Quantum Hopfield Neural Networks: A New Approach and Its Storage Capacity

Nicholas Meinhardt[1,2], Niels M. P. Neumann[1(✉)], and Frank Phillipson[1]

[1] The Netherlands Organisation for Applied Scientific Research, Anna van Buerenplein 1, 2595 DA The Hague, The Netherlands
{niels.neumann,frank.phillipson}@tno.nl
[2] Department of Physics, ETH Zurich, 8093 Zurich, Switzerland
nmeinhar@student.ethz.ch

Abstract. At the interface between quantum computing and machine learning, the field of quantum machine learning aims to improve classical machine learning algorithms with the help of quantum computers. Examples are Hopfield neural networks, which can store patterns and thereby are used as associative memory. However, the storage capacity of such classical networks is limited. In this work, we present a new approach to quantum Hopfield neural networks with classical inputs and outputs. The approach is easily extendable to quantum inputs or outputs. Performance is evaluated by three measures of error rates, introduced in this paper. We simulate our approach and find increased storage capacity compared to classical networks for small systems. We furthermore present classical results that indicate an increased storage capacity for quantum Hopfield neural networks in large systems as well.

Keywords: Hopfield neural networks · Gate-based quantum computing · Storage capacity · Quantum machine learning

1 Introduction

While conventional computers are restricted to classical operations, quantum computers implement the rules of quantum mechanics to process information [5], using quantum principles such as superpositions. The basic units to store information on quantum computers are two-level quantum bits, or qubits. Due to superpositions of both levels, qubits allow for a more flexible representation of information than classical bits. One widely accepted premise is that quantum computers have computational advantages over classical processing [6], giving rise to the notion of 'quantum supremacy' [13], which only recently has been claimed for the first time in experiments [1].

A candidate to show a quantum advantage is believed to be quantum machine learning (QML) [4,12], a field of research at the interface between quantum information processing and machine learning. Even though machine learning is an important tool that is widely used to process data and extract information

V. V. Krzhizhanovskaya et al. (Eds.): ICCS 2020, LNCS 12142, pp. 576–590, 2020.
https://doi.org/10.1007/978-3-030-50433-5_44

from it [4], it also faces its limits. The amount of data processed worldwide each year is steadily increasing, while the limits of computing power are rapidly approaching [7]. Therefore, more efficient algorithms, such as found in the quantum domain, are crucial.

We consider neural networks (NN), a subclass of machine learning algorithms consisting of nodes that can be connected in various configurations and interact with each other via weighted edges. As special case, Hopfield neural networks (HNN) consist of a single layer of nodes, all connected with each other via symmetric edges and without self-connections [8]. In an HNN, nodes are updated using the updating rule $x_i \leftarrow \text{sign}\left(\sum_{j=1}^{n} W_{ij}x_j\right)$, where $\text{sign}(\cdot)$ refers to the sign-function, W_{ij} is the weight between node i and j and $W_{ii} = 0$. A graphical representation of an HNN is given in Fig. 1, where k an indicator for the number of updating iterations in the direction of the dashed arrows.

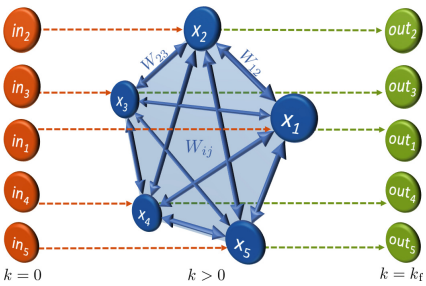


Fig. 1. Schematic overview of a fully-connected Hopfield neural network with 5 neurons. First, the neurons are initialized (orange nodes), then the network evolves in time or number of iterations k according to the weight matrix with entries W_{ij} (blue plane). The final configuration is read out (green nodes). The dashed arrows indicate the direction of updating or time. (Color figure online)

Due to this connectivity, HNNs can be used as associative memories, meaning that they can store a set of patterns and associate noisy inputs with the closest stored pattern. Memory patterns can be imprinted onto the network by the use of training schemes, for instance Hebbian learning [15]. Here, the weights are calculated directly from all memory patterns, and thereby only a low computational effort is required. It is possible to store an exponential number of stable attractors in an HNN if the set of attractors is predetermined and fixed [11]. In general, however, fewer patterns can be stored if they are randomly selected, resulting in a very limited storage capacity of HNNs. For Hebbian learning $n/(4 \log n)$ patterns can be stored asymptotically in an HNN with n nodes [9].

Translating HNNs to counterparts in the quantum domain is assumed to offer storage capacities beyond the reach of classical networks [14,18]. For example, in [18] a quantum HNN is proposed that could offer an exponential capacity when qutrits are used. When using qubits however, no increased capacity has been demonstrated yet for quantum HNNs.

In this work, we provide a new approach for hybrid quantum-classical HNNs, which 1) allows for classical and quantum inputs and outputs; 2) is able to store classical bit strings as attractors; and 3) fulfills three minimum requirements to allow an analogy to classical HNNs as posed in Ref. [16]. The first requirement is that the quantum HNN must comply with quantum theory while respecting the structure of NNs. The second requirement is that the quantum model should solve the discrepancy between unitary quantum evolution and dissipative dynamics of NNs. Thirdly, it should offer the feature of an associative memory, meaning that inputs are mapped to the closest stable outputs that encode the learned bit patterns. We furthermore provide numerical evidence that the capacity indeed increases for gate-based quantum HNNs, when storing randomly chosen bit strings.

