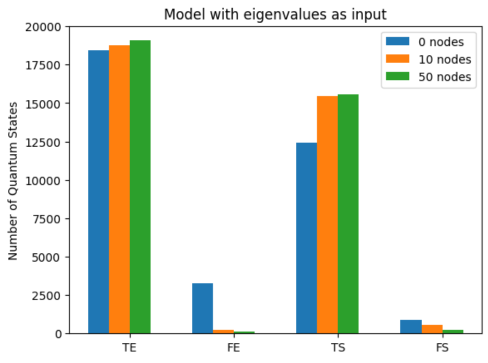
Model set\Nodes	0	10	50
Model1_Cr	99.222%	99.734%	99.994%
Model2_Cr	98.208%	99.952%	99.980%
Model3_Cr	98.784%	99.936%	99.992%
Mean	98.738%	99.874%	99.989%
std	0.509%	0.122%	0.008%

(b) Models trained on eigenvalues to the matrix U.

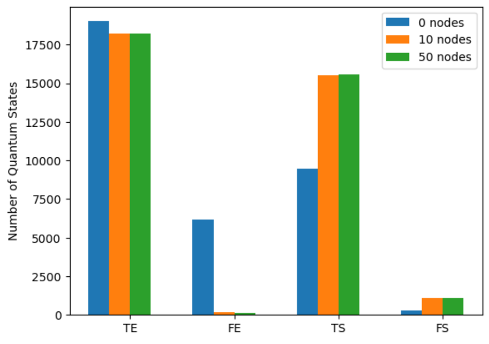
Model\Nodes	0	10	50
Model1_ueig	99.828%	99.790%	99.750%
Model2_ueig	99.794%	99.806%	99.802%
Model3_ueig	99.990%	99.996%	99.918%
Mean	99.871%	99.864%	99.823%
std	0.105%	0.115%	0.086%



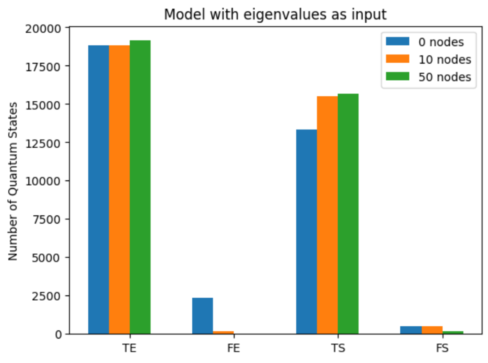
(a) Model1_Cr



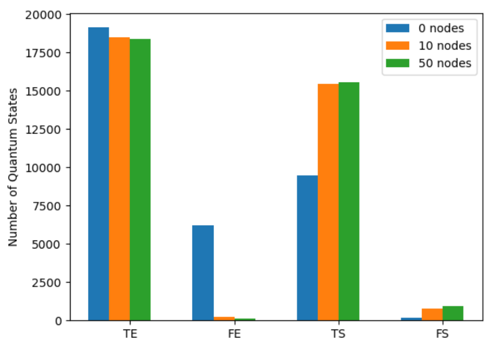
(b) Model1_ueig



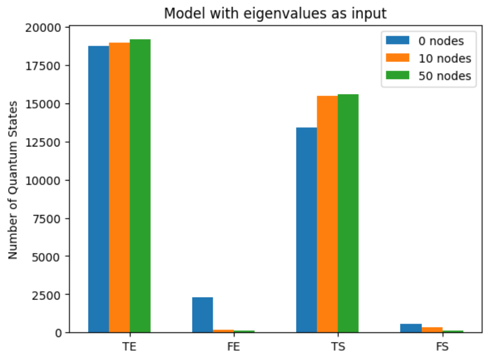
(c) Model2_Cr



(d) Model2_ueig

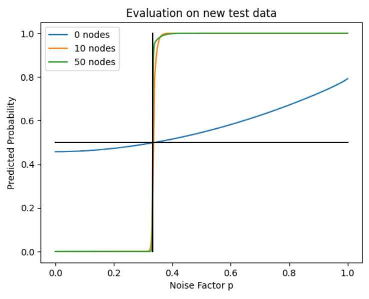


(e) Model3_Cr

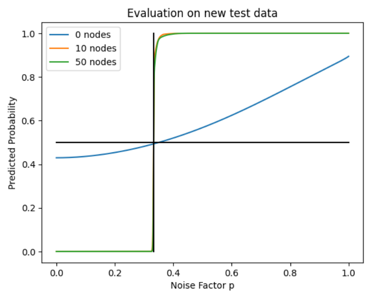


(f) Model3_ueig

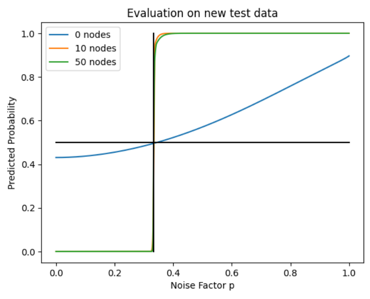
Figure 4: Number of truly entangled (TE), falsely entangled (FE), truly separable (TS) and falsely separable (FS) states for different number of nodes for trained models.



