Teaching a Neural Network Quantum Mechanics.

A Deep Learning Approach to the N-Representability Problem.

Emil Johansson

Submitted for the degree of Master of Science.

Department of Physics at Lund University.

Course: FYSM60 (60 ECTS points)

***Supervisors*: Mats-Erik Pistol, Gillis Carlsson & Najmeh Abiri**

Submitted: May 7, 2018

Examination: May 30, 2018

LUND UNIVERSITY

Faculty of Science


THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS
https://xkcd.com/1838

Abstract

We show that an unsupervised artificial neural network can be trained to parameterize the set of N representable density matrices well enough to enable ground state energy calculations. A one-dimensional harmonic oscillator system is used to test the method. 4, 5, or 6 fermions are placed in an external potential. They interact with one of three different interaction types. By choosing the most successful network according to a well-defined measure, the approach is shown to generalize to interaction types not considered by the measure.

Acknowledgements

I want to thank my supervisors Mats-Erik Pistol, Gillis Carlsson and Najmeh Abiri for their support and interest during this diploma work process. I want to thank NanoLund for funding making it possible to run the calculations in the cloud and Johannes Bjerlin for providing C.I reference calculations. I also want to thank my fellow diploma work students in B120 for being an excellent sounding board and for their friendship. Last but not least I want to send a special thanks to Desiré Nilsson for being in my life.

Abbreviations

2-RDM Two-particle reduced density matrix (also denoted with Γ)

$P(N)$ Set of N representable two-particle density matrices (set of all 2-RDMS)

AE Autoencoder

ANN Artificial Neural Network

Artificial Neural Network terminology

Deep Learning Using ANNs with many layers.

Weights Free parameters of ANNs.

Loss Measure of distance between output and desired output.

Training Adjusting weights to minimize loss.

Adversarial Examples Examples engineered explicitly to produce erroneous network behavior.

Contents

1 Introduction **5**

2 Theory **8**

2.1 N Representability 8

2.2 Artificial Neural Networks 15

3 Method **18**

3.1 Parametrization 18

3.2 Artificial Neural Networks 20

3.2.1 Discriminator 20

3.2.2 Autoencoder 24

3.3 Generating data 26

3.4 On not reinventing the wheel 28

3.5 A note about all the free parameters 29

4 Results **31**

4.1 Discriminator 31

4.2 Autoencoder 35

5 Conclusion **39**

6 Prospects **40**

A The TANDEM algorithm **45**

A.1 Example 46

B ANN approach to combining parametrization conditions **50**

C Network Details **54**

1 Introduction

Quantum mechanics is a remarkably successful theory. It is also a weird theory. Niels Bohr in conversation with Werner Heisenberg claimed that "Anyone who is not shocked by quantum theory has not understood it"[1]. However, time and time again, the predictions are observed in experiments. So we are in a tricky spot. There is an excellent theory describing the world of small objects and their interactions, but it does not make sense to us. What if we can create artificial intelligence for which quantum mechanics is reasonable? In its most broad and abstract form, the goal of this work is to be the first step towards an AI with intuition for quantum mechanics.

Philosophical problems with quantum mechanics are often "solved" with a "shut up and calculate" approach. The idea is to not care about the philosophical implications. Do not bother to understand it. Just make sure your math is correct. But what to do when the math is too hard? Only a handful of systems, like the hydrogen atom, can be solved exactly with pen and paper. With the help of computers, approximate solutions are possible for a range of small systems. But the time it takes to do the computation increases exponentially with the number of interacting particles. Making predictions with quantum mechanics is a tough challenge. There are many approaches to do many-body quantum mechanics. For example, density functional theory [2, 3] and Green's functions [4]. This work is trying to provide the first step in the creation of a new approach within the reduced density matrix branch [5].

The reduced density matrix formalism draws from the insight that the wave function tells us more than we need to know [6]. A two-particle reduced density matrix (2-RDM) is the reduction of an N-particle state to an ensemble of 2 particle states. The formalism replaces the physical significance of the wave function with the reduced density matrix. Using a reduced density matrix formulation seemingly solved the exponential increase in computation time with the number of interacting particles by flattening the space in which we need to search for a solution. However, the catch is that this smaller set of possible solutions (referred to as $P(N)$) is hard to search over. It is a formidable challenge to exclude all elements that cannot be physical

solutions from the search. This challenge is called the N representability problem.

In sharp contrast to quantum mechanical reformulations is the hot buzz-word field deep learning. The term deep learning has appeared in recent years as a term for many-layer artificial neural networks (ANN). The idea to mimic the way the human brain learns in a computer dates back to 1949 when Hebb first proposed that the brain learns by changing the strength of connections between neurons [7, 8].

Recent achievements in training ANNs to do image recognition [9], playing games [10], and translation [11] have dramatically improved the state-of-the-art in said fields. Attempts to utilize ANNs in physics have been made both in experimental settings [12] and theoretical calculations [13]. However, the role of ANNs in physics and their importance is still an open question. Partly this is because recent advancement in ANNs are, well, recent. Further, there is no consensus regarding why ANNs are so useful.

The initial approach of the project was to use a classifier network to discriminate between density matrices as either in $P(N)$ or not. The idea was to use the discriminator to exclude elements not in $P(N)$ from the search. Interestingly, creating an ANN that manages to classify matrices as either in $P(N)$ or not correctly was not the problem. Instead, adversarial examples [14] stopped the discriminator approach.

The adversarial examples and the general instability of the discriminator approach sent us back to the drawing board. ANNs work very well in many applications [8]. However, there is no consensus *why* they work so well. An often cited reason is that a feed-forward ANN can approximate any function given enough neurons [15]. Even though it is a significant result, the space of all functions is so large that all reasonably sized ANNs still only cover a small part of the space of all function. The question of why networks work so well can be rephrased as, why the subset of all functions that ANNs efficiently approximate hold so many practical uses.

Some argue that that physics can explain the success of ANNs. More precisely that ANNs efficiently incorporates mathematical properties often found in theoretical physics, such as symmetry, locality, compositionality and hierarchical structure [16].

This explanation is music to my ears, inciting hope that unsupervised learning might on its own uncover vital physical insights into the N-Representability problem.

Further, one can also argue that the power of ANNs lies in their ability to create a high-level description of the data. A representation is effectively a parametrization of the data set. Using unsupervised learning, we train a network to develop a parametrization of the set of N representable density matrices. This parametrization can then be used to solve the constrained optimization problem of finding the ground state. Our results show that autoencoder (AE) ANNs can find parametrizations good enough to do energy calculations as seen in figure 10 in section 4.2.

2 Theory

In the theory section 2.1 we will define the set of two particle reduced density matrices $P(N)$ and show their importance. Section 2.2 explain the basics of ANNs and introduce dense and convolutional layers.

2.1 N Representability

Before we get started, there is the question of notation. I will use second-quantization formalism and mostly follow *Variational approach to electronic structure calculations on second-order reduced density matrices and the N representability problem* by M. Nakata et al. [17]. Section two goes through much of what I will cover in the beginning of this section. [18] also quickly introduced the N representability problem using second quantization and bra-ket notation in a similar but more condensed way as here.

The second quantization formalism uses a set of creation and annihilation operators $\{a_i, a_i^\dagger\}_{i=1}^M$ where $a_i^\dagger$ creates and a_i annihilates a particle in the one-particle state ψ_i. M is the number of one-particle wave functions used as a basis. It is noteworthy that no result in this section is dependent on a specific choice of one-particle states $\{\psi_i\}$. The choice of $\{\psi_i\}$ determines the subset of the full Hilbert space that is considered.

Second quantization operators commute according to their statistics. We will only be considering fermions although the method is extendable to bosons as well. The commutation relations for the second quantization operators for fermions are $\{a_i, a_j\} = \{a_i^\dagger, a_j^\dagger\} = 0$ and $\{a_i^\dagger, a_j\} = \delta_{ij}$. Second quantization is convenient as it makes creating anti-symmetric combinations of one particle wave functions as easy as applying creation operators to *the* 0 particle state $|0\rangle$.

All operators can be expressed in terms of second quantization operators. For any system with only one- and two-particle interactions the Hamiltonian can be written as:

$$\hat{H} = \sum_{ij} v_j^i a_i^\dagger a_j + \sum_{ijkl} \omega_{kl}^{ij} a_i^\dagger a_j^\dagger a_k a_l \tag{1}$$

v_j^i in equation 1 are the coefficients determining one-particle interaction and ω_{kl}^{ij} the two-particle interaction [17]. For three-particle interactions, one would need to add terms on the form $a_i^\dagger a_j^\dagger a_k^\dagger a_l a_n a_m$. We can get v_j^i and ω_{kl}^{ij} from the position space-representation as follows:

$$v_j^i = \int \psi_i^*(z) \left(-\frac{1}{2}\nabla^2 - V(z) \right) \psi_j(z) dz \tag{2}$$

$$\omega_{kl}^{ij} = \int \psi_i^*(z_1)\psi_j^*(z_2) I(z_1, z_2) \psi_l(z_1)\psi_k(z_2) dz_1 dz_2 \tag{3}$$

$V(z)$ is the external potential. The results in section 4 use the Harmonic oscillator external potential $V(z) = \frac{1}{2}m\omega^2 z$. $I(z_1, z_2)$ is the interaction term. We will use three different interaction terms $I(x_1, x_2) = \delta(z_1 - z_2), e^{\frac{(z_1-z_2)^2}{0.5}}, e^{\frac{(z_1-z_2)^2}{1.0}}$. $\psi_i(z)$ are the space-representations of our one-particle wave functions basis. That is: $\psi_i(z) = \langle z | \, a_i^\dagger \, | 0 \rangle$.

By noting that $\sum_k a_k^\dagger a_k = \hat{N}$ we can for a space of only N-particle states get the identity $a_i^\dagger a_j = \frac{1}{N-1} \sum_k a_i^\dagger a_k^\dagger a_k a_j$. Combining this identity with equation 1 for the Hamiltonian, the exact expectation value can be rewritten from wave function form to the 2-RDM form. Γ is the 2-RDM. In the reduced density matrix formalism, the 2-RDM takes over the physical significance of the wave function.

$$
\begin{aligned}
E = \langle\Psi|\,\hat{H}\,|\Psi\rangle &= \sum_{ij} v^i_j \,\langle\Psi|\, a^\dagger_i a_j \,|\Psi\rangle + \sum_{ijkl} \omega^{ij}_{kl}\,\langle\Psi|\, a^\dagger_i a^\dagger_j a_k a_l \,|\Psi\rangle \\
&= \sum_{ijk} \frac{v^i_j}{N-1}\,\langle\Psi|\, a^\dagger_i a^\dagger_k a_k a_j \,|\Psi\rangle + \sum_{ijkl} \omega^{ij}_{kl}\,\langle\Psi|\, a^\dagger_i a^\dagger_j a_k a_l \,|\Psi\rangle \\
&= \sum_{ijkl} \frac{v^i_j \delta_{kl}}{N-1}\,\langle\Psi|\, a^\dagger_i a^\dagger_k a_l a_j \,|\Psi\rangle + \sum_{ijkl} \omega^{ij}_{kl}\,\langle\Psi|\, a^\dagger_i a^\dagger_j a_k a_l \,|\Psi\rangle \\
&\text{Do dummy index changes: } k \to j, \quad j \to l, \quad l \to k \\
&= \sum_{ijkl} \frac{v^i_l \delta_{jk}}{N-1}\,\langle\Psi|\, a^\dagger_i a^\dagger_j a_k a_l \,|\Psi\rangle + \sum_{ijkl} \omega^{ij}_{kl}\,\langle\Psi|\, a^\dagger_i a^\dagger_j a_k a_l \,|\Psi\rangle \\
&= \sum_{ijkl} \left(\frac{v^i_l \delta_{jk}}{N-1} + \omega^{ij}_{kl} \right) \langle\Psi|\, a^\dagger_i a^\dagger_j a_k a_l \,|\Psi\rangle \\
H_{ijlk} \equiv \frac{v^i_l \delta_{jk}}{N-1} + \omega^{ij}_{kl}, &\quad \Gamma^{ij}_{lk} \equiv \langle\Psi|\, a^\dagger_i a^\dagger_j a_k a_l \,|\Psi\rangle \qquad (4) \\
E = \sum_{ijlk} H_{ijlk} \Gamma^{ij}_{lk} &\equiv E_H(\Gamma) \qquad (5)
\end{aligned}
$$