Previously proposed implementations of HNNs either deal with non-random memory patterns [3], or do not account for the discrepancy between dissipative and unitary dynamics, one of the three minimum requirements [14]. We follow the recent proposal of deep quantum neural networks in Ref. [2] for our HNN-development. Our model involves a training set, which is generated based on the chosen memories, and all involved gate operations are optimized using the training scheme given in Ref. [2]. We test the model's ability to store randomly chosen bit strings and thereby estimate its capacity. While limited to small system sizes due to the model complexity, the results are compared to those of a classical HNN with Hebbian learning.

The remainder of this work is organized as follows: We present our quantum model in Sect. 2 and the setup for the simulations in Sect. 3. The results of these simulations are given in Sect. 4. Finally, we provide a summary of the results in Sect. 5 and a conclusion in Sect. 6.

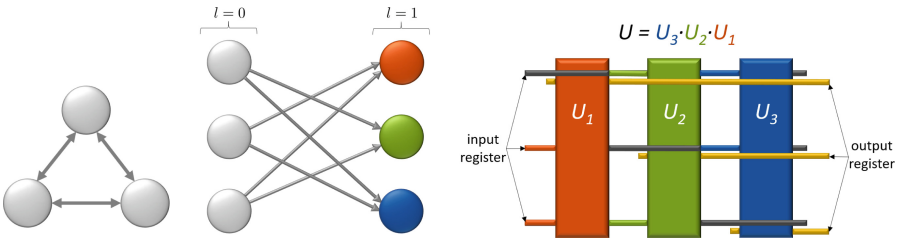
2 Quantum Hopfield Neural Networks

We first present a feed-forward interpretation of quantum HNNs in Sect. 2.1 and then explain how to train these feed-forward quantum HNNs in Sect. 2.2.

2.1 A Feed-Forward Interpretation of Quantum HNNs

HNNs can be implemented as feed-forward NNs by regarding each update step as a new layer of neurons. In the feed-forward interpretation, the weight matrix is layer depended and can be written as $W^{(l)}$. Depending on whether the HNN is updated synchronously or asynchronously, the weights might differ between layers. In the former case, the weights $W_{ij}^{(l)}$ are exactly as W_{ij} of the usual HNN. Hence, the weights are symmetric in both the subscripts and the layers and the superscript l can be omitted. Note that HNNs have no self-connections, such that $W_{ii} = 0$ for all i. Therefore, the interpretation of an HNN with synchronous updating as a feed-forward NN is valid. The number of layers l can be seen as a time parameter. Figure 2a shows an HNN with three neurons and a feed-forward interpretation of this network is given in Fig. 2b.

Note that we are not restricted to synchronous updating. In principle any updating rule may be applied and the weights of the feed-forward interpretation may differ drastically from the ones of the single-layer scheme in general. The weights do not necessarily need to agree with the ones of Hebbian learning. Note that the fundamental properties of HNNs of storing and retrieving patterns are retained.



(a) A Hopfield neural network with three neurons and without self-connections.

(b) A feed-forward interpretation of Fig. 2a. A single updating step l, without connections between the i-th neuron of adjacent layers. The colors indicate three different outputs.

(c) Quantum circuit of the network of Fig. 2b. Each unitary U_i is associated with the output neuron of the same color. The U_i act on a fresh ancilla qubit illustrated in gold and all input qubits except for the i-th qubit shown in gray. The overall quantum operation U is a product of all U_i.

Fig. 2. Two interpretations of the updating process in classical HNNs and the corresponding quantum model. (Color figure online)

One important advantage of the feed-forward interpretation is that we can use existing proposals to translate classical NNs to a gate-based quantum analog. To implement quantum analogs of feed-forward NNs, neurons can be implemented directly as qubits and weights between two neurons as operators [14,17]. We use another promising approach to feed-forward NNs, where unitary operations U acting on a quantum register are associated with the classical perceptrons [2]. In the following, we will only consider a single synchronous update. More updating steps can be added by repeating the presented approach.

Using the qubit encoding scheme, a classical bit string $(x_1, \ldots, x_n)$ is encoded in the corresponding computational basis state $|x_1 \ldots x_n\rangle$. In HNNs, neurons can only take the values ± 1 and we identify $+1 \leftrightarrow |0\rangle$ and $-1 \leftrightarrow |1\rangle$. Consequently, the classical input layer is replaced by state initialization of the quantum input register. The neurons of each subsequent layer of the classical feed-forward NN model are replaced by unitaries U_i, which act on the input register and each on an additional, freshly prepared ancilla qubit. Figure 2c gives an example of this quantum analogue for three neurons and a single update. The colors correspond with those of the classical neurons of the classical network in Fig. 2b and the golden lines represent the ancilla qubits.

Note that input qubit i is not affected due to the absence of self-connections. The only output qubit affected by unitary U_i is ancilla qubit i and the output state corresponds to the output of classical neuron i. To retrieve a classical output from the quantum system, ancilla qubits are measured at the end of the circuit. Using a majority vote over multiple measurement rounds, the most likely outcome is chosen as updated state. The original input qubits are discarded after an update round of applying all U_i, meaning the input register is traced out. For a single update, $2n$ qubits are needed. For l updates, i.e. for l layers, $(l+1)n$ qubits are needed and the output of a layer is used as input for the subsequent one.