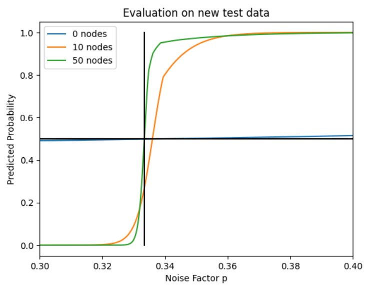
(a) Model1_Cr



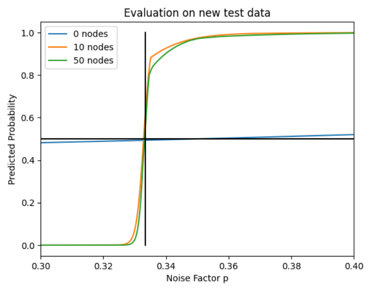
(b) Model2_Cr



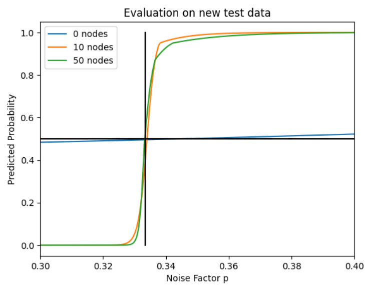
(c) Model3_Cr



(d) Model1_Cr Zoomed

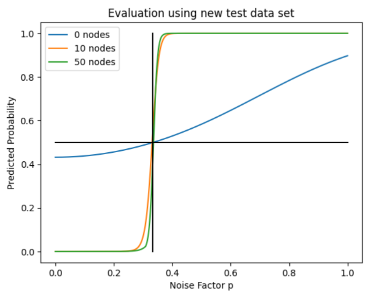


(e) Model2_Cr Zoomed

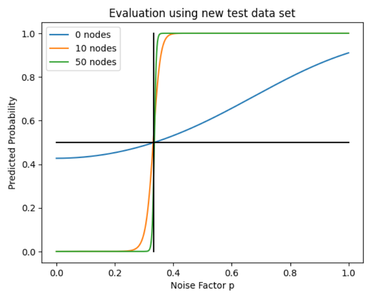


(f) Model3_Cr Zoomed

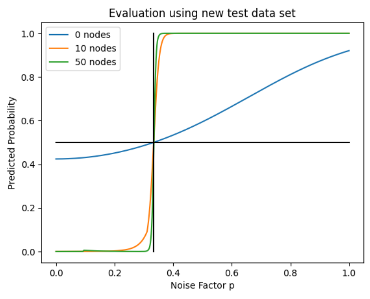
Figure 5: Predicted probability that a quantum state is predicted as entangled or separable as a function of the noise factor p for different nodes for models trained on relative entropy of coherence. The horizontal line indicates where the predicted probability is 0.5 and the vertical line marks where $p = \frac{1}{3}$. The subfigures in the lower row pictures the area close to the vertical line.



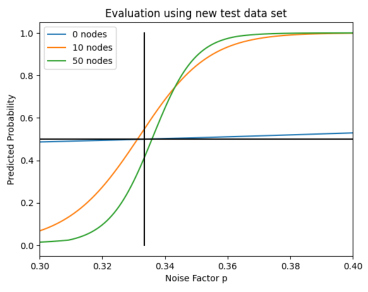
(a) Model1_ueig



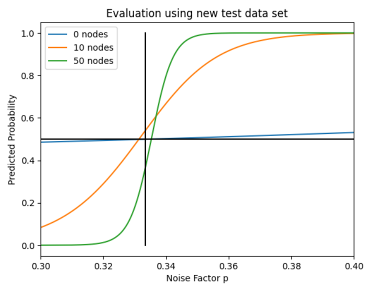
(b) Model2_ueig



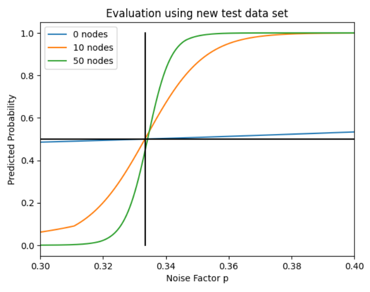
(c) Model3_ueig



(d) Model1_ueig Zoomed



(e) Model2_ueig Zoomed



(f) Model3_ueig Zoomed

Figure 6: Predicted probability that a quantum state is predicted as entangled or separable as a function of the noise factor p for different nodes for models trained on eigenvalues to the matrix U. The horizontal line indicates where the predicted probability is 0.5 and the vertical line marks where $p = \frac{1}{3}$. The subfigures in the lower row pictures the area close to the vertical line.

2.3 Discussion

The machine learning models trained on values of relative entropy of coherence have an average accuracy of 81.53% (0.19%) for the simplest model (0 nodes) and 97.03% (0.51%) for the ML model with 50 nodes for an evaluation on data set 1 (table 2a). Compared to the corresponding accuracies presented in article [3] (78.18% and 94.62%) a higher accuracy was achieved for the ML models implemented in this project. The increased accuracy could be due to the differences in our machine learning models which mainly consist of the increased number of epochs (250 instead of 100) and the implementing of batches which results in more and longer training of our models. It is unclear what preparations are done to the data sets in article [3], however normalization might have an involvement in the increased accuracy as well.

From the figures of the truly/falsely entangled/separable states in figure 4 one can note that the models with 0 nodes have the highest values of truly entangled state, but also falsely entangled states. The models with 10 and 50 nodes are therefore more preferable since the number of falsely entangled states are much smaller.

When comparing the results from the Model_Cr and the results from the Model_ueig in table 2 one notices that the average accuracies are higher for the Model_ueig. For these models the number of truly predicted states increases with increasing number of nodes and the number of falsely predicted states decreases (see figure 4), whereas for the Model_Cr this pattern is not visible. Therefore the Model_ueig seems to perform better on the test data set 1.

However, if looking at the average accuracies for evaluation on the test data set 2 in table 3, the Model_Cr with 50 nodes performs best out of all models with an average accuracy of 99.989% (0.008%). For the Model_ueig the model with 0 nodes performs best, so the accuracy is decreasing with increasing number of nodes, which seems contradictory. However, worth noting, the accuracies are all above 98% for all models, so all models performs very well on the test data set 2. This result is notable in figure 5 and 6 as well since an increasing number of nodes mostly results in an steeper gradient at the horizontal line $p = \frac{1}{3}$ and that the lines go through the cross-point between the horizontal and the vertical line.

Note that the ML models with 50 nodes have no early stopping since the accuracy slowly increases for all epochs, and would probably reach an even higher accuracy with a higher number of epochs. However, due to time limits the number of epochs are limited to 250 since this results in a training time of 35-40 minutes. Higher accuracies could perhaps also bed reached by tuning the parameters for the training of models, to generate more accurate models.