At first glance, this might seem only like a cumbersome renaming. However, notice that the degrees of freedom of Ψ are $\mathcal{O}(\binom{M}{N})$ and $\mathcal{O}(M^N)$ if $M \gg N$, where M is the number of one-particle basis states and N is the number of particles. On the other hand the degrees of freedom of Γ is M^4 from the four indices that range from 1 to M. From the definition of Γ^{ij}_{kl} in equation 4 it is clear that Γ^{ij}_{kl} will have many (anti-)symmetries. The number of free variables in the 2-RDM after considering the symmetries is $\binom{\binom{M}{2}+1}{2} = \mathcal{O}(M^4)$.

$$
\begin{aligned}
\Gamma^{ij}_{lk} = -\Gamma^{ji}_{lk} = -\Gamma^{ij}_{kl} = \Gamma^{ji}_{kl} \qquad (6) \\
\Gamma^{lk}_{ij} = -\Gamma^{lk}_{ji} = -\Gamma^{kl}_{ij} = \Gamma^{kl}_{ji}
\end{aligned}
$$

The same procedure as in equation 5 above for the Hamiltonian can be carried out with all one- or two-particle interaction observables. This generality means other

expectation values can be calculated from the 2-RDM as well. The mapping from the full N-particle state space to the reduced density matrix space is however not one-to-one. As the mapping is not one-to-on more than one wave function is mapped to the same density matrix. This loss of information about the original wave function is in line with the idea that the entire N-particle wave function tells us more than we need to know.

So what do we need to know, and what do we forget? Remember the first assumption we made about the Hamiltonian, that it can have at most two-particle interactions. Two-particle interaction is the only constraint for which operators we keep the expectation values exact. We have explicitly created the reduction so that the expectation value of relevant observables are conserved.

Let's consider the set of all operators that can be written with two-particle interaction as in equation 1 and call it the *two-operator set* O. The lowest eigenvalues of two operators are essential, after all, it is those eigenvalues we aim to calculate. wave functions that are non-degenerate eigenvectors with the smallest eigenvalue of the operator are special. For an operator with a non-degenerate ground state, there is only one wave function that gives the expectation value equal to the lowest eigenvalue. Thus non-degenerate ground state wave functions have a one-to-one mapping to their 2-RDM.

$$\langle\psi|\, a_i^\dagger a_j^\dagger a_k a_l \,|\psi\rangle = \langle\Phi|\, a_i^\dagger a_j^\dagger a_k a_l \,|\Phi\rangle \Leftrightarrow \langle\psi|\, \hat{A} \,|\psi\rangle = \langle\Phi|\, \hat{A} \,|\Phi\rangle \;\; \forall \hat{A} \in O \tag{7}$$

Equation 7 tells us that if we only have measurements corresponding to operators expressible in two-particle interactions, there is no way we can distinguish between states represented with the same 2-RDM by their expectation values. It is remarkable for an exponential reduction free parameters retaining so much relevant information.

The exponential improvement the in number of free variables as a function of particle number makes variational methods for many-body problems more feasible. Traditional variational methods approximate the ground state by minimizing the energy expectation value over the N-particle Hilbert space $\mathcal{H}^{\otimes N} = \mathcal{H} \otimes \mathcal{H} \otimes \mathcal{H} \ldots$.

In practice, a subspace or even just a subset of $\mathcal{H}^{\otimes N}$ is used. In the same way, we can construct a variational search among all possible Γ

$$E_g = \min_{\Psi \in \mathcal{H}^{\otimes N}} \langle \Psi | \hat{H} | \Psi \rangle = \min_{\Gamma \in P(N)} E_H(\Gamma). \tag{8}$$

Equation 8 reformulate the problem of calculating ground state energies to finding an efficient algorithm for the constrained optimization over the set of all 2-RDMs. Unfortunately, not all two-particle density matrices are the representation of some N particle state Ψ. The density matrices that are representations of N particle states are called N representable. The set of N representable density matrices (also named two-particle reduced density matrices) is denoted $P(N)$.

$$P(N) = \{\Gamma \mid \exists \Psi \in \mathcal{H}^{\otimes N} \mid \Gamma^{ij}_{lk} = \langle \Psi | a_i^\dagger a_j^\dagger a_k a_l | \Psi \rangle\} \tag{9}$$

Writing down the definition of $P(N)$ poses no problem and is done in equation 9. The tricky part is to create an efficient search algorithm. The problem to characterize $P(N)$ in such a way is called the N representability problem. Garrod and Percus [19] showed that $P(N)$ can be characterized by the fact a two-particle density matrix is in $P(N)$ if and only if for every Hamiltonian the expectation value is larger than the ground state energy for the corresponding Hamiltonian.

$$P(N) = \{\Gamma \mid E_H(\Gamma) \geq E_g(H) \text{ for all two-particles Hamiltonians } H\}$$

The ground state energy characterization is not practical since it requires all ground state energies to be known. However, the characterization can be useful in identifying examples of elements that are *not* in $P(N)$. Some necessary, efficient criteria have been discovered but no complete, practical solution to the N representability problem is known.Unfortunately, the N representability problem is in the NP-complete complexity class [18]. The classification as NP-compete means that finding a full and efficient solution is virtually impossible as that would solve the famous P vs NP problem

in computational complexity theory. However, that does not stop us from finding a good approximate solution.

Before we attack the NP-hard core of the N representability problem with ANNs, some conditions can be implemented explicitly. Shrinking the space from $\mathbb{R}^{M\times M\times M\times M}$ to the set of density matrices will make the job of the networks simpler.

First, we transform the 2-RDM $\Gamma^{ij}_{kl} \quad 1 \leq i < j \leq M;\ 1 \leq k < l \leq M$ to a matrix form $\Gamma_{j-i+(2M-i)(i-1)/2,l-k+(2M-k)(k-1)/2}$ that is more convenient when doing linear algebra [17]. Then we turn our attention to the fact that we want Γ to be a density matrix. Theorem 2.5 in section 2.4 of [20] aptly named Characterization of density operators specifies a set of necessary and sufficient demands on a density operator.

Characterization of density operators. *An operator ρ is the density operator associated with some ensemble $\{p_i, |\psi_i\rangle\}$ if and only if it satisfies the conditions:*

1. *(***Trace condition***) ρ has trace equal to one.*

2. *(***Positivity condition***) ρ is a positive operator.*

Note that the trace of Γ is not 1 but $N(N-1)$. The value of the trace differ between conventions, but what is important is not the value (that can always be changed by multiplying a scalar) but that it is fixed. The physical significance of conserving the trace is that of conserving the particle number. The ideas behind positive eigenvalues are a bit more involved.

A density matrix, in general, represents a statistical ensemble of pure states ψ_i with different probabilities p_i and can be written as $\sum_i p_i |\psi_i\rangle \langle\psi_i|$. However, the set of pure states and probabilities that form a density matrix is not unique. More than one set of wave functions and probabilities combine to form the same density matrix. For any density matrix, one way to get such a set is to diagonalise the matrix. The eigenvectors are the states and the eigenvalues the probabilities. As probabilities must be non-negative, the eigenvalues must be non-negative. So we see that the positivity conditions comes from the fact that only positive, or zero, probabilities are meaningful.

In summary, to search the space of two-particle density matrices, we need a way to make Γ satisfy the symmetries in equation 6, trace $N(N-1)$, and positive definiteness.

All two-particle reduced density matrices are (unsurprisingly) density matrices. However, one might wrongly assume that all two-particle density matrices are reduced N particle density matrices, as unfortunately, that is not the case. A.J Coleman discovered this when he in 1951 did variational calculations over the set of two-particle density matrices for the ground state of lithium. Variational methods typically guarantee that the ground state energy obtained is *above* the actual ground state energy. However, Coleman got an energy 20% *below* the ground state energy [21].

This concludes the necessary theory from a physics perspective. The next section will bring us up to speed on the primary tool that we use to achieve our goal of an efficient algorithm to search $P(N)$.

2.2 Artificial Neural Networks

A feed-forward artificial neural network (ANN) is a function taking a vector input and returning a vector or scalar output. An essential element is that ANNs also depend on a large number of free parameters called the weights. The number of weights can be in the millions [9]. The loss is a measure of the difference between the actual output and the desired output. The value of each weight is determined by minimizing the loss with respect to the weights. Gradient descent is often used as the optimization algorithm.

$$
\begin{aligned}
y &= f\left(\begin{bmatrix}\theta_{h1} & \theta_{h2}\end{bmatrix} f\left(\begin{bmatrix}\theta_{11} & \theta_{21}\\ \theta_{12} & \theta_{22}\end{bmatrix}\begin{bmatrix}X_1\\ X_2\end{bmatrix}\right)\right) \qquad (10)\\
&= f(\theta_{h1} f(\theta_{11}X_1 + \theta_{21}X_2) + \theta_{h2} f(\theta_{12}X_1 + \theta_{22}X_2))
\end{aligned}
$$

The terminology introduced so far for ANNs fits just as well to linear regression and in a sense, ANNs are nothing more than linear regression on steroids. ANNs can bee seen as consisting of multiple linear regressions, but with nonlinear functions in between each layer of linear regression.

Another way to see ANNs is as weighted directed computational graphs. Graphs consist of nodes (also called points or vertices) connected by lines (also called arcs or edges). A graph can be directed, meaning that each line is an arrow with a direction from one node to another. If an arrow exist from x to y, x is a direct predecessor to y and y is a direct successor of x. A weighted graph has a real number weight associated with each line in the graph.

The computation in computational graphs is the process of assigning a value to each node in the graph. Input nodes are nodes without incoming lines (no direct predecessors) and are assigned the values of the inputs to the network. For all other nodes, the value of the node is calculated as the weighted sum of the values of direct predecessors passed as argument to an activation function. Nodes without successors (outgoing lines) are output nodes and constitutes the output of the network.