2.2 Training a Quantum Hopfield Neural Network

The goal is to train the unitaries, opposed to variational quantum circuits, where classical gate parameters are trained and the gates themselves remain the same. Assume we have a training set of N input states $\left|\phi_k^{\text{in}}\right\rangle$ for training and their desired output states $|\phi_k^{\text{out}}\rangle$, for $k = 1, \ldots, N$. Let $\rho_k^{\text{in}} = \left|\phi_k^{\text{in}}\right\rangle\!\left\langle\phi_k^{\text{in}}\right|$ and let $\rho_k^{\text{out}} = U\left|\phi_k^{\text{in}}\right\rangle\!\left\langle\phi_k^{\text{in}}\right|U^\dagger$ be the actual output of the quantum circuit U with input $\left|\phi_k^{\text{in}}\right\rangle$. Furthermore, let the fidelity of the circuit be given by

$$\mathcal{F}\left(U\left|\phi_k^{\text{in}}\right\rangle\!\left\langle\phi_k^{\text{in}}\right|U^\dagger, \left|\phi_k^{\text{out}}\right\rangle\!\left\langle\phi_k^{\text{out}}\right|\right) = \left\langle\phi_k^{\text{out}}\right| U\left|\phi_k^{\text{in}}\right\rangle\!\left\langle\phi_k^{\text{in}}\right| U^\dagger \left|\phi_k^{\text{out}}\right\rangle. \tag{1}$$

This fidelity corresponds with how well the output state after applying the unitary gates matches the desired output state. The cost function C is defined as

$$C = \frac{1}{N}\sum_{k=1}^{N}\left\langle\phi_k^{\text{out}}\right| \rho_k^{\text{out}} \left|\phi_k^{\text{out}}\right\rangle. \tag{2}$$

To optimize C, we train the unitary operations $U_j(s)$, which are parametrized by s as a measure of the training iterations or the training duration. After a time step ε, the unitaries are then updated according to

$$U_j(s+\varepsilon) = e^{i\varepsilon K_j(s)} U_j(s), \tag{3}$$

where the $K_j(s)$ are Hermitian matrices given by

$$K_j(s) = \frac{i2^{n+1}}{2N\lambda}\sum_k \text{Tr}_{\text{rest}}[M_j^k(s)]. \tag{4}$$

Here $1/\lambda$ is a learning rate. These $K_j(s)$ can be estimated by taking a partial trace of matrices M_j^k that act on the whole space $\mathcal{H}_{2^{2n}}$ of all input and output qubits. This partial trace Tr_{rest} traces out all qubits not related to the unitary U_j. These qubits are all other ancilla qubits and the j-th input qubit if self-connections are removed. The M_j^k can be calculated from all unitaries, input and output training states as

$$\begin{aligned} M_j^k(s) = \Big[&U_j(s)\cdots U_1(s)\rho_k^{\text{in}} \otimes |0\ldots0\rangle\!\langle 0\ldots0| \, U_1^\dagger(s)\cdots U_j^\dagger(s), \\ &U_{j+1}^\dagger(s)\ldots U_n^\dagger(s)\mathbb{1} \otimes \left|\phi_k^{\text{out}}\right\rangle\!\left\langle\phi_k^{\text{out}}\right| U_n(s)\cdots U_{j+1}(s)\Big], \end{aligned} \tag{5}$$

where $[A, B] = AB - BA$ is the commutator of two operators A and B.

This updating scheme can be applied and implemented directly. In each iteration, all M_j^k are estimated and K_j is obtained by tracing out all unrelated qubits. Using Eq. (3), the unitaries are consequently updated in small time steps ε. The derivation of Eq. (4) and (5) involves Tayloring the exponential in Eq. (3) around $\varepsilon = 0$ to the first order and is provided in [2].

3 Simulating HNNs

In this section, we present the setup of our simulations of the HNNs. First we introduce the training set in Sect. 3.1, then we discuss the scaling of the simulations in Sect. 3.2 and afterwards we explain how to evaluate the performance of the HNNs in Sect. 3.3. Finally, Sect. 3.4 explains how we implemented the simulations and how we ran them.

3.1 Creating a Training Set

Let $\mathcal{S}$ be a set of m classical memory patterns $\mathcal{S} = \{\boldsymbol{x}^{(p)}\}_p$. We generate a training set $\mathcal{T}$ of input and output states from $\mathcal{S}$ using the qubit encoding. First, we add all memory patterns $\boldsymbol{x}^{(p)}$ as both input and output patterns to $\mathcal{T}$. Additionally, we add noisy memory patterns to the training set to prevent the unitaries from simply swapping input and output registers, without actually acting as an associative memory. All bit strings at a Hamming distance smaller or equal to d around each memory pattern in $\mathcal{S}$ are used as input states and are denoted by $\left|\boldsymbol{x}_{p_k}^{\text{in}}\right\rangle$. These $\left|\boldsymbol{x}_{p_k}^{\text{in}}\right\rangle$ states are noisy versions of the memory state $\boldsymbol{x}^{(p)}$. Hence, for each memory state $\boldsymbol{x}^{(p)}$, the respective $\left|\boldsymbol{x}_{p_k}^{\text{in}}\right\rangle$ are associated with $\left|\boldsymbol{x}_{p}^{\text{out}}\right\rangle$ as output states.