3 Neural network and the Bell-inequality

The previous part of this report showed that a shallow neural network can classify quantum states with accuracies in some cases reaching up to 99%. However, it is difficult to make a quantum physical interpretation from the machine learning model, which could be a desirable feature. Therefore, in this part of the project an attempt to implement a modified version of the Bell-inequality in the neural network will be presented and analyzed.

3.1 Theory

3.1.1 Modified Bell-inequality

From the Bell-inequality for the relative entropy of coherence formulated in equation 9 the inequality is reformulated by adding independent probabilities P_{AB} to each term where $P_{AB} \in [0,1]$ and AB is the basis. Since

$$P_{QS}C_r(QS,\rho_{AB}) + P_{RS}C_r(RS,\rho_{AB}) + P_{RT}C_r(RT,\rho_{AB} - P_{QT}C_r(QT,\rho_{AB}) \leq (P_{QS}+P_{RS}+P_{RT}+P_{QT})\cdot 4 = 4$$

the Bell-inequality can be written as in equation 16, where the bases Q, R, S and T are defined in equation 8.

$$P_{QS}C_r(QS,\rho_{AB}) + P_{RS}C_r(RS,\rho_{AB}) + P_{RT}C_r(RT,\rho_{AB}) - P_{QT}C_r(QT,\rho_{AB}) \leq 4 \tag{16}$$

The goal is to determine the probabilities $\{P_{AB}\}$ so that the inequality in equation 16 can classify quantum states as accurate as possible. However, since the model will output a set of values of $\{P_{AB}\}$, it is of interest to analyze if the individual parameter values tend to converge or not. Also for simplicity the probabilities $\{P_{AB}\}$ are from now on redefined as $P_1 = P_{QS}$, $P_2 = P_{RS}$, $P_3 = P_{RT}$ and $P_4 = P_{QT}$, so equation 16 can be rewritten as:

$$P_1C_r(QS,\rho_{AB}) + P_2C_r(RS,\rho_{AB}) + P_3C_r(RT,\rho_{AB}) - P_4C_r(QT,\rho_{AB}) \leq 4 \tag{17}$$

Since our input data set take values in the range $(1\cdot 10^{-14},\ 1.4)$, $P_{AB} \in [0,1]$ and that the original Bell-inequality was unable to detect any entangled states in our data set (see section 2.2), an additional term might be needed for the entangled states to disobey the inequality. Therefore, there is also an attempt adding an extra term, P_0 to the inequality resulting in equation 18.

$$P_0 + P_1C_r(QS,\rho_{AB}) + P_2C_r(RS,\rho_{AB}) + P_3C_r(RT,\rho_{AB}) - P_4C_r(QT,\rho_{AB}) \leq 4 \tag{18}$$

3.1.2 Machine learning model

Two different kind of models are defined, trained and tested here representing two different strategies. The first machine learning model is here defined as a neural network with one input layer, one hidden layer and one output layer, in total 3 layers. The input layer consists of four nodes, representing the four input features, the hidden layer has ten nodes and the output layer has four nodes, representing the four probabilities P_{QS}, P_{RS}, P_{RT} and P_{QT}. The model is trained on the data set based on relative entropy of coherence described in section 2.1.1 and has two activation functions: ReLU and Sigmoid. A visualization of the model can be seen in figure 8a.

It is in the loss function where the modified Bell-inequality is defined and used, which can be seen in figure 7. The loss function takes the four output values from the neural network and inserts them in equation 16 as the set of probabilities. The expression on the left side of the inequality is then evaluated using the output models from the neural network and the corresponding C_r-values. The resulting sum is then fed into the in-built Python function `torch.nn.functional.binary_cross_entropy_with_logits` together with the labels of our data set. This function firstly evaluates the sum using a sigmoid function and then use binary cross-entropy to calculate the loss. The resulting loss is then used when performing the back-propagation of the machine learning model. As in the previous section `RMSProp` is defined as the optimizer of the ML model.

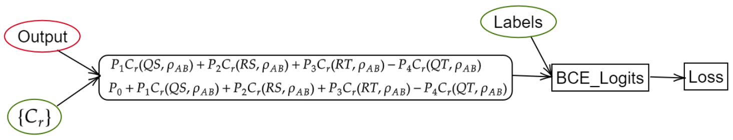
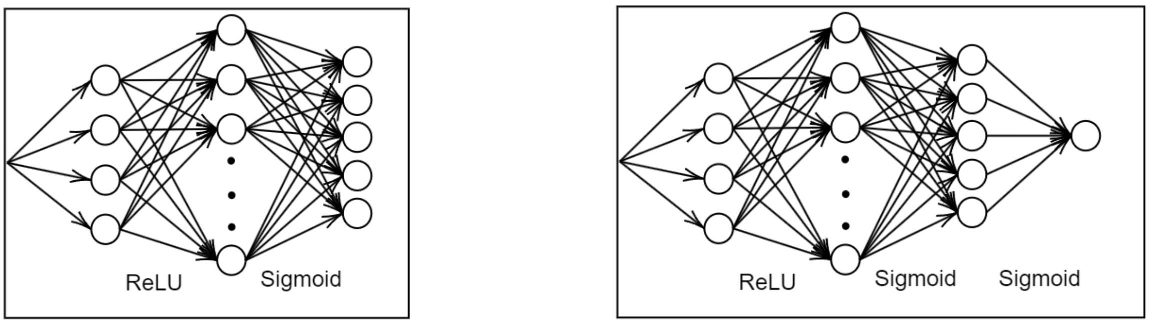

Output
{C_r}
Labels
$P_1C_r(QS,\rho_{AB}) + P_2C_r(RS,\rho_{AB}) + P_3C_r(RT,\rho_{AB}) - P_4C_r(QT,\rho_{AB})$
$P_0 + P_1C_r(QS,\rho_{AB}) + P_2C_r(RS,\rho_{AB}) + P_3C_r(RT,\rho_{AB}) - P_4C_r(QT,\rho_{AB})$
BCE_Logits
Loss

Figure 7: A visualization of the loss function used for the 3 layer model.