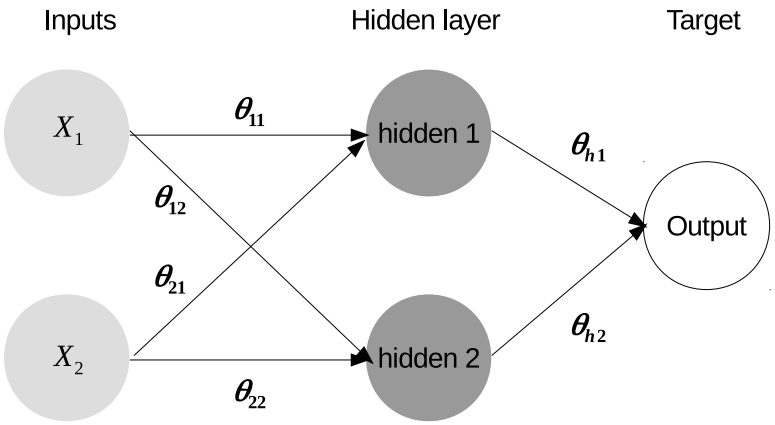

Figure 1: Graph representation of a neural network. The argument passed to the network is set as the value of the input nodes X_1 and X_2 with no incoming lines. The value of a node, multiplied by the line weight is feed as input to the next layer nodes. For example X_1 send $X_1\theta_{11}$ to the "hidden 1" node. The value of the hidden nodes are calculated as the sum of all inputs passed as argument to an activation function f. For example, the value of "hidden 1" becomes $f(\theta_{11}X_1 + \theta_{21}X_2)$ and will in turn be multiplied by θ_{h1} and passed as input to the next layer that happen to be the output layer. The output layer nodes have no outgoing lines and their values are the resulting output of the network. Equation 10 writes down the function representation of the neural network depicted in this graph.

The configuration of nodes and lines creating the graph representation of an ANN is called the architecture of the ANN. A common architectural property is to divide the nodes into ordered layers. The first layer is the input layer and the last layer is the output layer. A neuron only have direct predecessors in the layer before it and direct successors in the next layer. Networks of this type are called *feed-forward neural networks* .

If all nodes in a layer are connected with all nodes in the next layer, the layer is said to be dense. This dense connectivity is not always desirable due to the high number of weights. To many weights can cause over fitting, where the network memorizes the examples instead of understanding them. The task of finding optimal weights also becomes harder, and finally all weights might not fit in memory at some point.

Convolutional neural networks reduce the number of weights in two ways. Firstly, nodes only have direct predecessors from a small part of the previous layer. Secondly, weights are shared between lines. Deep (many layers) convolutional neural networks have had great success in image recognition tasks [9].

3 Method

The theory section left us with a constrained optimization problem formulation of many body quantum mechanics and an ANN toolbox. The aim of the method section is to transform the constrained optimization problem into a function approximation problem, solvable with our ANN toolbox. Before we hand over control to the ANN, we need to check what conditions we can incorporate ourselves explicitly. As derived in section 2.1, there are some conditions on Γ we know we need to satisfy.

3.1 Parametrization

Before the ANN is involved, we make sure that the conditions of (anti-)symmetry, fixed trace, and positivity are met. These demands are met with the help of parametrization. The parametrization makes it possible for the network to only worry about the problems that we cannot solve with explicit parametrization.

The most challenging condition that we want to parameterize is the positivity condition. As discussed in the theory section the 2-RDM is a positive semi-definite, symmetric matrix. Fortunately, we are not the first to crave a parametrization for symmetric, positive-definite matrices [24]. J. C. Pinheiro and D. M. Bates from the departments of biostatistics and statistics at the University of Wisconsin-Madison published a paper in 1996 where they modified the Cholesky decomposition to be a parametrization of symmetric positive-definite matrices suitable for minimization.

Any positive definite matrix A can be decomposed into $A = LL^\dagger$ where L is a lower triangular matrix called the Cholesky decomposition of A. If the elements on the diagonal are positive, then L is unique [24]. Using the logarithm of the diagonal elements in L as parameters produces a unique parametrization. To get a more clear picture of how this parametrization works, we show next the procedure to get from a list of parameters specifying the two particle density matrix $(c_1, c_2, c_3, c_4, c_5, c_6)$ to a general positive semi-definite symmetric 3×3 density matrix.

$$(c_1, c_2, c_3, c_4, c_5, c_6) \rightarrow L = \begin{bmatrix} e^{c_1} & 0 & 0 \\ c_2 & e^{c_4} & 0 \\ c_3 & c_5 & e^{c_6} \end{bmatrix}$$

$$A = LL^\dagger = \begin{bmatrix} e^{2c_1} & c_2 e^{c_1} & c_3 e^{c_1} \\ c_2 e^{c_1} & c_2^2 + e^{2c_4} & c_2 c_3 + c_5 e^{c_4} \\ c_3 e^{c_1} & c_2 c_3 + c_5 e^{c_4} & c_3^2 + c_5^2 + e^{2c_6} \end{bmatrix}$$

$$\mathrm{Tr}[A] = e^{2c_1} + c_2^2 + e^{2c_4} + c_3^2 + c_5^2 + e^{2c_6}$$

The log-Cholesky parametrization only works for positive-definite matrices [24]. The eigenvalues of a positive-definite matrix is always greater than zero. However, in the case of density matrices, as zeros occur as a probabilities, eigenvalues can be zero. Fortunately, any positive-semi-definite matrix can be approximated arbitrarily close by a positive-definite matrix.

Parametrization of the index (anti-)symmetries is conceptually trivial. Index (anti-)symmetries as in equation 6 dictate direct relationships between pairs of elements. Any set of linearly independent tensor elements from which the 2-RDM can be fully specified constitutes a proper parametrization. This type of parametrization can be done for all symmetries in equation 6. It is not necessary to impose all symmetries in equation 6 in this way as the log-Cholesky parametrization technique for imposing positivity at the same time take care of the symmetry $\Gamma^{ij}_{kl} = \Gamma^{kl}_{ij}$.

To practically implement the transformation between the index symmetry parametrization and the full tensor is however not trivial. It requires constructing a transformation tensor with elements with values 0, 1 or -1 that specify the mapping between the parametrization and the full rank four tensor.

Fixing the trace of the operator Γ to $N(N-1)$ is achieved by multiplying the operator with $\frac{N(N-1)}{Tr[\Gamma]}$. This brings up the special case of $Tr[\Gamma] = 0$. Luckily $Tr[\Gamma] \neq 0$ as Γ is positive semi-definite. The non-zero trace is evident from the fact that the trace is the sum of all (in this case positive) eigenvalues.

Why stop here? From Γ^{ij}_{kl} one can get more matrices than $\Gamma_{j-i+(2M-i)(i-1)/2,l-k+(2M-k)(k-1)/2}$ that need to be positive. These matrices are often called Q, G, T_1, and T_2 [17]. Our parametrization scheme transformed Γ such that it is positive semi-definite. Then from the parametrization, we calculate Γ^{ij}_{kl} as made possible by the one to one mapping between them. We could do the same thing for the other matrices, as long as the transformations are invertible, but there is no clear way how to do it for more than one condition at a time. Incorporating our knowledge of other conditions into the design of the network is probably a more rewarding approach. Such networks are not explored in this work. However, appendix B describes how knowledge of multiple parametrizable conditions can be tightly incorporated into the network, given the existence of inverse transformations.

3.2 Artificial Neural Networks

The problem we want to use neural networks for is to solve the constrained optimization problem in equation 8. A picture that might help to make sense of our search for ANNs is to make an analogy to mining. In a sense we are mining the space of all computable functions that are efficiently approximated by ANNs. A space that has turned out to include many useful functions. The following two subsections will deal with methods to extract valuable functions from this space. But before starting the search, we must know what functions we are looking for. I will show two different ways to transform the problem of constrained minimization to the search for a function.

3.2.1 Discriminator

The initial approach (pursued with limited success) use a discriminator $D : \mathbb{R}^n \to [0, 1]$. D multiplied with γ forms a punishment term with max size γ. The idea is that the

new term will force the minimization to stay in the set $P(N) \subset \mathbb{R}^n$

$$\left.\begin{aligned} D(x) &= \begin{cases} 0 \text{ if } x \in P(N) \\ 1 \text{ if } x \notin P(N) \end{cases} \\ \gamma &> \min_{x \in P(N)} E(x) - \min_{x \in \mathbb{R}^n} E(x) \end{aligned}\right\} \Rightarrow \min_{x \in P(N)} E(x) = \min_{x \in \mathbb{R}^n} \left[E(x) + \gamma D(x)^2\right] \tag{11}$$

The value of $\min_{x \in P(N)} E(x)$ in equation 11 is not known beforehand, so there is no way to determine how large γ need to be. However, by checking the output of D on the result, it can be determined if γ was too small.

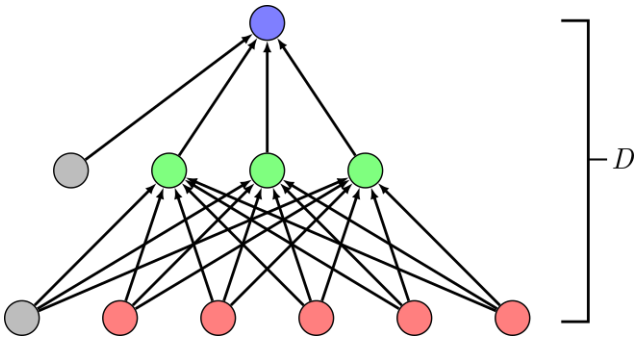

Figure 2: The figure depicts an illustration of a simple feed-forward artificial neural network classifier. Examples are feed in as input at the bottom (red). Weights multiply the inputs (represented by the lines), forming the input to the next layer (green). The activation function is applied generating the output of the green layer. The process is then done one more time to produce a single output neuron (blue). In both layers, there is also a bias node (grey) that always gives the same value independent of input. The entire network (with fixed weights) is used as the discriminator D. Figure from [22]

The network needs one input node for every element in our log-Cholesky parametrization and a single output node. Figure 2 shows an illustration of such a network. To assure the network output is between zero and one the activation function for the output layer is a sigmoid function $S(x) = \frac{1}{1+e^{-x}}$. The Binary cross entropy below is used as loss function.

$$\text{loss}(y_{\text{pred}}, y_{\text{target}}) = -(1 - y_{\text{target}}) \log(1 - y_{\text{pred}}) - y_{\text{target}} \log(y_{\text{pred}}).$$

y_{target} is the desired target output as defined by equation 11. y_{pred} is the networks output. The network is trained on pairs of log-Cholesky parametrizations and targets. The positive examples (with $y_{\text{target}} = 1$) are generated by reducing N particle wave functions with equation 4. Negative examples are generated by randomly sampling log-Cholesky parametrizations and finding two-particle density matrices that have an energy lower than the ground state energy. Note that the process of generating negative examples requires known ground state energies making the method empirical.

An unfortunate feature of ANNs doing classification is that they are very susceptible to adversarial examples. Adversarial examples are input to the classifier that only differ just so slightly from a correctly classified example but are explicitly created to become wrongly classified. An example from image recognition is the adversarial images to a binary car classifier in figure 3.

(a) (b)

Figure 3: Figure from [14]. The figure shows adversarial examples to a binary car classifier. In both a) and b) the randomly chosen example to the left is classified correctly as a car, but the car in the middle is not. The image to the right is the difference between the two images.

A way to generate adversarial examples is by the *fast gradient sign method* [25], that use the gradient to find the steepest path away from the correct classification. If gradient descent is used for optimization over a classifier (as we have done) the problem of adversarial examples will be significant. The minimization algorithm will most likely end up in a wrongly classified density matrix that has to low energy.

To remedy the effect of adversarial examples, we add adversarial examples to the training set. Again, generating adversarial examples requires Hamiltonians with known ground states.