The number of training samples depends on the number of patterns at distance at most d to a given pattern. For m memories, the total number of generated training samples N_{train} depends on the binomial coefficients $\binom{n}{d}$ and is given by

$$N_{\text{train}}(m, d) = m \sum_{c=0}^{d} \binom{n}{c}. \tag{6}$$

Note that the order of training samples does not influence the results, as K_j is estimated as a sum over all training samples in Eq. (4). Also note that for large enough d, one noisy input pattern may be associated with different output states. For example, for $n = 3$, $m = 2$ and $d = 1$, the string $(1, 0, 1)$ is at distance one from both $(1, 0, 0)$ and $(0, 0, 1)$, yielding two contradicting training pairs. Consequently, the cost function in Eq. (2) cannot be exactly one, but takes smaller values. Clearly, the larger m and d are with respect to n, the more contradicting training pairs there are and the smaller the maximum of the cost function is.

3.2 Model Complexity

Let us consider the complexity of our model. To estimate K_j using Eq. (4), in each iteration, $\mathrm{Tr}_{\mathrm{rest}}[M_j^k]$ must be estimated for each training pair k. Hence, the duration of training is linear in the number of training samples $N_{\mathrm{train}}(m,d)$ and the time required to estimate $\mathrm{Tr}_{\mathrm{rest}}[M_j^k]$, denoted by $t_{M_j^k}$. The time to update U_j according to Eq. (3) is denoted as t_{upd}. This is repeated for all n unitaries and N_{iter} iteration steps. The total training duration is thus given by

$$t_{\mathrm{tot}} = N_{\mathrm{iter}} n \left(t_{\mathrm{upd}} + N_{\mathrm{train}}(m,d) t_{M_j^k} \right). \tag{7}$$

To estimate both terms in Eq. (5), we need $2(n+1)$ multiplications of $2^{2n} \times 2^{2n}$-matrices. As most matrices in Eq. (5) do not depend on k, the result of the multiplication can be reused. Therefore, the second term in Eq. (7) can be rewritten as $N_{\mathrm{train}}(m,d) t_{M_j^k} = \mathcal{O}\left((2n-4)2^{3(2n)} + 4N_{\mathrm{train}} 2^{3(2n)}\right)$, where we used that multiplying two complex $a \times a$ matrices requires $\mathcal{O}(a^3)$ multiplications of complex numbers in general. Neglecting the computational costs for the partial trace and matrix exponential and assuming a constant time for each multiplication, the total time complexity can be summarized as

$$t_{\mathrm{tot}} = \mathcal{O}\left(N_{\mathrm{iter}} m n^2 2^{6n}\right), \tag{8}$$

where only the samples at distance $d = 1$ are included in the training set. This complexity is independent of whether or not self-connections are removed. It does however restrict us to classical simulations of small systems with $n \leq 5$ only.

3.3 Evaluating the Performance

Different HNNs with different training and updating schemes can be compared by the capacity of the HNN, an important measure to estimate the performance as an associative memory. The capacity relates to the maximum number of storable patterns, which requires some measure of the number of retrieval errors. We give three types of errors, each decreasingly strict in assigning errors. The proposed error rates are the *strict*, *message* and *bit* error rates and are given by:

$$\mathrm{SER}_{n,m} := 1 - \mathbf{1}\Big[\forall p \in \{1,\ldots,m\},\ \forall j \in \{1,\ldots,n\} :\ \boldsymbol{x}_j^{(p)} = f(\boldsymbol{x}^{(p)})_j\Big], \tag{9}$$

$$\mathrm{MER}_{n,m,\eta} := \frac{1}{m N_{\mathrm{vic}}} \sum_{p=1}^{m} \sum_{p_k=1}^{N_{\mathrm{vic}}} \left(1 - \mathbf{1}[\forall j \in \{1,\ldots,n\} : \boldsymbol{x}_j^{(p)} = f(\boldsymbol{x}_\eta^{(p_k)})_j]\right), \tag{10}$$

$$\mathrm{BER}_{n,m,\eta} := \frac{1}{m N_{\mathrm{vic}}} \sum_{p=1}^{m} \sum_{p_k=1}^{N_{\mathrm{vic}}} \frac{1}{n} H\left(\boldsymbol{x}^{(p)}, f(\boldsymbol{x}_\eta^{(p_k)})\right). \tag{11}$$

Here, n is the input size and m the number of distinct stored patterns, with $m \leq 2^n$. The memory patterns are chosen randomly. Furthermore, $\mathbf{1}[\cdot]$ is the indicator function, which is one if its argument is true, and zero otherwise.