The second model is an extension of the firstly described model. It consist of one extra layer containing one node, placed after the previous output layer resulting in a total of 4 layers. A visualization of the model can be seen in figure 8b. This model is trained on the same data set as the first model. This model has one ReLU and two Sigmoid activation functions and uses BCE as a loss function. The probabilities are here extracted as the outputs from the nodes in the third layer.



(a) 3 layer model (b) 4 layer model

Figure 8: A visualization of the two neural networks used when implementing the Bell-inequality into ML models.

To implement the Bell-inequality with an extra term (equation 18) into the machine learning models, the number of nodes in the output layer of the 3 layer model and the third layer in the 4 layer model, are changed to five (instead of four). For the inequality in the loss function and for the evaluation, equation 18 is used (instead of equation 16).

3.2 Method

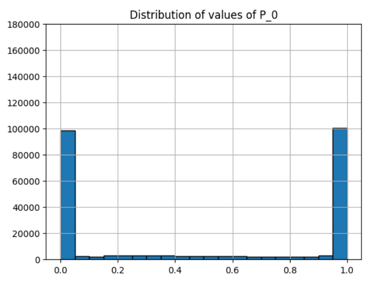
The two neural network models, with and without P_0, are trained on the data set containing the relative entropy of coherence features using approximately the same procedure as described in section 2.1.2. However, here the training was executed for 50 epochs, used batches with a batch size of 32 and the optimizer had a learning rate of 0.001. An evaluation was performed at the end of every epoch where the loss and the accuracy on the training data set, and the accuracy on the validation data set were calculated using accuracy metrics. The number of truly and falsely entangled states were also noted. Note that the input data to the model was not normalized in this part of the project.

3.3 Result

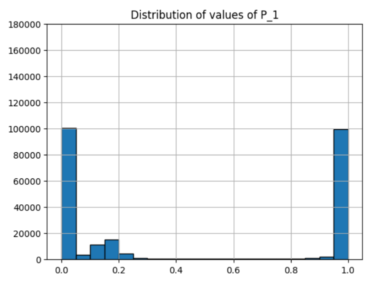
The results presented below are from an evaluation done on the fully trained models of 50 epochs. The accuracy results and the number of states identified as TE, TS, FE and FS for evaluation on the training data set, validation data set and test data set 1, can be seen in table 4. For comparison the accuracy results received when evaluating the different data sets on the ”original” Bell-inequality in equation 9, are also included in the table. The distribution of probability values received from the 3 layer model with and without the extra term P_0 can be seen in figure 9 and 10 respectively. This distribution was not done for the 4 layer model since the model didn’t detect any states as TE.

Table 4: Accuracy results and number of TE, FE, TS and FS predicted by the Bell-inequality (eq. 9) and the neural network implemented with the modified Bell-inequality and with/without extra term P_0.

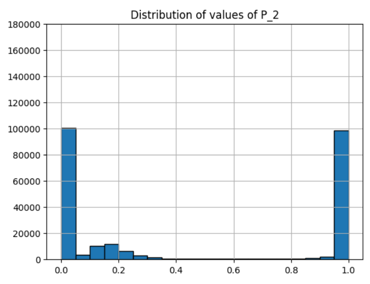
Data set	Model	Accuracy	TE	FE	TS	FS
Train	Inequality	44.577%	0	0	109213	135787
	3 layer With P_0	49.272%	11503	0	109213	124284
	3 layer Without P_0	44.878%	737	0	109213	135050
	4 layer With P_0	44.577%	0	0	109213	135787
	4 layer Without P_0	44.577%	0	0	109213	135787
Validation	Inequality	44.454%	0	0	31118	38882
	3 layer With P_0	49.196%	3319	0	31118	35563
	3 layer Without P_0	44.771%	222	0	31118	38660
	4 layer With P_0	44.454%	0	0	31118	38882
	4 layer Without P_0	44.454%	0	0	31118	38882
Test	Inequality	44.837%	0	0	15693	19307
	3 layer With P_0	49.317%	1568	0	15693	17739
	3 layer Without P_0	45.154%	111	0	15693	19196
	4 layer With P_0	44.837%	0	0	15693	19307
	4 layer Without P_0	44.837%	0	0	15693	19307