The idea is to train the network to classify density matrices correctly by training on its own mistakes. With the classifier, the optimization in equation 11 is now possible. If the answer is lower than the known ground state energy, the "solution" is added to the training set, and labeled with 0 (not N representable). To balance the training set, a positive example is also added. After doing this procedure for finding wrongly categorized examples for all training Hamiltonians, the network trains with the expanded training set.

The solution found by minimizing the effective energy can also end up with an energy that is higher than the ground state energy. In this case, the convex property of the set of N representable density matrices comes in handy. The solution will always be a pure-state density matrix. Allowing convex combinations does not change the global minima as they will always have higher energy than the ground state. Convex combinations of the initial starting point of the minimization and the actual ground state 2-RDM are added to the training set. The idea is that a path down to the ground state will be formed in the discriminator function. In theory, this is not needed as if D always correctly classifies its input there will be a path inside $P(N)$ between any initial guess and the ground state. However, flaws in D and problems with getting stuck in a local minimum can make the use of "shortcuts" through convex combinations practical.

The loop updating the training set *should* both prevent false positives and create paths in the landscape down to the ground states. The first test will be if the network can converge to the solution on the test Hamiltonians, but generalization to other Hamiltonians are vital. Unfortunately, the results of this approach are very disappointing as we will see in the results section.

3.2.2 Autoencoder

The fundamental idea of the AE (autoencoder) approach is to force a network to encode N representable density matrices into a smaller space $\mathbb{R}^m$ that is also called the Z layer (not to be confused with the set of integers). The hope is that the AE will discover relationships between elements and remove such relationships to fit all N representable density matrices into the Z layer. Figure 4 illustrates the idea that by extensively sampling $P(N)$ and making m so small that the AE *needs* the entirety of $\mathbb{R}^m$ to encode $P(N)$, an approximate parametrization of $P(N)$ is formed.

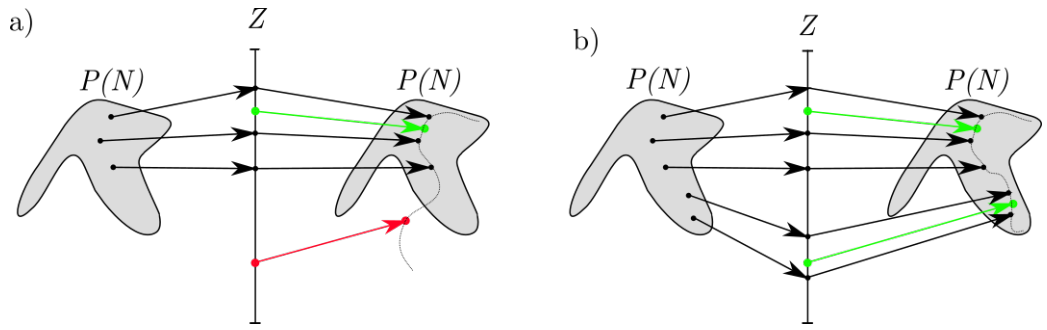

Figure 4: Visualization of the idea behind using an AE for constrained search. $P(N)$ is the set of N representable 2-RDMs, Z is the layer forming the bottleneck of the AE. The arrows are the mappings defined by the encoder: $P(N) \to Z$ and generator $G : Z \to P(N)$. The faint dotted line is the image of G. **a)** The AE is trained on too few samples, or the Z layer is too large. The image of the generator network is not contained within $P(N)$. **b)** By increasing the number of points, the network "needs" more of the Z space to encode elements from $P(N)$. The image of G becomes a better approximation of $P(N)$.

The main innovation of this work is that we train an autoencoder such that the decoder part (G for generator) will have the property $G[\mathbb{R}^m] = P(N)$. If so we can change $E(x) \to E(G(z))$ and $P(N) \to \mathbb{R}^m$ and get:

$$G[\mathbb{R}^m] = P(N) \Rightarrow \min_{x \in P(N)} E(x) = \min_{z \in \mathbb{R}^m} E(G(z)) \tag{12}$$

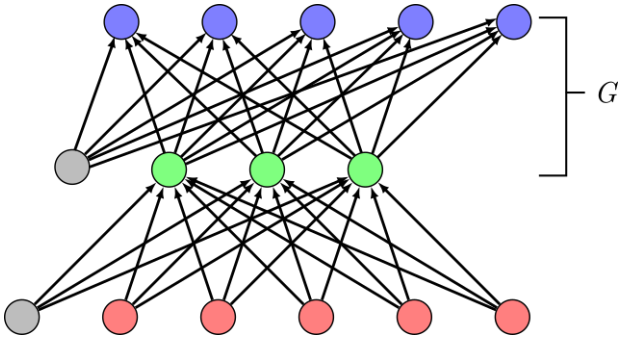

Figure 5: The figure shows an illustration of a simple feed-forward artificial neural network AE. Input is feed into the bottom neurons (red). Weights then multiply the inputs (represented by the lines). The result is summed together in the next layer (green) before the activation function is applied. The procedure is applied one more time to produce the output neurons (blue). In both layers, there is also a bias node (grey) that always gives the same value independent of input. The decoder part of the network forms the generator G. Figure based on [22]

In practice, an AE is nothing more than an ANN with the goal of reproducing the input in the output. The difficulty for the network is created by having a bottleneck in the network architecture. The most straightforward AE (as the one shown in figure 5) has two layers and the hidden layer is the bottleneck. After training, the decoder part of the network is separated from the encoder and used as the generator function G in equation 12. The activation function used is the hyperbolic tangent, tanh. The loss is defined as the mean square error between input and output of the network. For each training step, the loss is calculated over a small subset called a batch of all training data.

The above-described approach will in the best case characterize $P(N)$ for one specific N. To calculate systems with another number of particles the training and architecture fine tuning needs to be redone. Adding N as an extra element in the parametrization the network can be told to encode into $P(4)$ or $P(5)$. To be able to create a modified generator $\tilde{G} : Z, N \to P(N)$, N need to be feed as input to the hidden layer. This is done by adding the squared difference between one neuron in

the Z layer and the input neuron that is feed with the number of particles. The extra loss will make the network allocate that Z neuron to hold N, and it becomes possible to change N after training by changing the input to that hidden neuron.

The ability to generalize in N becomes even more critical when scaling to higher particle numbers. In section 2.1 the number of free variables of the 2-RDM approach is M^N, where M is the number of one-particle base functions used. Generating positive examples suffer from a time complexity of $\mathcal{O}(\binom{M}{N})$.

3.3 Generating data

Drawing random samples from $P(N)$ is done by drawing random wave functions Ψ from the N-particle wave-function Hilbert space $\mathcal{H}^{\otimes N}$. Each wave-function can then be transformed into an N representable density matrix via equation 4 derived in section 2.1. From the density matrix, the log-Cholesky parametrization can be computed, and we have our N representable data point.

$$\Gamma^{ij}_{lk} \equiv \langle \Psi | \, a_i^\dagger a_j^\dagger a_k a_l \, | \Psi \rangle \tag{13}$$

The implementation details of the C++ code sampling wave-functions and reducing them to log-Cholesky parametrizations of 2-RDMs can be found in the GitHub repository [26]. Appendix A describes the algorithm reducing Ψ to Γ^{ij}_{lk}. The GNU Scientific Library [27] is used for the Cholesky decomposition.

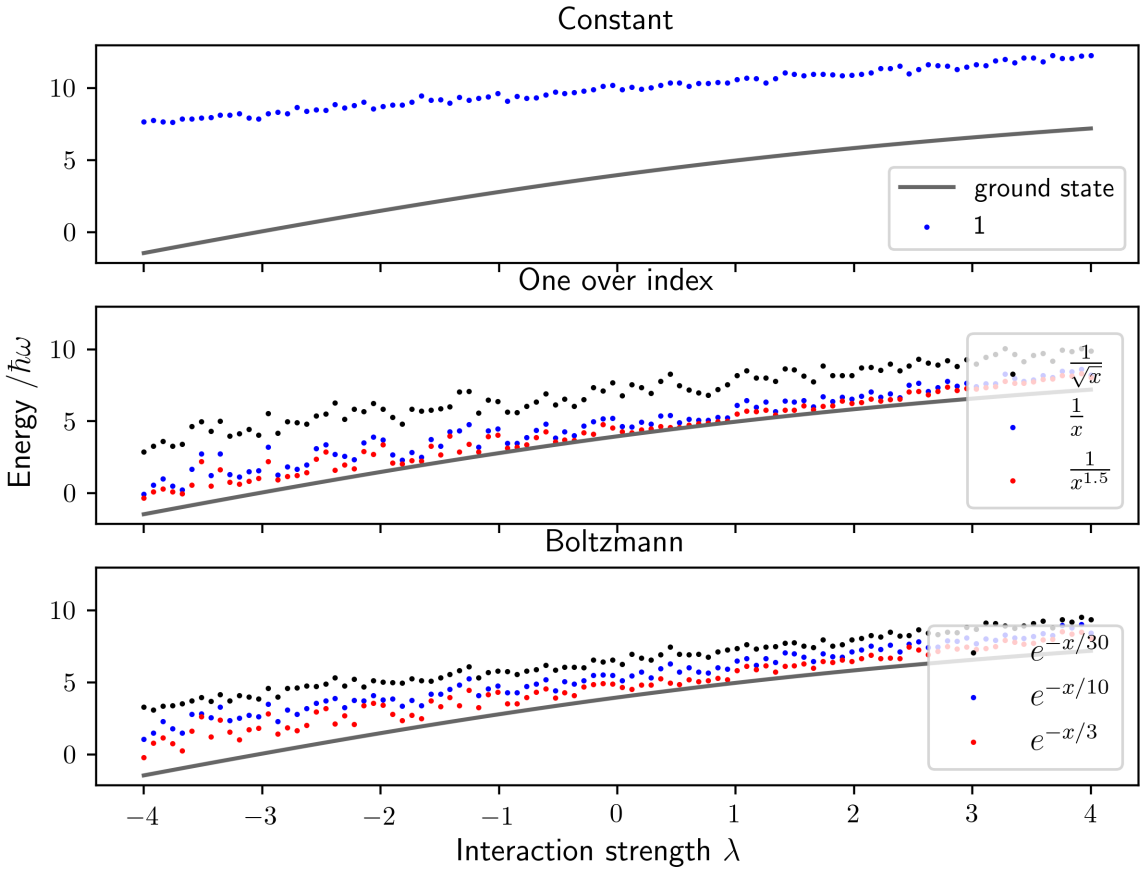

Figure 6: Sampling positive examples according to different N particle wave function distributions. x is the coefficient number. The energy expectation values are calculated relative to a test Hamiltonian. The test Hamiltonian consists of 4 particles in a harmonic oscillator interacting with delta interaction for different interaction strengths. The black line is the ground state energy as a function of interaction strength. Note that all positive examples must be above the ground state line. Ground state energies are known by configuration interaction calculations.

The natural choice is to draw the coefficients of Ψ uniformly and then normalize the resulting state. As seen in figure 6, uniform sampling results in very little spread

in energy. What we are seeing is that a uniform distribution of coefficients results in a non-uniform distribution of energies. This distributio is caused by that there are many more wave functions with energy in the middle of the energy spectra.

It is not the high entropy states that we are looking for but the ground states. To incorporate this bias, we need to abandon the uniform distribution. First assume the basis is ordered in energy expectation value, starting with the lowest. Then the ground state we are looking for will be heavily skewed toward the first couple of coefficients.