The SER (Eq. (9)) only considers the patterns the HNN should memorize and equals one if at least one bit of any memory pattern cannot be retrieved. This definition corresponds to the one given in [9]. The MER (Eq. (10)) is less strict and uses N_{vic} noisy probe vectors $\boldsymbol{x}_\eta^{(p_k)}$ for each memory $\boldsymbol{x}^{(p)}$. These probe vectors are random noisy versions of the memory patterns, generated with noise parameter η. The MER equals the fraction of the probe vectors from which $\boldsymbol{x}^{(p)}$ cannot be recovered exactly. Finally, the BER (Eq. (11)) also uses the probe vectors $\boldsymbol{x}_\eta^{(p_k)}$. The BER considers all bits separately that cannot be retrieved correctly. For $\eta = 0$, these three error rates are decreasingly strict: $\text{SER} \geq \text{MER} \geq \text{BER}$.

For error rates $\text{ER} \in \{\text{SER}, \text{MER}, \text{BER}\}$ and threshold t, we estimate the number of storable patterns in an HNN using

$$m_{\max}^t(n, ER) \coloneqq \max\{m \in \{1, \ldots, 2^n\} \mid ER(n, m) \leq t\}. \tag{12}$$

The capacity C_t of an HNN is now given by normalizing $m_{\max}^t$ by n. For large system sizes, $m_{\max}^t(n, \text{SER})$ is independent of t. For classical HNNs with Hebbian learning the capacity is given by $n/(4 \log n)$.

The capacity of an HNN cannot be determined accurately with only a single set of random memory patterns. Therefore, the error rate for r different random sets of memory patterns are averaged to better approximate the error rates. Note that we require the learned patterns to be stable states of the network and that other states in the vicinity should be attracted. Furthermore, SER is not an appropriate measure for the attractiveness of memory patterns. This follows as the noisy probe samples are randomly generated and might therefore not be part of the memories basin of attraction.

Memories may contain some patterns multiple times, for instance due to small system sizes. For such memories, effectively fewer patterns are stored. Therefore, we generate memories at random, but require them to be distinct.

3.4 Simulation Methods

We simulate both the classical and the quantum HNN using matrix multiplications. The classical simulation is implemented straightforwardly by applying the updating rule for neurons and the Hebbian learning scheme to estimate the weights. For the quantum HNN, the unitaries U_j are initialized uniformly at random and updated according to Eq. (3), where the matrices K_j are estimated from Eq. (4) and (5). Quantum measurements are implemented by choosing the most likely outcome based on the probabilities calculated from the output states, in case of several equally likely outcomes an error is assigned in general. The code used for our simulations is available at [10].

The learning rate $1/\lambda$ introduced in Eq. (4) can be chosen freely and controls the step width of updates. We chose $\lambda = 1$ based on an estimation of the MER with varying $\lambda \in [0.01, 50]$ for system size $n = 4$, $N_{\text{train}} = 50$ training iterations and $r = 100$ repetitions.

We train the unitaries in 50 training iterations on r randomly generated sets of memory patterns. For each set, we estimate the three error rates of retrieval when presenting the memories as input to the trained quantum model. The training sets include all samples at a distance $d \leq 1$ around the respective memory patterns. We repeat this estimation with systems of size $n \leq 5$ and $m \leq 2^n$ memories for all r runs. We sample the error rates $r = 500$ times for $n \in \{3, 4\}$, and up to 1200 times for $n = 5$, to reduce the confidence intervals to a reasonable level.

4 Results

We present the results for the error rates for noisy input retrieval and the capacity of both the quantum HNN (Sect. 4.1 and Sect. 4.2) and the classical HNN (Sect. 4.3 and Sect. 4.4). We end by comparing the results for both in Sect. 4.5. All results are presented with 99% confidence intervals (CI).

4.1 Error Rates of Retrieving Memory Patterns

The error rates when presenting stored memories as input states are displayed in Fig. 3 for system sizes $n \in \{3, 4, 5\}$. The error rates are averaged over the corresponding r rounds. In all simulations, the error rates are zero for $m = 1$ and increase monotonically with m. The SER increases quickly for small m and reaches one at $m = 5$ ($n = 3$), $m = 7$ ($n = 4$) and $m = 9$ ($n = 5$). The MER increases moderately and does not reach one, but settles at around 0.7 for $n = 3$ and $n = 4$. The BER increases the least of all rates and remains below 0.2 for all considered systems.

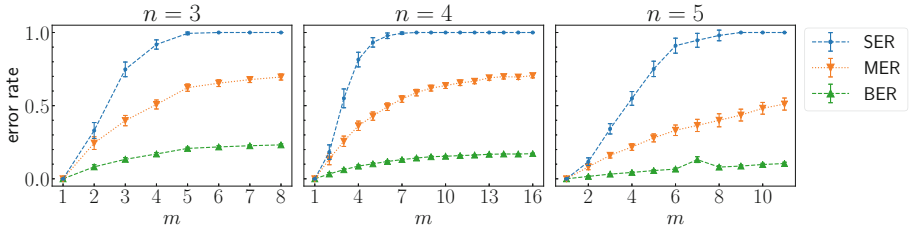


Fig. 3. Estimated SER, MER and BER versus the number of stored patterns m for different system sizes $n = 3$, 4, 5 for a quantum HNN.

The noisy input samples are generated with noise rates $\eta \in \{0.1, 0.2, 0.3\}$ and performance is evaluated for BER and MER. The results are shown in Fig. 4,

together with the noiseless results for $\eta = 0$. We find that both the MER and BER monotonically increase with m. Even for $m = 2^n$ and noise rate $\eta = 0.3$, the BER remains below 0.3 and the MER below 0.85 in all considered cases.