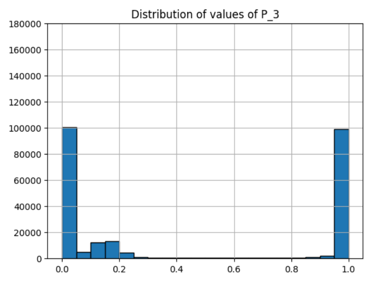
(a) P_0



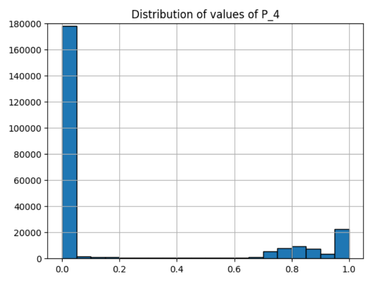
(b) P_1



(c) P_2

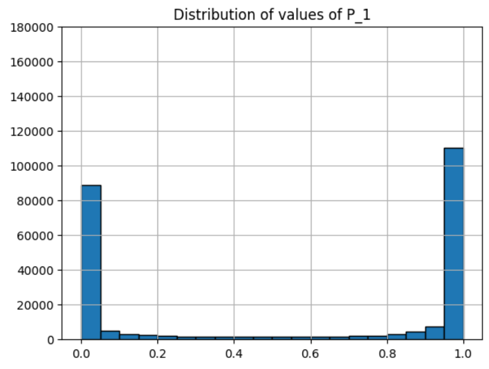


(d) P_3

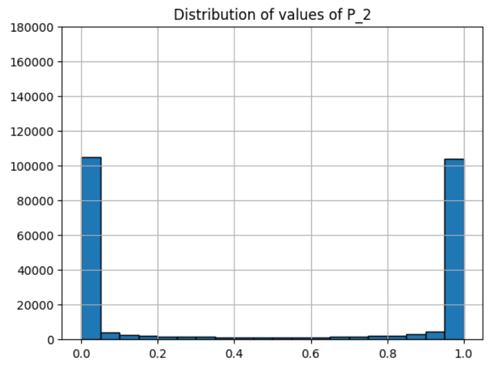


(e) P_4

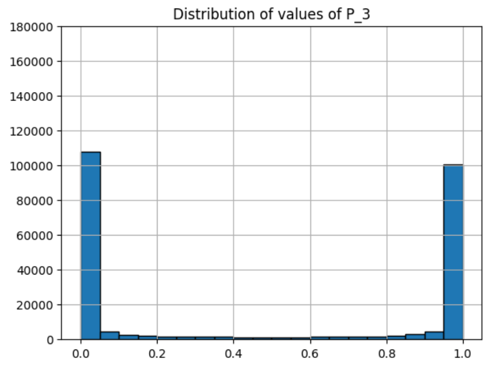
Figure 9: Distribution of individual probability values P_i for neural network with extra term P_0.



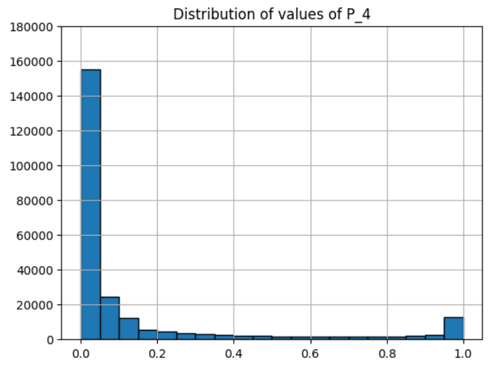
(a) P_1



(b) P_2



(c) P_3



(d) P_4

Figure 10: Distribution of individual probability values P_i for neural network without extra term P_0.

3.4 Discussion

From table 4 it is possible to note that both of the ML models are inaccurate. However, the 3 layer model performs better than the 4 layer model since this kind of model manage to detect some entangled states when the other model doesn't. The "original" Bell-inequality doesn't manage to detect any. Also, the model with the extra term seem to perform slightly better than the model without the extra term for the 3 layer model.

The poor performance of the 4 layer model might be due to that the model doesn't update the model parameters in the neural network based on the performance of the evaluation of the inequality. The update is based on the output from the model i.e. the class prediction. The 3 layer model do update model parameters based on the modified Bell-inequality, however, by comparing the LHS of the inequality to the class label, which don't make sense. This might explain why it manages to detect some entangled states and why it is just a small amount.

Regarding the values of the probabilities $\{P_1, P_2, P_3, P_4\}$ it is notable from figure 9 and 10 that the individual probabilities don't converge to a single value. Instead they mostly take values either close to zero or one. One probability, P_4, differs from the others since the proportion of values close to zero is much larger than for the other probabilities. This could indicate that the P_4 is less relevant than the other probabilities, however, it is difficult to draw any certain conclusions since the models are inaccurate.

To be able to draw any certain conclusions regarding the usefulness of the modified Bell-inequality, the performance of the models need to be improved by either improving the models presented here or by changing strategy completely. Increased training time could perhaps improve these models performances somewhat, however, the training will be very time consuming and will probably not reach the level of accuracy needed to make any relevant conclusions. An alternative approach would be to formulate a neural network such that the four or five individual probabilities are tuned for the whole training data set rather than receiving a set of four or five probabilities for each set of data points the model are trained on. The difficulty is that the sum of the LHS expression in the modified Bell-inequality is unknown, so we don't have any data to train on for the Bell-inequality.

4 Further investigations using Bloch representation

The earlier parts of the project focused on how the different features that can be extracted from a density matrix are related to the entanglement of the state. This part will instead focus on finding the region of density matrices which represent entangled states through the use of a Monte Carlo method. In addition to this, we will use a more general representation of the quantum state in the Bloch representation.

4.1 Theory

A Monte Carlo method is a method that is based on repeated random generation.[9] Here, it is used to describe when a large number of density matrices on the from described below are generate to see which regions that represent entanglement.

The Bloch representation of a general two-qubit system is given by

$$\rho = \frac{1}{4}\left(I \otimes I + \vec{x}\cdot\vec{\sigma}\otimes I + I \otimes \vec{y}\cdot\vec{\sigma} + \sum_{n,m=1}^{3} T_{nm}\sigma_n \otimes \sigma_m\right)$$

and can be rewritten as

$$\tilde{\rho} = \frac{1}{4}\left(I \otimes I + \vec{a}\cdot\vec{\sigma}\otimes I + I \otimes \vec{b}\cdot\vec{\sigma} + \sum_{i} c_i\sigma_i \otimes \sigma_i\right), \tag{19}$$

where $\vec{a}$ and $\vec{b}$ are three dimensional vectors and c_i are the elements of a third three dimensional vector. If $\vec{a}$ and $\vec{b}$ are set to $\vec{0}$, the equation can be reduced to

$$\tilde{\rho} = \frac{1}{4}\left(I \otimes I + \sum_{i} c_i\sigma_i \otimes \sigma_i\right), \tag{20}$$

and written as a matrix representation on the form [10]

$$\tilde{\rho} = \frac{1}{4}\begin{pmatrix} 1+c_3 & 0 & 0 & c_1-c_2 \\ 0 & 1-c_3 & c_1+c_2 & 0 \\ 0 & c_1+c_2 & 1-c_3 & 0 \\ c_1-c_2 & 0 & 0 & 1+c_3 \end{pmatrix}. \tag{21}$$

Here, $c1$, $c2$ and $c3$ are constants with the conditions

$$\begin{gathered}|c_i| \leq 1 \quad \forall\, i \\ \sum_i c_i \leq 1 \\ c_1 - c_2 - c_3 \leq 1, \quad c_2 - c_1 - c_3 \leq 1, \quad c_3 - c_1 - c_2 \leq 1.\end{gathered} \tag{22}$$