The straightforward approach is thus to draw N-Particle state coefficients according to a distribution tilted toward the low energy states. Let d be a function describing this "tilt" such that d takes the coefficient index as input. The i:th coefficient is drawn from $\text{uni}(-d(i), d(i))$. After all coefficients are drawn, the state is normalized. Figure 6 shows energies of samples drawn with different d.

Adding a bias toward some states might seem very limiting, but to parameterize the full space is unnecessarily challenging, when we have prior knowledge of in what neighborhood we are going to find the solution.

3.4 On not reinventing the wheel

Up until this section, we have formulated our problems down to unconstrained optimization. As mentioned before calculating the gradient of ANNs is cheap, so it makes sense to use a gradient-based method. The naive approach would be to write a program that calculates the derivatives given architecture, weights, and input and use gradient descent to do optimization. With such a formidable software to write, there would probably be little to no results to present in this report. Further, building and debugging such a sophisticated piece of software would force design choices giving little to no agility for prototyping new ideas.

Instead, we build our work using Tensorflow [28]. Tensorflow is a free (as in freedom) python package with a C++ back-end that facilitates the creation of symbolic functions. Notably, derivatives of the symbolic functions can easily be

calculated and an optimization schemes applied in a line of code. Tensorflow is optimized for machine learning in general and training ANNs in particular. A result of Tensorflow's focus on machine learning is that Tensorflow support distributed workloads on both different processing units inside the same computers and a cluster of computers. Tensorflow gives the user precise control of the machine learning model but automates the grunt work of implementing all the details of gradient calculations enabling fast prototyping.

Even though the application of constrained optimization is as far as I know unique, the dense and convolution layer described in section 2.2 are standard. Constructing custom gradient-based optimization schemes is outside the scope of this work as well. Again it is foolish to try and reinvent the wheel. Keras [29] is an extra layer on top of Tensorflow that takes care of the details of implementing different types of standard layers and ANN training schemes. It is still possible to access the underlying Tensorflow objects when needed. Such a case arises when calculating the energy given a log-Cholesky parametrization.

Using Keras with the Tensorflow back end, work that ten years ago would have demanded the attention of a small herd of computer science Ph.D. students now can be achieved by a single master student. This progress in facilitating prototyping makes it even more important sometimes to stop and think about what one is doing.

3.5 A note about all the free parameters

In Freeman Dysons essay "A meeting with Enrico Fermi" [30], Dyson describes a meeting with Fermi in which according to Dyson, Fermi had the following to say about the use of arbitrary variables in theoretical physics:

> I remember my friend Johnny von Neumann used to say, with four parameters I can fit an elephant, and with five I can make him wiggle his trunk

In the essay, Dyson describes how Fermi "politely but ruthlessly demolished a

program of research that [Dysons] students and [Dyson himself] had been pursuing for several years". Fermi is also credited with the quote:

> There are two ways of doing calculations in theoretical physics. One way, and this is the way I prefer, is to have a clear physical picture of the process that you are calculating. The other way is to have a precise and self-consistent mathematical formalism. You have neither.

So what would Fermi think about using ANNs in theoretical physics? The number of free parameters is in the hundreds of thousands. A clear physical picture is nowhere to be seen, and even though the reduced density matrix formalism is a consistent mathematical formalism, our methods rely on ANNs and why ANNs work is not even well understood.

So should we do as Dyson and his students, disperse and find other lines of work? Maybe, maybe not. First of all the weights are not actually free as they are determined by the method through training. Instead it is the system architecture constitutes the free parameters that will be adapted to work well on the test systems. The property that separates the wheat from the chaff is the ability of an ANN to generalize to unseen systems. Therefore, we need to be meticulous about the ANN selection method. The act of selecting a network architecture based on its performance for a system results in that we can no longer use the result on said system to asses if the method works.

Our solution to this problem is two-fold. For every architecture tested, the Hamiltonians that it will be tested against are explicitly stated. The tests and architecture selection with the help of test Hamiltonians is automated with a clear and stated metric. There is a critical need to select the result to present solely based on specific, explicit tests. Even though the best network according to the test score is not the best overall network we must only choose according to the formal tests when presenting results.

4 Results

Up till now, there have been a lot of ideas, math, and formalisms. But in the end, methods need to answer questions. Now it is time to investigate if our approach works. Everything up to this point can be done agnostic to the Hamiltonian we want to solve. Using our technique on a particular system comes down to deciding the one-particle basis, the interaction function $I(x_1, x_2)$ between two-particles and the external potential $V(x)$. From equation 2 and 3 the Hamiltonian in equation 4 can be calculated and an energy expectation value function created from equation 5.

The systems used to test the method have 4-7 fermions in a one-dimensional harmonic oscillator $V(x) = \frac{1}{2}x^2$ with three different interactions $I(x_1, x_2) = \delta(z_1 - z_2), e^{\frac{(z_1-z_2)^2}{0.5}}, e^{\frac{(z_1-z_2)^2}{1.0}}$. The one-particle base is chosen to be the ten lowest eigenstates to the one-particle harmonic oscillator.

Before we take a look at how the ANNs perform let's see what happens if only the normalized log-Cholesky parametrization (see section 3.1) is used. Using just the parametrization corresponds to searching over the space of two-particle density matrices. The space of two-particle density matrices contain more than $P(N)$. The results in figure 7 are thus unsurprisingly lower than the actual ground state energy. However, by the nature of gradient-based optimization algorithms, there is no way of knowing if our solutions correspond to the global minima or just a local minimum.

4.1 Discriminator

The discriminator approach started this project. It is based on ANNs achievements in classifying data [9]. On the task of distinguishing between negative and positive samples, the network achieved satisfactory accuracy. The problem? Adversarial examples. The dashed line in figure 8 is the output of the constrained minimization scheme in equation 11 before adding any adversarial examples to the training data. The error is larger than the search over all density matrices in figure 7. Without adversarial examples in the training set, the punishment term did not hinder the

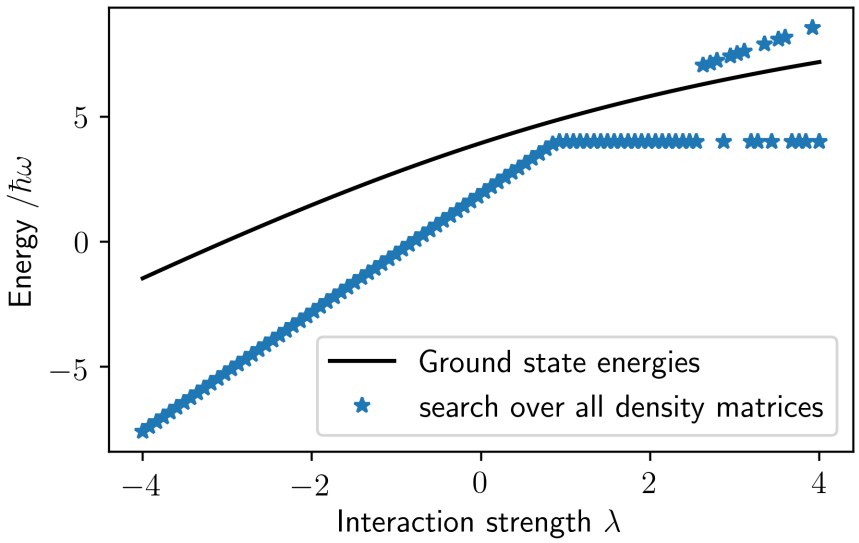

Figure 7: Variational calculations of ground state energies over the space of two-particle density matrices using log-Cholesky parametrization with fixed trace. The test system consists of 4 fermions in a one-dimensional harmonic oscillator with delta interaction. The solution is calculated for 100 interaction strengths evenly distributed between $\lambda = -4$ and 4. The reference solutions are CI calculations.

search from going outside of the set of 2-RDMs. Instead, the punishment term leads the optimization past the local minima that trapped the search in figure 7, resulting in an even larger error than with no discriminator at all.

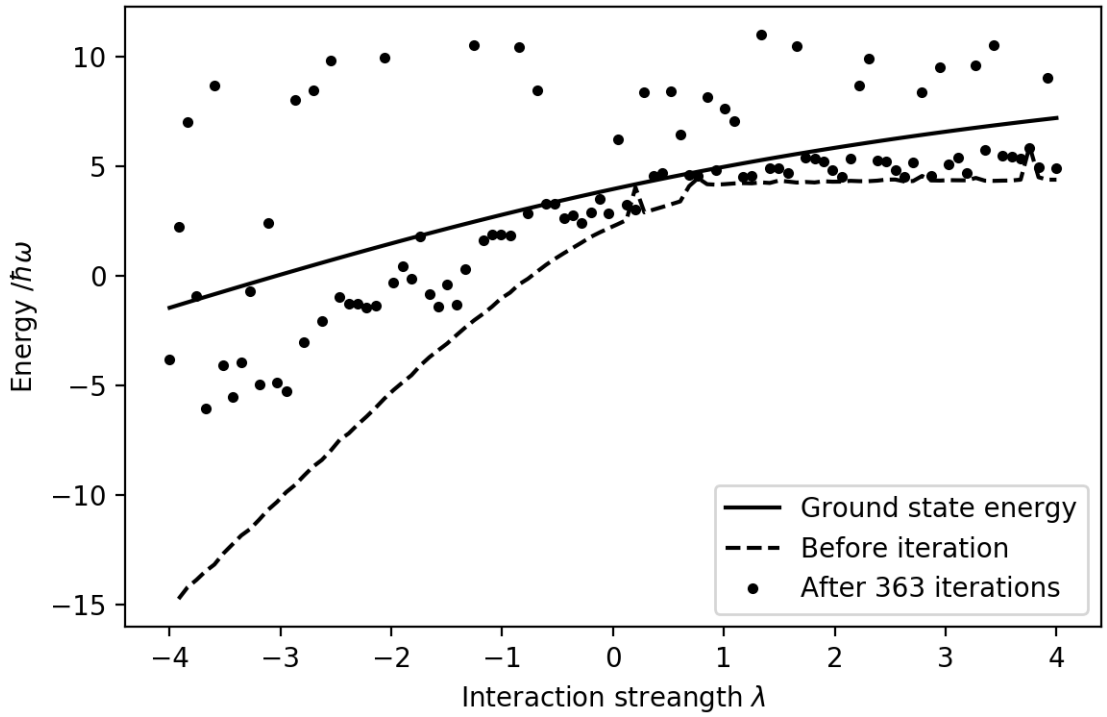

Figure 8: Calculated ground state energies of 4 fermions with delta interaction in a harmonic oscillator. The adversarial examples loop was run for 700 iterations. Iteration 363 was the one with the lowest mean square error. The reference solutions are from CI calculations.

That those adversarial examples would be a problem was clear from the beginning of the project, and the plan was always to add adversarial examples to the training set. The hope was that for every iteration of adding new adversarial examples (as described in section 3.2.1) the solution would slowly rise towards the ground state energy. That is not what happened.

As we see in figure 9, for the first 250-300 iterations the mean square error steadily improve. The lowest mean square error was achieved after 363 iterations of adding adversarial examples to the network. The solutions after 363 iterations are shown in figure 8. Gone is the continuous looking line and instead, the result seems stochastic. The stability of the solution with regards to small changes in starting position and interaction strengths is at best weak.