For all m, the differences between the error rates for different noise rates remain approximately constant. We notice that the MER for $\eta = 0$ and $\eta = 0.1$ are within the range of each other's confidence intervals for almost all m. For $n = 5$, the CIs are increasingly large due to the varying number of repetitions.

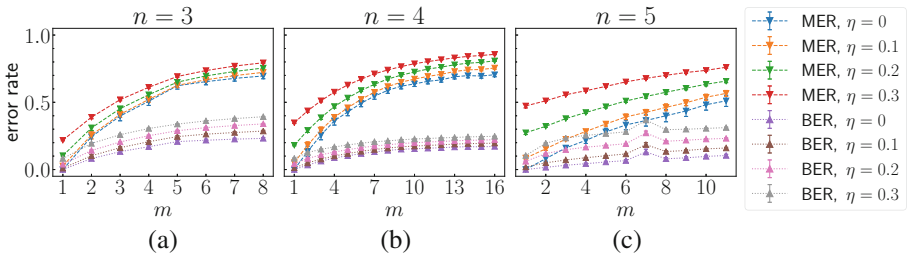


Fig. 4. Estimated MER and BER of retrieving the correct memory patterns from noisy inputs versus the number of stored patterns m. The considered system sizes are (a) $n = 3$, (b) $n = 4$ and (c) $n = 5$.

4.2 Capacity of the Quantum Model

Based on the error rate estimations, we estimate the ability of our quantum model to store and retrieve patterns. The estimated maximum numbers of storable patterns $m^t_{\max}$ are given in Table 1 for error rates SER and MER and thresholds $t = 0$ and $t = 0.1$. For this, the point estimates of both error rates and their CIs are compared to the thresholds. Only for $n = 5$, there are several m-values with confidence intervals that contain error rates below the threshold values. Hence, not all $m^t_{\max}$ can be estimated with certainty, and therefore all possible values are indicated by curly brackets.

Table 1. Maximum number of storable patterns $m^t_{\max}(\cdot)$ when presenting memories as inputs, for SER and MER and thresholds $t \in \{0, 0.1\}$. To obtain all $m^t_{\max}$, the 99% confidence intervals around error rates are considered.

n	$m^0_{\max}$(SER)	$m^{0.1}_{\max}$(SER)	$m^0_{\max}$(MER)	$m^{0.1}_{\max}$(MER)
3	1	1	1	1
4	1	1	1	1
5	1	{1, 2}	1	{1, 2}

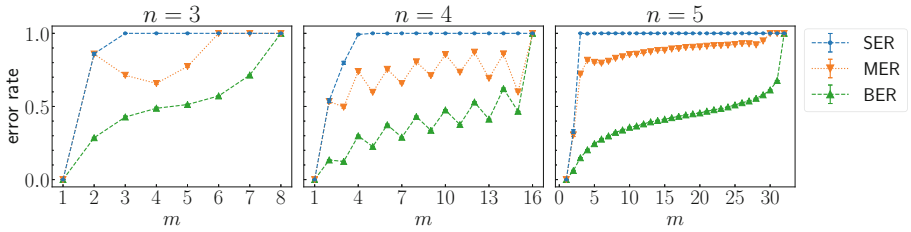


Fig. 5. Estimated SER, MER and BER when presenting the stored patterns to the classical HNN versus their number m for different system sizes n. The stored patterns are required to be distinct.

4.3 Error Rates of Retrieving Memory Patterns Classically

We estimate the error rates for retrieving memory patterns with classical HNNs. For each fixed n and $1 \leq m \leq 2^n$, we generate $r = 10^4$ sets of memories at random. Each of the three error rates are estimated for $n \in \{3, 4, 5\}$ and the memory patterns as inputs, the results are shown in Fig. 5. We find that with increasing number of patterns m, the error rates increase as well. All error rates are exactly zero for $m = 1$ and one for $m = 2^n$. For even n, both the MER and SER fluctuate for different m and are higher if m is even. In contrast, for odd n we see a smooth evolution. The SER increases to 1 rapidly for all n. The results for MER are similar to those for the SER. The BER stays well below the other error rates and increases only moderately, before reaching unity for $m = 2^n$.

When presenting noisy input states to the HNN, we see different behavior. As in the quantum case, only the MER and BER are estimated. For each memory pattern, we generate $N_{vic} = 100$ noisy samples with the same noise rates η as before. The results for different system sizes n are shown in Fig. 6. The different noise rates are indicated by different colors. Again we see less fluctuations for increasing n. Errors increase earlier in the noisy case than in the noiseless case, as expected.