In the earlier parts of the report, the labeling was done through the PPT criterion. This part however, will use a different measure of entanglement called concurrence. The state is referred to as maximally entangled when the concurrence is equal to one. The concurrence C of a state ρ is given by

$$C(\rho) = max[0, 2\lambda_{max}(\hat{\rho}) - tr\hat{\rho}] \tag{23}$$

where $\lambda_{max}(\hat{\rho})$ is given by the eigenvalues of the matrix

$$\hat{\rho} = \sqrt{\sqrt{\rho}\tilde{\rho}\sqrt{\rho}}, \quad \tilde{\rho} = (\sigma_2 \otimes \sigma_2)\bar{\rho}(\sigma_2 \otimes \sigma_2) \tag{24}$$

and $\bar{\rho}$ is given by the complex conjugate of ρ.[11]

When using the full version of the Bloch representation the input vectors $\vec{a}$, $\vec{b}$ and $\vec{c}$ follow the constraints

$$(a_1^2 + a_2^2 + a_3^2) + (b_1^2 + b_2^2 + b_3^2) + (c_1^2 + c_2^2 + c_3^2) \leq 3. \tag{25}$$

4.2 Method

Two data sets, one consisting of 50,000 bipartite quantum states and the other of 350,000 bipartite quantum states, were created in MATLAB based on the density matrix shown in Eq(21), where c_1, c_2, and c_3 are random values constrained by Eq(22). From this density matrix, the 3-by-3 matrix U was calculated (see section 2.1.1), and the eigenvalues of this matrix extracted and used as an input feature for the machine learning algorithm. As an output feature, the concurrence of the density matrix was calculated and used as a label to determine whether the state was entangled or separable.

The data set comprising 50,000 data points was used as a training set together with the seven data sets using the eigenvalues to the U-matrix as input features described in section 2.1.1 (Data Set X) on the models with 0, 10 and 50 nodes, while the data set with 350,000 quantum states (Data Set Y) was trained separately and exclusively on the models. All the trained models were then evaluated by using 350,000 data points generated from Eq(21) using a different random seed to obtain the models accuracy and plot c_1, c_2, and c_3 against each other to see which regions were entangled. Out of curiosity this evaluation data set was also used on our 50 node Model_ueig from section 2 for comparison.

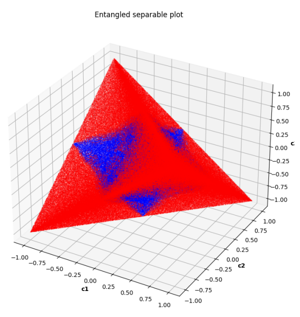
Following this, a data set of random vectors forming positive matrices $\tilde{\rho}$ on the form in equation 19 with the constraints in 25 was generated. The non-positive matrices where sorted out and the concurrence of the positive matrices was used to determine their entanglement. Each vector was then plotted with a color representation of whether they were part of an entangled state or not to investigate if any of the vectors were more important to the level of entanglement than the others.

4.3 Results

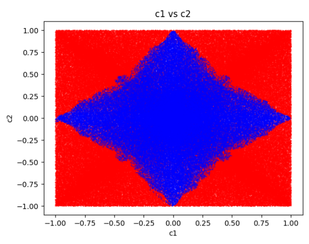
Table (5) shows the accuracy for each model of 0, 10, and 50 nodes, as well as for the data sets the models have been trained on. The models using Data Set Y as training set, have an overall higher accuracy than the respective models using Data Set X, where the highest accuracy reaches 99.625%. For both the training sets, the highest accuracy is obtained for the models using 50 nodes. In figure (11) and (12), scatter plots for the models using 50 nodes are presented to show the relationship between c1, c2 and c3, and the regions for which the quantum states are entangled (red color) or separable (blue color). The figures show that the region of the entangled states, as well as the region of separable states, forms a clear pattern where the entangled states surrounds the non-entangled states. When comparing the two-dimensional plots between the figures, one can see that the edges of the separable states-region are slightly more diffuse for the model using the Data Set X training set (Figure 11). The highest concurrence found was in the 50 node model trained on dataset Y, where a concurrence of 0.9970 was located close to one of the corners of the pyramid shape.

Training Set	Model	Learning Rate	Accuracy
Data Set X	0 nodes	0.01	83.282%
	10 nodes	0.001	96.420%
	50 nodes	0.001	96.557%
Data Set Y	0 nodes	0.01	90.296%
	10 nodes	0.01	99.564%
	50 nodes	0.001	99.625%

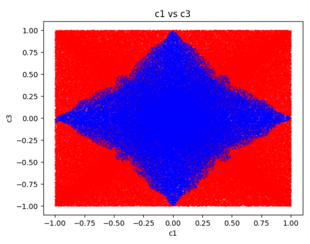
Table 5: Models trained on the quantum states generated from Eq (21), and their respective accuracy.



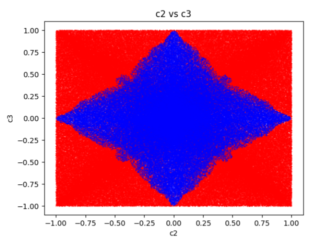
(a) The plot illustrates the relationship between c1, c2 and c3.



(b) The plot illustrates the relationship between c1 and c2.

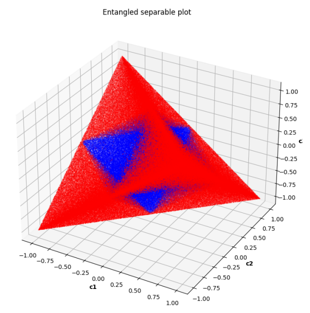


(c) The plot illustrates the relationship between c1 and c3.

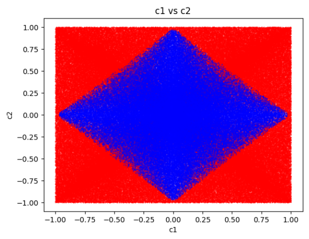


(d) The plot illustrates the relationship between c2 and c3.

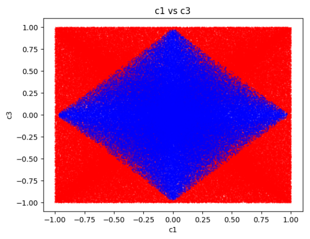
Figure 11: Scatter plots of entangled states (red) and separable states (blue). The model uses 50 nodes and is trained on Data Set X.