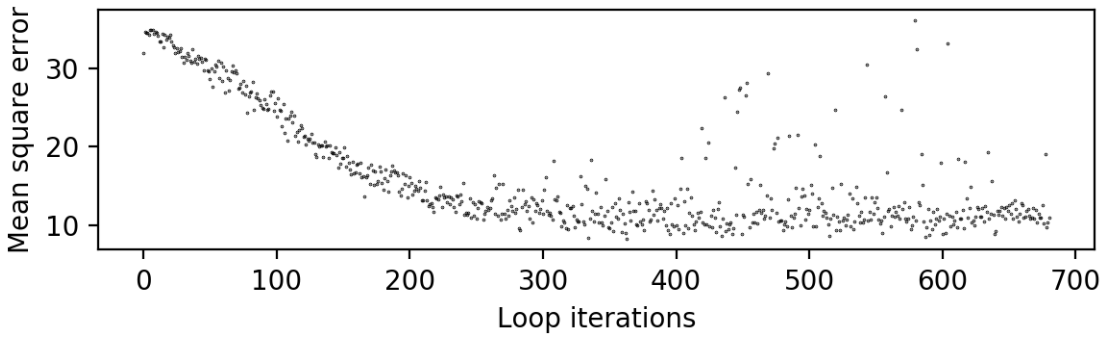

Figure 9: Mean square error of ground states calculations as a function of iteration. Each iteration adds new adversarial examples to the set used to train the discriminator

When investigating the solutions at different iterations, the picture is not that of a slow stable *rise* towards the ground state. Instead, the main driving force for better mean square error is the increased noise in the solution. The noise pushes answers up towards the ground states. However, once noise cause some solutions to be high above the ground state, the mean square error no longer shrinks. Instead, the mean square error also starts to exhibit more and more stochastic behavior.

It is interesting to note that after each iteration the validation accuracy never dropped below 90%. When dealing with programming in general and ANNs, in particular, a common theme emerges. The network does what we ask of it but not what we want from it. We ask it to classify examples generated with some sampling method correctly, and so it does. However, it does so in a weird way that does not fit well with what we want it to use the network for, act as a judge in a punishment

term.

Only two discriminator architectures were tested and only one with 700 iterations. In the next section on the AE hundreds of architectures where tested in an organized fashion to find the optimal architecture. So why not do the same thing for discriminators? The answer is threefold. First, completing 700 iteration takes on the order of days, while an AE takes less than 30 minutes to train. Secondly, even the worst AEs did not exhibit the stochastic behavior both discriminator tests did.

If the initial tests of an approach are unsuccessful, chances are that no amount of fine tuning will suffice. Instead, it is often a better idea to try out other ideas until a promising way forward is found, and then fine tune the promising design.

4.2 Autoencoder

A significant difference to the discriminator approach is that the iterations are no longer needed. The main advantage resulting from that is speed. The increased speed makes it possible to try many more architectures than the discriminator approach. Calculation quickness is also an essential factor in the general attractiveness of the method. However, high design iteration frequency also adds complexity, the many architectures that we test cannot all be shown. As discussed in section 3.5 we need to be clear about what metric has been used to select the results presented.

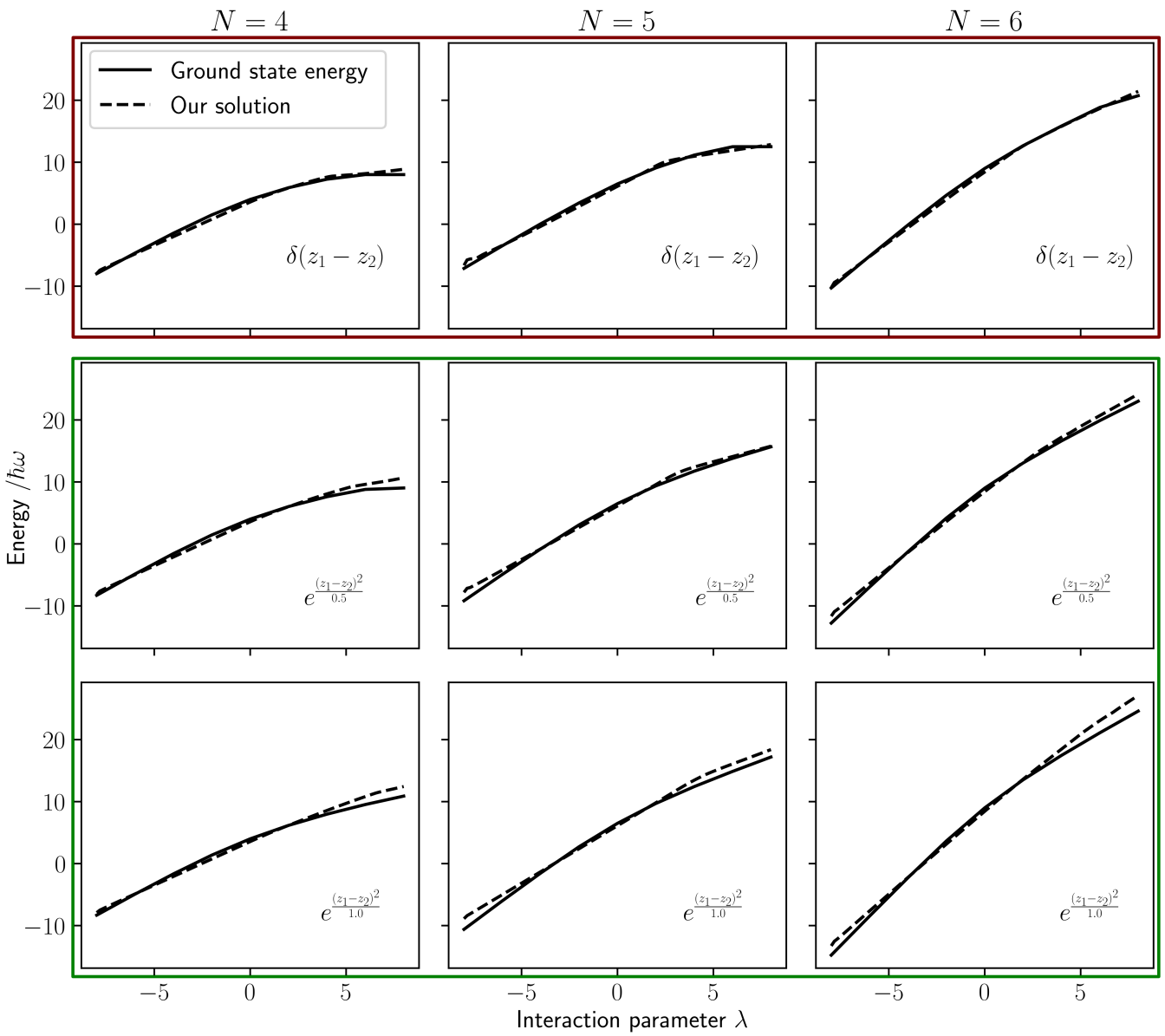

Figure 10: Generalizing in interaction type while allowing for different architectures and weights for different particle numbers. Each row has the same interaction type as written in each sub-figure. Columns have the same particle number as indicated at the top. The mean square error on the systems with delta interaction was used as a metric. The results that were used in choosing the architecture are in the red box, and unseen systems are in the green box. The reference solutions are from variational calculations by Gillis Carlsson [31].

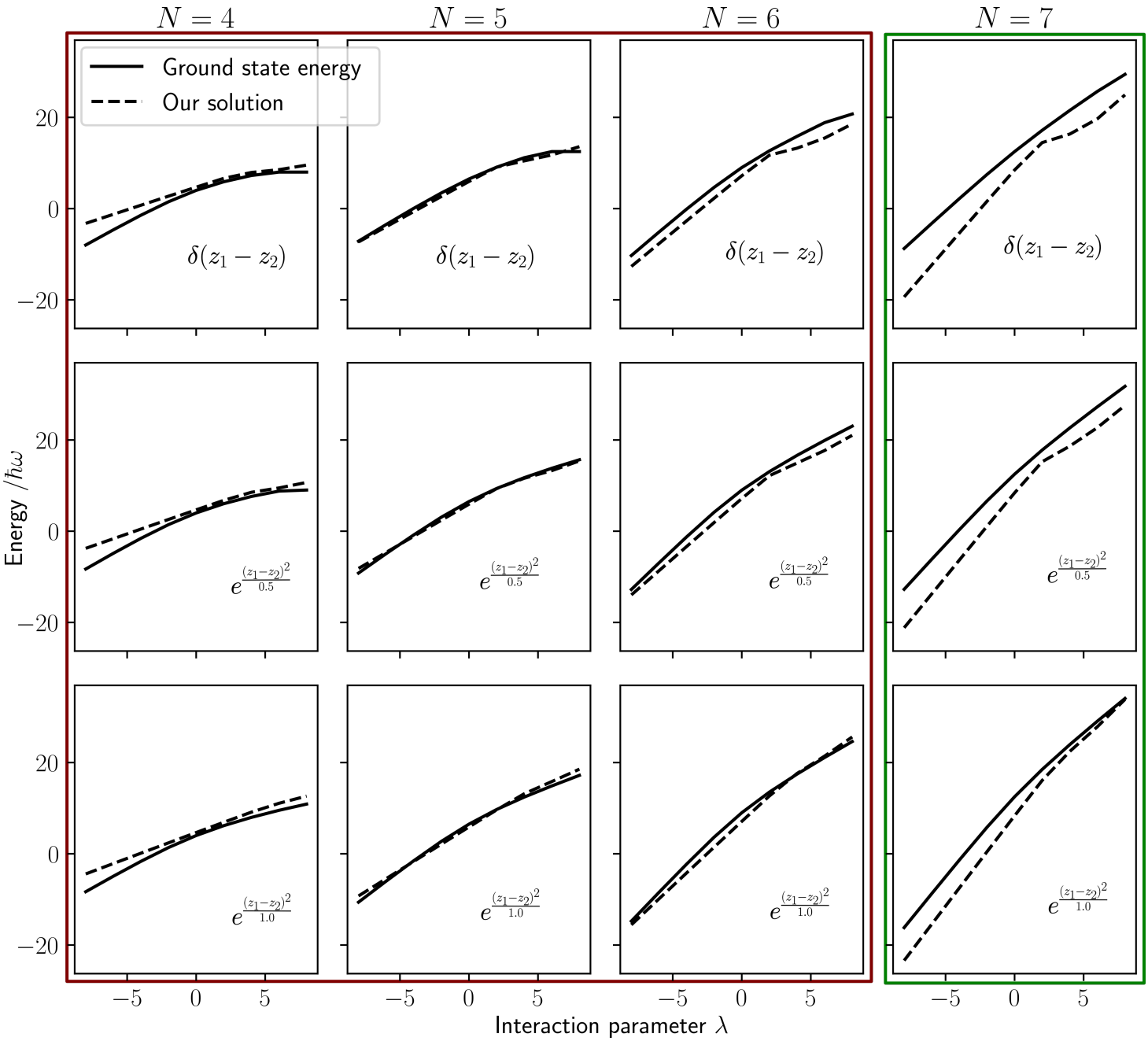

Figure 11: Results of attempt to generalize in particle number. The metric used is the mean square error of all systems with four, five, and six particles. The AE is trained on four-, five-, and six-representable density matrices. Each row has the same interaction type as written in each sub-figure. Columns have the same particle number as indicated at the top. The results that were used in choosing the architecture are in the red box, and unseen systems are in the green box. The reference solutions are from variational calculations by Gillis Carlsson [31].