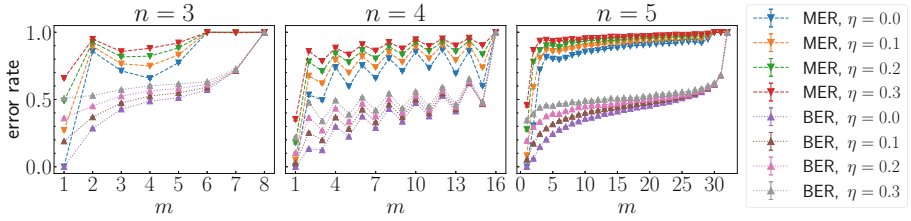
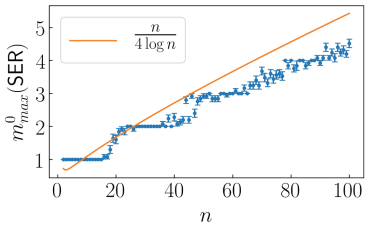


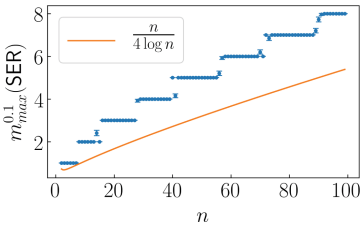
Fig. 6. Estimated MER and BER versus the number of stored patterns m for system sizes $n \in \{3, 4, 5\}$ when presenting noisy test samples as inputs to the classical HNN. The test samples are generated with noise rates $\eta \in \{0.1, 0.2, 0.3\}$. Additionally, the error rates for presenting the noiseless memories are shown.

4.4 Capacity of Classical HNNs with Hebbian Learning

We evaluate $m^t_{\max}$ for 100 iterations and in each iteration we estimate the error rates using $r = 10^4$ randomly chosen sets of distinct memories for different m and n. We consider the strict error rates in this analysis. In Fig. 7a and 7b the results are shown for thresholds $t = 0$ and $t = 0.1$. The results for the MER are similar to those of the SER. The theoretical capacity $n/(4 \log n)$ is shown as an orange line. We see a step-wise behavior for all shown results and we see that the results for $t = 0$ correspond relatively well with the theoretical limit.



(a) Threshold $t = 0$



(b) Threshold $t = 0$

Fig. 7. Estimated $m^t_{\max}(n, \text{SER})$ for thresholds (a) $t = 0$ and (b) $t = 0.1$. The obtained values are based on $r = 10^4$ runs with random sets of memories to estimate the SER for a classical HNN with Hebbian learning. The asymptotic limit of retrievable patterns is displayed by the orange curve $n/(4 \log n)$. (Color figure online)

4.5 Comparison of Classical and Trained Quantum HNN

We compare the error rates, which are estimated when presenting the memory patterns to the respective model, for the classical HNN with Hebbian learning and our trained quantum model in Fig. 8. The considered system sizes are $n \in \{3, 4, 5\}$. For $n = 5$, we have only few data for the quantum model due to the computational cost, such that a comparison can be only made for $m \leq 11$.

For all n, the MER and BER of the quantum model are smaller than for the classical HNN. The only exception is for $n = 4$ and $m = 15$, where the MER of the classical model is smaller. We also find that the SER of the quantum model is smaller than the classical SER for small m and reaches one only for higher values of m. While the MER and BER fluctuate like a saw-tooth for even n for the classical HNN, we do not find this behavior for the quantum model.

In Table 1 the maximum number of stored patterns without errors is given for the trained quantum model. Based on the results in Fig. 8, we see that the classical HNN with Hebbian learning can only store one pattern reliably.

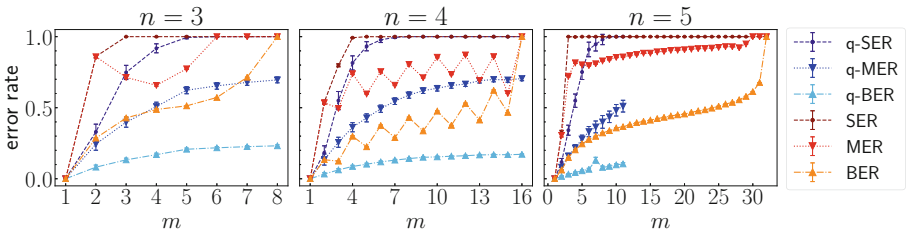


Fig. 8. Comparison of estimated error rates versus the number of stored patterns m for different system sizes $n \in \{3, 4, 5\}$. The blue (red) data are achieved using our trained quantum model (classical HNN with Hebbian learning). (Color figure online)

5 Discussion

In contrast to the case of classical HNNs with Hebbian learning, both MER and BER remain well below one even for $m = 2^n$ for our quantum model. This is reasonable, as it is possible that there are invariant sub-spaces of quantum states, that are not affected by the trained quantum channel. Even if all possible input states are considered as memories, a small number of them can remain invariant under the channel action and thus yield a retrieval error rate less than one.

The estimated error rates for noisy inputs for a quantum HNN stay well below the results for the classical HNN with Hebbian learning and they increase slower. However, when comparing the relative increase in error rate for noiseless and noisy patterns, the classical and the quantum HNN score roughly the same.

Within the level of confidence obtained with the results, we can conclude that our quantum model can store more memories than the classical HNN using Hebbian learning. Already for $n > 4$ it is likely that the quantum model can store more than one memory given that SER or MER are below $t = 0.1$, whereas the classical model can only store a single memory reliably.

The capacity estimates for the classical HNN with Hebbian learning follow the theoretical optimal curve. Due to the high computational costs of the simulations, these results are unavailable for the quantum HNN. Based on the shown results, we do expect capacity improvements for the quantum model over the classical theoretical optimum.

The high computational costs of the simulations of the quantum model originate from the exponential complexity given in Sect. 3.2. This in turn results in very limited system sizes we can simulate. Nonetheless, simulating larger systems in sufficiently many repetitions is valuable, because it allows us to compare the number of stored patterns to other implementations of HNNs.