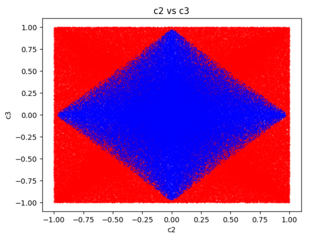
(a) The plot illustrates the relationship between c1, c2 and c3.



(b) The plot illustrates the relationship between c1 and c2.



(c) The plot illustrates the relationship between c1 and c3.



(d) The plot illustrates the relationship between c2 and c3.

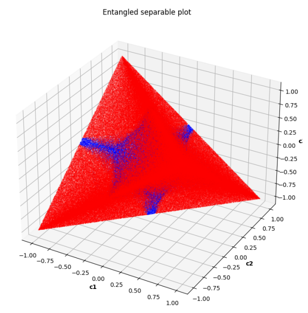
Figure 12: Scatter plots of entangled states (red) and separable states (blue). The model uses 50 nodes and is trained on Data Set Y.

The accuracy values and number of TE, FE, TS and FS for the 50 nodes models trained on the large training data set based on the eigenvalues to the matrix U from section 2, can be seen in table 6. The corresponding scatter plots for Model1_ueig are seen in figure 13. The average accuracy for these models are 86.763% which is a lower accuracy compared to the previously mentioned models in this part. This result is also visible in the scatter plots since the blue area (the states classified as separable) takes another shape compared to the scatter plots from more accurate models in figures 11 and 12.

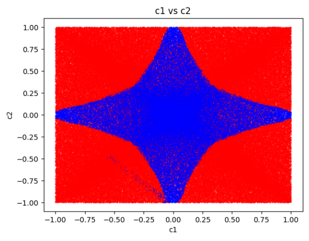
For comparison the scatter plots based on the data set used for testing in this part, are seen in figure 14. These plots demonstrates the shape the scatter plots should have if the models manages to predict with an accuracy of 100%.

Model	Accuracy	TE	FE	TS	FS
Model1_ueig	86.674%	240982	44554	62376	2088
Model2_ueig	86.251%	242301	47354	59576	769
Model3_ueig	87.364%	242361	43517	63413	709
Average	86.763%	241881	45142	61788	1189
std	0.562%	779	1985	1985	779

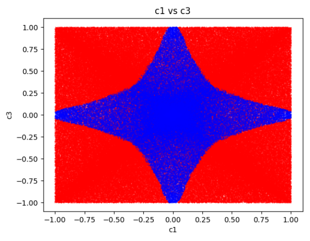
Table 6: Accuracy and number of TE, FE, TS and FS for 50 node models from section 2.



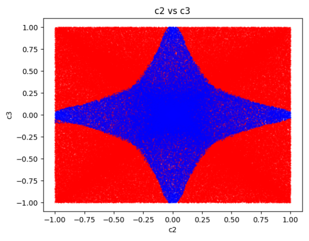
(a) The plot illustrates the relationship between c1, c2 and c3.



(b) The plot illustrates the relationship between c1 and c2.

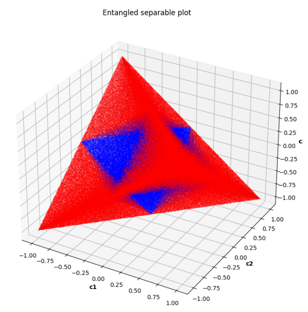


(c) The plot illustrates the relationship between c1 and c3.

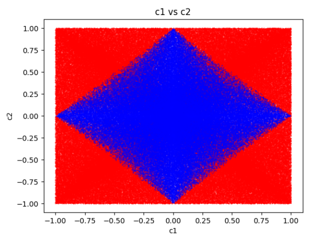


(d) The plot illustrates the relationship between c2 and c3.

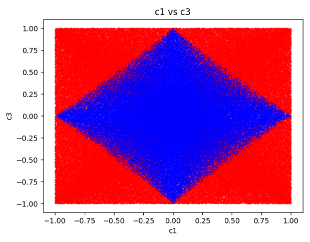
Figure 13: Scatter plots of entangled states (red) and separable states (blue). The model uses 50 nodes and is Model1_ueig from section 2.



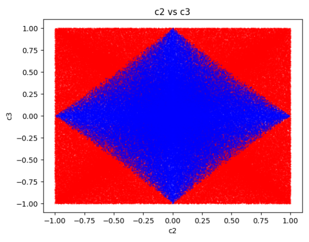
(a) The plot illustrates the relationship between c1, c2 and c3.



(b) The plot illustrates the relationship between c1 and c2.



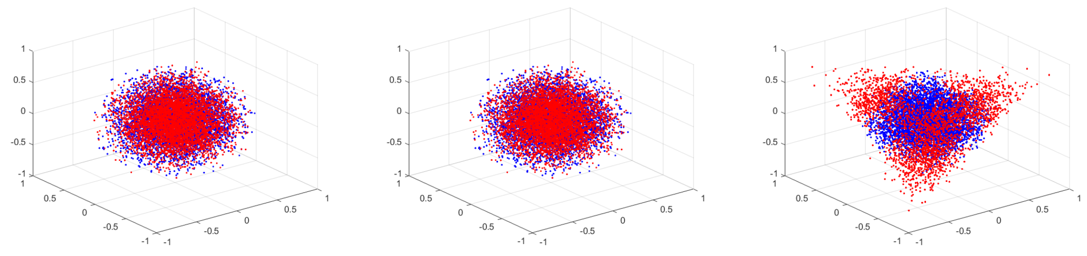
(c) The plot illustrates the relationship between c1 and c3.



(d) The plot illustrates the relationship between c2 and c3.

Figure 14: Scatter plots of entangled states (red) and separable states (blue) from the testing data set.