Figure 10 shows that using the mean square error on systems with delta interaction as metric our method generalizes to other interaction types. We achieve generalization for both four, five, and six particles. Note that training a new AE is necessary when adapting the method to a different number of particles but not to an unseen Hamiltonian.

In section 3.2.2 the benefits of generalizing not only to new Hamiltonians but also to higher particle number is discussed. Figure 11 shows the result of our attempts to achieve such generalization. Note that our solution is too high for $N = 4$, accurate for $N = 5$, too low for $N = 6$, and much to low for $N = 7$. This trend is clear for all interaction types. Also, note that the errors are qualitatively similar when changing interaction but not when switching particle number. The hypothesis is that networks fail to take into account the changes between $P(4)$, $P(5)$, and $P(6)$, only approximating $P(5)$. Something more seems to be needed to make an AE that successfully generalize in the number of particles.

5 Conclusion

How did the marriage between the old quantum mechanical formalism and the hot buzz-word technology end up? The work aimed to approach the QMA-hard problem of N representability from a neural network perspective and produce a proof-of-concept. After a detour in classifiers and adversarial examples, we have shown that AEs can find parametrizations of $P(N)$ that produce results in qualitative agreement with exact solutions for the subspace. So we achieved the goal of providing a proof-of-concept that ANNs can be used to tackle the N representability problem. However, we have not show that ANNs can be useful in that role and compete with standard methods.

6 Prospects

The next step is to use the AE approach for fixed particle number and increase the number of orbitals and use it for real physical problems. The aim is to do calculations in a situation where polynomial scaling free parameters becomes significant.

It is worth to note that the AE approach described herein contains none of the progress that have been made on the N representability problem since the 1950s. The technique introduced in appendix B makes it possible to integrate knowledge of more physical conditions and incorporate them in the AE architecture. Implementing the method described in appendix B is a lot of work but and might not even be possible. It could however, for example, form the starting point of a future diploma-work.

Using an autoencoder to train a network into having the image of the training set is not commonly used. Instead, generative neural nets (GANs) [32] are the state-of-the-art in generating content such as images. Training GANs is a tough challenge that also might yield interesting results.

References

[1] Werner Heisenberg. *Der Teil und das Ganze Gespräche im Umkreis der Atomphysik*. 1969.

[2] P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *Phys. Rev.*, 136:B864–B871, Nov 1964.

[3] Wolfram Koch and Max C Holthausen. *A chemist's guide to density functional theory*. John Wiley & Sons, 2015.

[4] Eleftherios N Economou. *Green's functions in quantum physics*, volume 3. Springer, 1983.

[5] A J. Coleman. Structure of fermion density matrices. *Reviews of Modern Physics*, 35:1, 06 1962.

[6] C. A. Coulson. Present state of molecular structure calculations. *Rev. Mod. Phys.*, 32:170–177, Apr 1960.

[7] DO Hebb. The organization of behavior. *New York*, 1949.

[8] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[10] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[11] Hany Hassan et al. Achieving human parity on automatic chinese to english news translation. *arXiv preprint arXiv:1803.05567*, 2018.

[12] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5:4308, 2014.

[13] Giuseppe Carleo and Matthias Troyer. Solving the quantum many-body problem with artificial neural networks. *Science*, 355(6325):602–606, 2017.

[14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[15] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[16] Henry W Lin, Max Tegmark, and David Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168(6):1223–1247, 2017.

[17] Maho Nakata, Mituhiro Fukuda, and Katsuki Fujisawa. Variational approach to electronic structure calculations on second-order reduced density matrices and the n-representability problem. *Complex Quantum Systems-Analysis of Large Coulomb Systems, Institute of Mathematical Sciences, National University of Singapore*, pages 163–194, 2013.

[18] Yi-Kai Liu, Matthias Christandl, and Frank Verstraete. Quantum computational complexity of the n-representability problem: Qma complete. *Physical review letters*, 98(11):110503, 2007.

[19] C Garrod, Miodrag V Mihailović, and Mitja Rosina. The variational approach to the two- body density matrix. *Journal of Mathematical Physics*, 16(4):868–874, 1975.

[20] Michael A Nielsen and Isaac L Chuang. Quantum computation and quantum information, 2000.

[21] Albert John Coleman and Vyacheslav I Yukalov. *Reduced density matrices: Coulsons challenge*, volume 72. Springer Science & Business Media, 2000.

[22] Martin Thoma. Feed-forward-perceptron.svg. `https://commons.wikimedia.org/wiki/File:Feed-forward-perceptron.svg?fastcci_from=281461&c1=281461&d1=15&s=200&a=fqv`, 2013.

[23] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[24] José C. Pinheiro and Douglas M. Bates. Unconstrained parametrizations for variance-covariance matrices. *Statistics and Computing*, 6(3):289–296, Sep 1996.

[25] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[26] Tandem. `https://github.com/emiljoha/tandem`, 2018.

[27] Brian Gough. *GNU scientific library reference manual*. Network Theory Ltd., 2009.

[28] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[29] François Chollet et al. Keras. `https://keras.io`, 2015.

[30] Freeman Dyson. A meeting with enrico fermi. *Nature*, 427(6972):297, 2004.

[31] Gillis Carlsson. personal communication.

[32] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

A The TANDEM algorithm

TANDEM calculates the 2-RDM Γ of a state Ψ, where Ψ is a state consisting of N fermions. The input to TANDEM is the coefficients that define Ψ. The basis for Ψ is the antisymmetrized products of M one particle states.

$$\Gamma_{kl}^{ij} = \langle \Psi | \, a_i^\dagger a_j^\dagger a_l a_k \, | \Psi \rangle \tag{14}$$

Each basis state is represented in the algorithm by a string of M bits. Each bit represents the occupation number of an orbital. As the particles are fermionic the occupation number can be only 0 or 1. Each bit string will consist of N ones and $M - N$ zeros.

We need a convention to map bit strings of length M, like 01011, to second quantization states like $a_i^\dagger a_j^\dagger a_k^\dagger \, |0\rangle$. If we assume the one particle orbitals are ordered according to energy expectation value for some Hamiltonian, to make low energy states come first in the basis ordered in lexicographical order, the index is counted from the right starting at 1. In the example above the indices will become 1, 2, and 4. As the creation operators anti-commute, to have a consistent sign there is a need to specify in which order to apply the creation operators. For example: $a_4^\dagger a_2^\dagger a_1^\dagger \, |0\rangle = -a_4^\dagger a_1^\dagger a_2^\dagger \, |0\rangle$. The convention adopted is to order the creation operators such that the one with the lowest index acts first. Now we have created a one-to-one mapping between bit strings of length M with N non-zero elements to second quantization states formed by applying creation operators to vacuum. For $M = 5$, the example state is unambiguously identified as $|01011\rangle = a_3^\dagger a_1^\dagger a_0^\dagger \, |0\rangle$. All basis states are permutations of such bit strings. The basis is ordered in lexicographic order.

Expanding Ψ in the basis we get $|\Psi\rangle = \sum_s C_s \, |s\rangle$. The sum is over all bit permutations of bit strings s of length M with N ones. We rewrite Γ in equation 14 as:

$$\Gamma_{kl}^{ij} = \sum_{s's} C_{s'}^* C_s \, \langle s' | \, a_i^\dagger a_j^\dagger a_l a_k \, | s \rangle \tag{15}$$

The symmetries of Γ^{ij}_{kl} given in equation 6 means that many elements can be trivially deduced from others. Only some elements need to be calculated explicitly by TANDEM as the rest can later be deduced if needed. The choice of which variables to use is:

$$\begin{aligned} &i < j \quad k < l \\ &\text{and if } i = k \text{ then } j < l \end{aligned} \tag{16}$$

The general idea of TANDEM is to iterate over the permutations of s and s' and iterate though the ways the second quantization operators can turn s into s'. The sign of $\langle s'| a_i^\dagger a_j^\dagger a_l a_k |s\rangle$ is calculated by applying the creation and annihilation operators counting the number of commutations needed. Then adding $C_s^* C_{s'}$ multiplied with the calculated sign to the appropriate position in Γ. Pseudo code can be found in algorithm 1 and the complete code including python bindings in the git repository [26].

The algorithm as a function of wave function is efficient and executes in polynomial time. But the N-particle basis have the length $\binom{M}{N}$, where M is the number of one-particle orbitals used. The algorithm itself is efficient, but the input as a function of M and N is not. If used as a way to randomly sample $P(N)$, taking M and N as input, the computational complexity is $\mathcal{O}\left(\binom{M}{N}^2 M\right)$.

A.1 Example

The simplest non trivial case of calculating the 2-matrix is when there are 3 particles in 4 orbitals with $|\Psi\rangle = |0111\rangle$. By equation 4 the 2-RDM is given by:

$$-\Gamma^{ij}_{kl} = \langle\Psi| a_i^\dagger a_j^\dagger a_k a_l |\Psi\rangle = \langle 1110| a_i^\dagger a_j^\dagger a_k a_l |0111\rangle$$

TANDEM start with the state $s = s' = 0111$. This will give a non-trivial contribution. The next iteration have the same s but $s' = 1011$. As $C_{1011} = 0$ TANDEM will move to the next s'. As all these will be zero nothing more than

Algorithm 1 Tandem

1: $\Gamma^{ij}_{kl} \leftarrow 0$ for all $ijkl$
2: $C \leftarrow$ Coefficients of Ψ
3: $S \leftarrow$ List of all permutations in lexicographic order
4: **for** s **in** S **do**
5: **if** $C_s \neq 0$ **then**
6: **for** s' **in** S **do**
7: **if** $C_{s'} \neq 0$ **then**
8: $\Gamma =$ CONTRIBUTION(Γ, s, s')
return Γ
9: **function** CONTRIBUTION(Γ, s, s')
10: $a \leftarrow s$ Xor s'
11: $b_s \leftarrow a \,\&\, s$
12: $b_{s'} \leftarrow a \,\&\, s'$
13: **if** Number of nonzero elements in $a > 4$ **then**
14: **return** Γ
15: **else if** Number of nonzero elements in $a = 4$ **then**
16: (k, l) and $(i, j) \leftarrow$ non-zero elements of b_s and $b_{s'}$
17: $i, j, k, l \leftarrow i, j, k, l$ ordered according to equation 16.
18: $S \leftarrow 0$
19: **for** q **in** (k, l, j, i) **do**
20: $s_q \leftarrow$ **not** s_q
21: **for** all $s_i = 1$ with $i > q$ **do**
22: $S \leftarrow S + 1$
23: $\Gamma^{ij}_{kl} \leftarrow \Gamma^{ij}_{kl} + (-1)^S C_s C^*_{s'}$
24: **return** Γ
25: **else if** Number of nonzero elements in $a = 2$ **then**
26: k and $i \leftarrow$ non-zero elements in b_s and $b_{s'}$
27: **for** $k = i =$ common nonzero element of s and s' **do**
28: Do same steps as on line 16-21
29: **return** Γ
30: **else if** Number of nonzero elements in $a = 0$ **then**
31: $L \leftarrow$ indices of non-zero element of s.
32: **for all** $\binom{N}{2}$ choices of i, j from L **do**
33: Do line 16-21 for i, j, i, j
34: **return** Γ

checking if coefficients are zero will be done. When all permutations have been iterated though on s', s will be changed. Now $C_s = C_{s'} = 0$. Now it is clear that no s' can make a difference and the algorithm moves on test the remaining permutations. None of them will give any contributions.