The presented model can be implemented on general quantum devices and an implementation would require $3n + 1$ qubits and $n4^n$ multi-qubit gates.

6 Conclusion and Outlook

In this work we consider classical HNNs with Hebbian learning and quantum HNNs, where the unitaries are explicitly trained. Based on the presented results, we conclude that the quantum HNN can indeed be used to store classical bit strings as stable attractors with a higher capacity than classical HNNs.

Using a numerical analysis, we consider the number of randomly chosen bit strings that can be stored by an associative model. For $n = 5$ we found that the number of storable patterns is one or two, given an error rate threshold of 0.1, whereas only a single pattern can be stored using a classical HNN with Hebbian learning. For threshold zero, the storage capacity for small system sizes is equal for both classical and quantum HNNs.

It is possible to implement the trained quantum model on actual quantum devices, requiring $3n + 1$ qubits. This might even allow for faster evaluation of the training scheme due to fast execution times on quantum devices. This would allow testing of the trained quantum model on larger systems than in our simulations. However, the number of required gate parameters of the algorithm has a similar scaling as the time complexity when implemented straightforwardly. Therefore, we expect that the scaling prevents experimental realizations of much larger systems.

We conclude that the trained quantum model of our work should be understood as a toy example on the path towards a quantum algorithm for associative memories with possibly larger capacity. The achievement of a quantum advantage by increasing the storage capacity of quantum neural networks beyond classical limits is far from obvious, and more research is required.

Although only classical inputs have been considered, the presented quantum models can also be used for quantum data as inputs and outputs. The ability of our model to store and retrieve quantum states should be studied in future research. We suggest comparing our trained quantum model to classical algorithms that involve non-static training schemes for HNNs, i.e., where the weights are optimized on a training set with respect to a cost-function. In this way, it can be clarified experimentally, whether the better performance of the quantum model originates purely from the fact that it is trained, or from an actual quantum advantage over classical schemes. Moreover, we propose to analyze the storage capacity of our model theoretically, both for quantum and classical memory states. In this way, we hope to find an answer to the ultimate question of whether a quantum advantage can be achieved in the storage capacity of neural networks.

References

1. Arute, F., et al.: Quantum supremacy using a programmable superconducting processor. Nature **574**, 505–510 (2019)
2. Beer, K., Bondarenko, D., Farrelly, T., Osborne, T.J., Salzmann, R., Wolf, R.: Efficient learning for deep quantum neural networks. arXiv:1902.10445, February 2019

3. Cabrera, E., Sossa, H.: Generating exponentially stable states for a Hopfield neural network. Neurocomputing **275**, 358–365 (2018)
4. Dunjko, V., Briegel, H.J.: Machine learning and artificial intelligence in the quantum domain: a review of recent progress. Rep. Prog. Phys. **81**(7), 074001 (2018)
5. Feynman, R.P.: Quantum mechanical computers. Opt. News **11**(2), 11 (1985)
6. Harrow, A.W., Montanaro, A.: Quantum computational supremacy. Nature **549**(7671), 203–209 (2017)
7. Hilbert, M., Lopez, P.: The world's technological capacity to store, communicate, and compute information. Science **332**(6025), 60–65 (2011)
8. Hopfield, J.J.: Neural networks and physical systems with emergent collective computational abilities. Proc. Natl. Acad. Sci. **79**(8), 2554–2558 (1982)
9. McEliece, R., Posner, E., Rodemich, E., Venkatesh, S.: The capacity of the Hopfield associative memory. IEEE Trans. Inf. Theory **33**(4), 461–482 (1987)
10. Meinhardt, N.: NMeinhardt/QuantumHNN 1.0 (Version 1.0). Zenodo (2019), 11 April 2020. https://doi.org/10.5281/zenodo.3748421
11. Mitarai, K., Negoro, M., Kitagawa, M., Fujii, K.: Quantum circuit learning. Phys. Rev. A **98**(3), 032309 (2018)
12. Neumann, N., Phillipson, F., Versluis, R.: Machine learning in the quantum era. Digitale Welt **3**(2), 24–29 (2019). https://doi.org/10.1007/s42354-019-0164-0
13. Preskill, J.: Quantum computing and the entanglement frontier. In: 25th Solvay Conference on Physics, March 2012
14. Rebentrost, P., Bromley, T.R., Weedbrook, C., Lloyd, S.: Quantum Hopfield neural network. Phys. Rev. A **98**(4), 042308 (2018)
15. Rojas, R.: Neural Networks. Springer, Heidelberg (1996). https://doi.org/10.1007/978-3-642-61068-4
16. Schuld, M., Sinayskiy, I., Petruccione, F.: The quest for a quantum neural network. Quantum Inf. Process. **13**(11), 2567–2586 (2014). https://doi.org/10.1007/s11128-014-0809-8
17. Schuld, M., Sinayskiy, I., Petruccione, F.: Simulating a perceptron on a quantum computer. Phys. Lett. A **379**(7), 660–663 (2015)
18. Ventura, D., Martinez, T.: Quantum associative memory with exponential capacity. In: 1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98CH36227), vol. 1, pp. 509–513. IEEE (2002)