When plotting the randomly generated vectors it was hard to see a correlation between entanglement and the vectors $\vec{a}$ and $\vec{b}$ as seen in figure 15a and 15b. With $\vec{c}$ there is a more clear correlation and the figure also has a semblance with the results seen in 11a and 12a as seen in figure 15c. However, the highest concurrence found is 0.7907 which is lower than when only using $\vec{c}$.



(a) a_1 vs a_2 vs a_3 (b) b_1 vs b_2 vs b_3 (c) c_1 vs c_2 vs c_3

Figure 15: Red dots represent entangled states, blue dots represent separable states

4.4 Discussion

When comparing the results between the different training sets in Table (5), it is clear that when Data Set Y is used to train on the models, much higher accuracy can be achieved. This indicates that the method of using only the eigenvalues from the U matrix calculated from Eq (21) as input features is superior to the method when the seven data sets described in section (2.1.1) are mixed in. However, this might be because Data Set Y is more uniform compared to Data Set X, and does not give as much of a general solution as when using Data Set X as the training set. The old models, trained only on the large data set, are the most inaccurate models which could be due to that they are not trained on any data point corresponding to the data set generated from the Bloch representation in equation 20. This result can also be seen in the scatter plots. The scatter plots of the most accurate model are the most similar to the scatter plots corresponding to the testing data in figure 14, whereas the least accurate model differs the most from these plots. The highest level of entanglement was found to be located at the corners of the pyramid shape.

From Figure 11 and 12, it is evident that when the vectors $\vec{a}$ and $\vec{b}$ are set to $\vec{0}$, the model can still classify entangled states. Using the Monte Carlo approach, it was found that even if $\vec{a}$ and $\vec{b}$ do not seem to contribute to whether the state is entangled or not, they may affect the level of entanglement negatively.

5 Summary and Conclusions

This report has investigated and tested some of the uses machine learning has in the field of quantum technology. In the first part of the project the neural network was allowed to determine how to classify the input features on its own. This lead to a high accuracy when determining which states were entangled, something that the basic inequality was unable to do despite having the same inputs. This hints at there being some further correlation between relative entropy of coherence and entanglement beyond just the inequality as written.

This was investigated further in the second part where probability coefficients were added to the inequality. A high accuracy was not obtained, even though some hints were found suggesting that an inequality with different weighted values may be better at detecting entanglement than the basic inequality. This was both seen in that one of the parameters had a higher probability of being zero than the others and that using the probability coefficients lead to a slightly higher accuracy.

Finally, a Monte Carlo approach was applied to look at which regions of the vectors in the rewritten Bloch representation of the density matrices lead to entanglement. Here it was found that there was a separation between the regions that produce entangled states and the regions that don't. This, however, was only true for one of the three vectors while the other two seemed to only contribute to the level of entanglement.

In conclusion, this project has found that a machine learning model can be used to find entangled states with a high accuracy. However, making a model applicable in physics while still keeping its accuracy is harder. Making a machine learning model that both can detect entanglement with a high accuracy that is also more connected to physics may also require an entirely different approach.

References

[1] R. Horodecki, P. Horodecki, and M. Horodecki. “Violating Bell inequality by mixed spin-1/2 states: necessary and sufficient condition”. In: *Physics Letter. A* 200.5 (1995), pp. 340–344.

[2] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum information*. 10th Anniversary edition. Cambridge University Press, 2010. ISBN: 9781107002173.

[3] Naema Asif et al. “Entanglement detection using artificial neural networks”. In: *Scientific reports* 13.1.1562 (2023).

[4] J.J Sakurai and Jim Napolitano. *Modern Quantum Mechanics*. 3rd ed. Cambridge University Press, 2021. ISBN: 9781108473224.

[5] John S. Bell. “On the Eistein Podolsky Rosen Paradox”. In: *Physics, Vol. 1, No. 3.* (1964).

[6] Bohdan Horst, Karol Bartkiewicz, and Adam Miranowicz. “Two-qubit mixed states more entangled than pure states: Comparison of the relative entropy of entanglement for a given nonlocality”. In: *Physical Review A* 87 (2013).

[7] Michał Horodecki, Paweł Horodecki, and Ryszard Horodecki. “Separability of mixed states: necessary and sufficient conditions”. eng. In: *Physics letters. A* 223.1 (1996), pp. 1–8. ISSN: 0375-9601.

[8] Kiprono Elijah Koech. *The Basics of Neural Networks (Neural Network Series) — Part 1.* 2022. URL: `https://towardsdatascience.com/the-basics-of-neural-networks-neural-network-series-part-1-4419e343b2b` (visited on 01/13/2024).

[9] Steven C. Chapra. *Applied numerical methods with MATLAB for engineers and scientists*. McGraw-Hill Higher Education, 2012.

[10] Shunlong Luo. “Quantum discord for two-qubit systems”. eng. In: *Physical Review A* 77 (2008).

[11] Łukasz Derkacz and Lech Jakóbczyk. “Clauser-Horne-Shimony-Holt violation and the entropy-concurrence plane”. In: *Physical Review A* (2005).

Contributions

An overview of our individual contributions to this project will be described here. A more detailed description can be found in our status reports. The contributions are divided into the parts of the project report for simplicity. Note that we all have contributed with ideas and solutions to problems in most of the parts.

First part

In the first part, Julia did the data generation part except for a few Matlab functions which Tim wrote. Tim and Isabella worked with and implemented the neural network. We all did runs of the code and similar to receive results, test ideas/strategies and for analyzes.

Second part

In the second part, Isabella did most of the implementation of the machine learning model with contributions from Tim. We all discussed the results and what improvements could be done.

Third part

In the third part, Julia did analyze when looking at only the c-values and Tim the part analyzing all parameters; a, b and c.

Report

In the report, the introduction, abstract and the summary and conclusion were mainly written by Tim. In the first part, "Implementation using a shallow neural network", Julia and Isabella made the largest contributions and in the second part, "Neural network and the Bell-inequality" were mainly written by Isabella. In the third part, "Further investigations using Bloch representation", Julia and Tim made the largest contributions. Tim has also made the figures visualizing the different neural networks presented throughout the report.