Going back to that first and only non-trivial combination $s = s' = 0111$. We calculate a and see that there is no non-zero elements in a.

$$a = s \oplus s' = 0111 \oplus 0111 = 0000 \tag{17}$$

$$\tag{18}$$

The list of ones in s becomes $L = [0, 1, 2]$. The combinations of taking two elements out of this set becomes $(0, 1), (0, 2), (1, 2)$. This means that the interesting i, j, k, l combinations will be $(0, 1, 0, 1), (0, 2, 0, 2), (1, 2, 1, 2)$. These indices are ordered so that they satisfies the conditions in equation 16.

The sign calculation is done by iterating over the four indices. Flip the k:th bit from the right and count the number of ones to the left of the k:th bit. Repeat for l, j, i. Table 1 contains these calculations for our example. Each row represents one flip and count. The first column is filled by k, l, j, i, the positions to flip. The second column are the bit strings after each flip with the flipped bit in bold. The third column is the number of ones before the flipped bit. Summing up the number of flips column gives 5, 3, and 1 respectively. As these are all odd the sign will be negative for all three. However, as the observant reader might have already noted this calculation have been calculating $\langle 1110| a_i^\dagger a_j^\dagger a_k a_l |0111\rangle$ instead of $\langle 1110| a_i^\dagger a_j^\dagger a_l a_k |0111\rangle$. Therefore, a final sign change is needed.

In the end, the non-zero elements of Γ becomes:

Table 1: Sign table calculations

(a)

	0111	count
0	011**0**	2
1	01**0**0	1
0	010**1**	1
1	01**1**1	1

(b)

	0111	count
0	011**0**	2
2	0**0**10	0
0	001**1**	1
2	0**1**11	0

(c)

	0111	count
1	01**0**1	1
2	0**0**01	0
1	00**1**1	0
2	0**1**11	0

$$\Gamma_{01}^{01} = \Gamma_{02}^{02} = \Gamma_{12}^{12} = 1 \Rightarrow$$
$$\Gamma_{10}^{10} = -\Gamma_{01}^{10} = -\Gamma_{10}^{01} = 1$$
$$\Gamma_{20}^{20} = -\Gamma_{02}^{20} = -\Gamma_{20}^{02} = 1$$
$$\Gamma_{21}^{21} = -\Gamma_{12}^{21} = -\Gamma_{21}^{12} = 1$$

B ANN approach to combining parametrization conditions

As mentioned in section 3.1 there are more necessary positivity demands for a density matrix to be N representablex. The matrices that are to be kept positive are often called P, G, Q, T_1 and, T_2 [17] and can be calculated as linear combinations of elements in the 2-RDM defined in equation 4, section 2.1.

Fortunately we have log-Cholesky parametrization that makes it possible to construct a general symmetric positive semi-definite matrix. As mentioned in section 3.1 it is not evidently clear how this can be useful in the case of multiple positivity demands. There is a need to synchronize the parametrizations such that they all generate the same two-particle density matrix. This appendix proposes an untested way of solving this problem by making the AE responsible for the synchronization.

The method is general to any number of different positivity conditions, as long as the matrices have a one-to-one mapping to Γ. Let's assume we have m conditions and the corresponding set of matrices $A_i(\Gamma)_{i\in\{1,\dots m\}}$. We call the function we want to minimize $E(\Gamma)$. The problem can be formulated as finding $\min_\Gamma E(\Gamma)$ under the condition $A_i(\Gamma) \succeq 0 \, \forall i$.

For every matrix A_i we can get the log-Cholesky parametrization x_i. Every example decomposed into the log-Cholesky parametrizations of all the matrices A as:

$$\Gamma \leftrightarrow \begin{bmatrix} A_1 \\ \vdots \\ A_m \end{bmatrix} \leftrightarrow \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \tag{19}$$

Figure 12 depicits the schematics of an AE that map one parametrization of Γ to z and then to every parametrization. The m parametrizations hold exactly the same information, that of Γ. The job of the AE $T = T_{\text{gen}} \otimes T_{\text{enc}}$ is to see this, and with T_{enc} encode that information in a way that it can be retrieved by T_{gen}. As I said before, x_i hold the same information so we really only need one parametrization as

input. Using only one parametrization reduces the number of parameters in T_{enc}.

The loss for the network in figure 12 is composed of two parts. First a regular AE part, mean square error of the expected output and the actual output. The second part makes sure that for any input z, the output parametrizations $x_1, \ldots, x_m$ all generate the same Γ. T^i_{gen} gives x_i from z as in figure 12. $\tilde{z}$ is a uniformly drawn from $[0,1]^m$ and $\alpha \in (0,1)$ determines how much the network priorities between the two goals. m is the number of neurons in the z layer.

$$\text{loss} = \alpha \|T(x_1) - (x_1, \ldots, x_m)\|^2 + (1-\alpha) \sum_{i,j} \|\Gamma[T^i_{\text{gen}}(\tilde{z})] - \Gamma[T^j_{\text{gen}}(\tilde{z})]\|^2 \qquad (20)$$

A large α means that the network priorities to successfully encode and decode the examples over that random input results in sensible synchronized parametrizations. For small α the network will prioritize to sync the output parametrizations for all input but making the network prone to mode collapse. Mode collapse means that the image of T_{gen} will be to small, giving the same output for many different inputs. It is the combination of the two loss terms that create overall synchronization while avoiding mode collapse.

The naive approach is to use T_{gen} in the place of G in the AE approach described and tested in the main work. But we can take it further and utilize the fact that we have a way to check if all the positivity conditions are met. The way we check is to assure that the parametrizations are synchronized. There is a natural metric of *how* close the parametrizations are to be synchronized. The second loss term in equation 20. By using this extra loss term as a punishment term we get the effective energy in equation 21.

$$\min_{\Gamma \in P(N)} E(\Gamma) = \min_z [E(\Gamma[T^0_{\text{gen}}(z)]) + \gamma \sum_{i,j} \|\Gamma[T^i_{\text{gen}}(\tilde{z})] - \Gamma[T^j_{\text{gen}}(\tilde{z})]\|^2] \qquad (21)$$

The error of AE approach described in equation 12 is hard to predict and understand. The network could potentially ignore any conditions not hard-coded into the

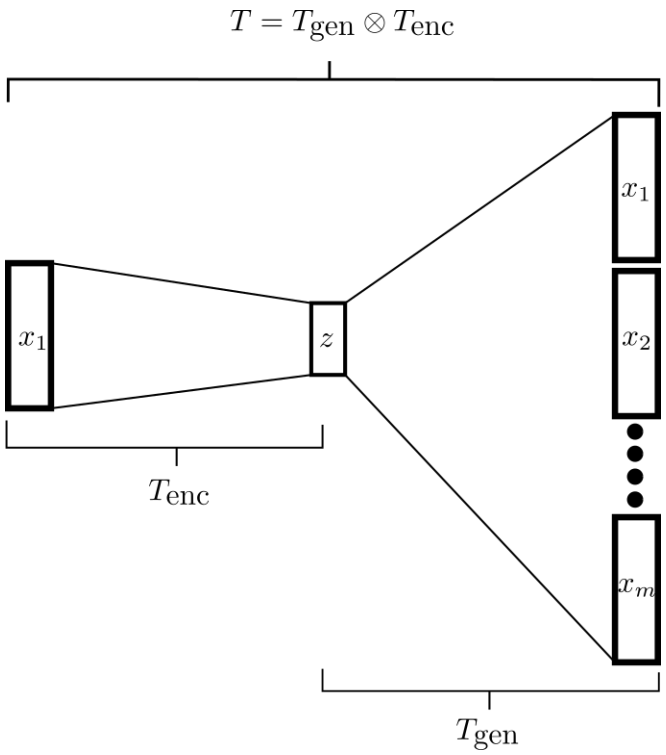

Figure 12: AE-like network architecture that takes as input one of the log-Cholesky parameterized matrices that must be positive. The output is trained to be the m log-Cholesky parametrizations for all the matrices that have a positivity condition. T_{gen} consist of m separate networks that all take the same z layer as input and returns the output parametrizations for one condition each. (Represented as separate rectangles in the output)

parametrization. Here the network cannot ignore the positivity conditions without increased loss. Further by adding the extra synchronization punishment term to the minimization the global minima (for a large enough γ) will be large or equal to the ground state energy as calculated with all the positivity conditions. Further, by checking the synchronization after convergence it is possible to assure that the found solution actually satisfies the conditions. If the punishment term is zero we know that the found solution satisfies the positivity conditions. If it is non-zero the solution might not satisfy the conditions. The increased knowledge of how the error behaves means that a network performance for problems can be compared without actually knowing the answer to the problem we test the network against. The network that gives the lowest energy while still maintaining synchronization is closest to the real answer. This only hold if no other conditions than the positivity conditions explicitly stated need to be respected.

N-Representabilty is not just a semi-definite programming problem. There are probably other conditions as well. These extra conditions need not to be ignored. The it is still an AE that will remove any unnecessary freedom to fit all data to the z layer just like in the regular AE approach.

Note that the method can be generalized to any set of conditions that individually can be guaranteed by parametrization. No modification needed the loss, effective energy or, the high-level network architecture as shown in figure 12.

C Network Details

The main ANN architecture and optimization parameters used for presented AE results are tabulated in table 2. Especially note that the decoder units specify the decoder but also indirectly the encoder part by the transpose regularization. For example if the decoder units is [10, 20, 100] it means that the Z layer will have 10 neurons. And the rest of the network will look like:

$$\binom{\binom{N}{2}+1}{2} \rightarrow 100 \rightarrow 20 \rightarrow 10 \rightarrow 20 \rightarrow 100 \rightarrow \binom{\binom{N}{2}+1}{2}$$

It is also important to remember that weight sharing is employed by transposing the dense layers. For example, in the example above, if $\omega_{20\rightarrow 10}$ is the weight matrix in the last layer of the encoder $20 \rightarrow 10$ then the matrix in the first layer of the decoder $10 \rightarrow 20$ is $\omega_{10\rightarrow 20} = \omega_{20\rightarrow 10}^{\intercal}$.

Table 2: AE method hyper parameters. The AE where all composed of dense layers. The encoder is the transpose of the generator with bias weights as only weights not shared. *Decoder units* specify the number of neurons in each layer, starting with the z layer until the last hidden layer. *Activation* sets the activation function for the hidden layers while *z layer activation* specifies the activation function for the output layer of the encoder.*Batch size* is the number of examples for which each gradient decent step is calculated over. *Gradient norm condition* sets the condition on the energy gradient norm at which the energy search is considered converged. The *loss* function specify the AE network training loss that is reduced with the algorithm specified by the *optimizer*.

Result	Decoder Units	Activation	Batch Size	Epochs	Gradient Cond.	Learning Rate	Loss	Optimizer	Z activation
Fig 11	37 31 29 182 1200	tanh	181	9	0.001	0.001	Mean Square	Adadelta	sigmoid
$N = 4$ Fig 10	14, 18, 31, 62	tanh	96	8	0.001	0.001	Mean Square	Adam	sigmoid
$N = 5$ Fig 10	18, 15, 33, 62	tanh	96	9	0.001	0.001	Mean Square	Adam	sigmoid
$N = 4$ Fig 10	20, 16, 27, 66, 251	tanh	97	9	0.001	0.001	Mean Square	Adam	sigmoid