Applying a Quantum Annealing Based Restricted Boltzmann Machine for MNIST Handwritten Digit Classification

Mateusz Slysz, Marek Subocz, Krzysztof Kurowski

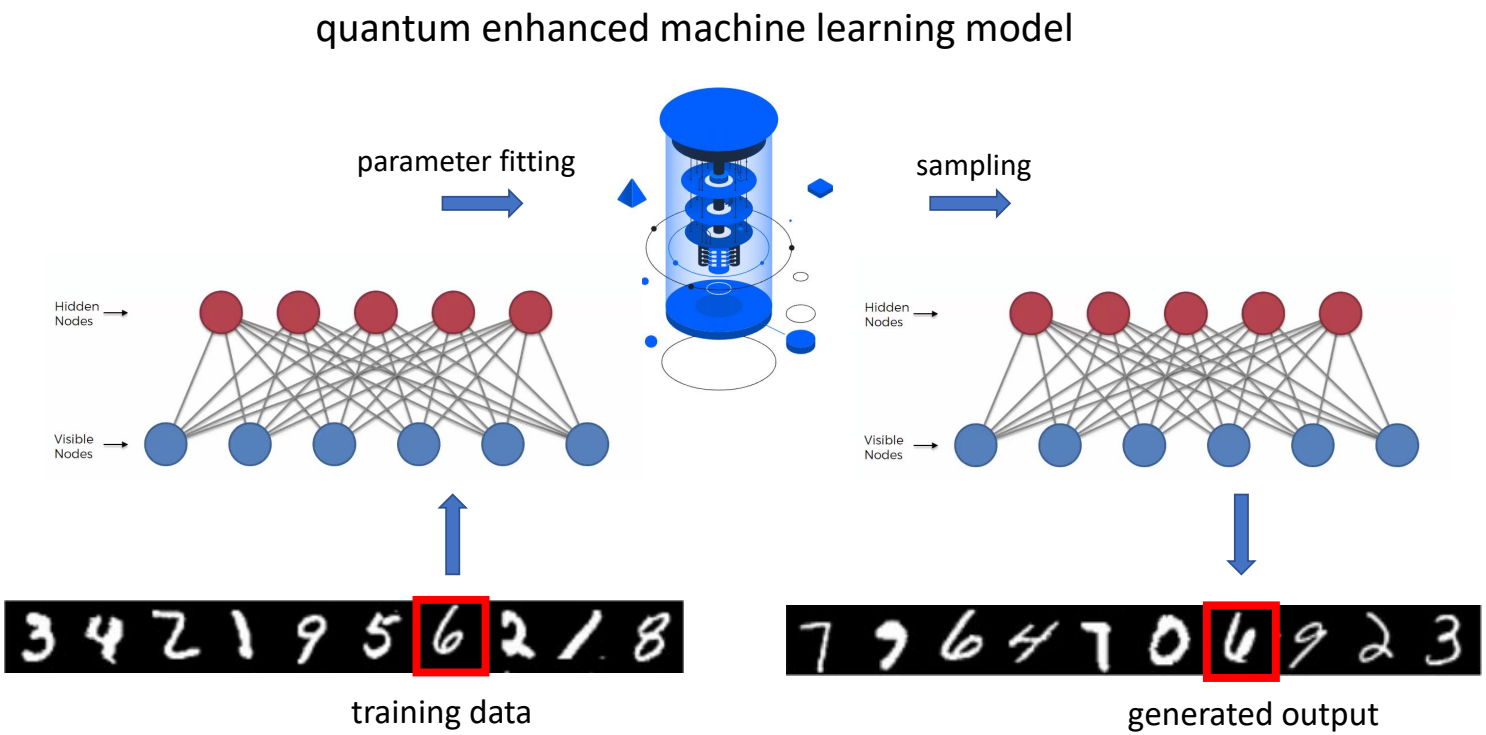
Poznan Supercomputing and Networking Center

May 10, 2022

Motivations

- How can we use quantum computing/quantum annealing in a wide range of pattern recognition tasks supported by machine learning?
- What are the main limitations today in terms of problem size, solutions quality, speed of different quantum computing architectures?
- How do hybrid (quantum-classical) algorithms compare to classical approaches for simple and more complicated real-world problems?

Motivations



What is a Restricted Boltzmann Machine (RBM)

- Restricted Boltzmann Machine (RBM) is a machine learning model introduced by Paul Smolensky in 1986, and rose to prominence after Geoffrey Hinton and collaborators invented fast learning algorithms for them in the mid-2000.
- RBM is a stochastic, generative machine learning model that can learn the underlying probability distribution of a given dataset.
- Unsupervised learning algorithms like RBMs are commonly used for feature extraction, dimensionality reduction, topic modeling or generating new data samples e.g. images, videos or voice samples.
- Learning an RBM corresponds to fitting its parameters such that the distribution represented by the RBM models the distribution underlying the training dataset.

Definition of an RBM

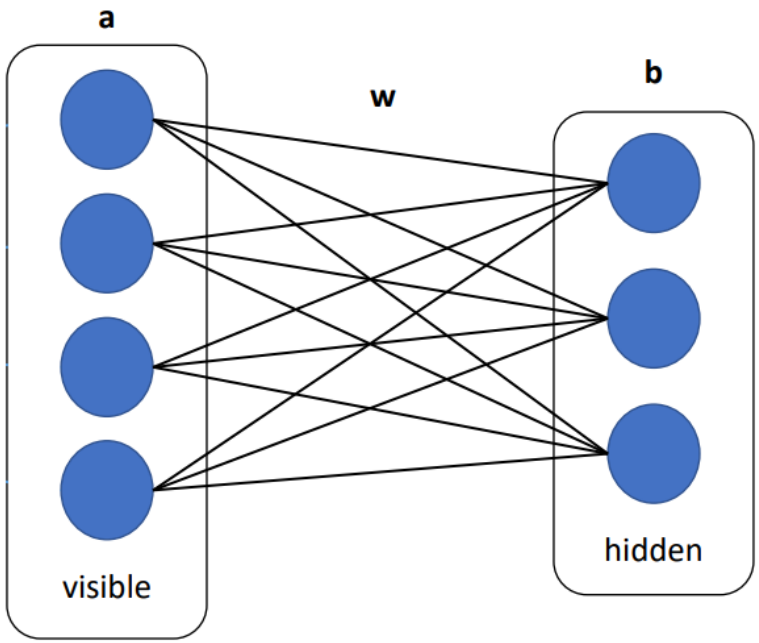
An RBM is a complete bipartite graph with two groups of nodes called visible $\boldsymbol{v}$ and hidden $\boldsymbol{h}$.

Nodes in both visible and hidden layers can take binary values.

$$\forall_{v_i \in \boldsymbol{v}} v_i \in \{0, 1\}$$

$$\forall_{h_j \in \boldsymbol{h}} h_j \in \{0, 1\}$$

The visible layer (of size n) corresponds to the data points, while the hidden layer (of size m) can be treated like an unknown random variable.

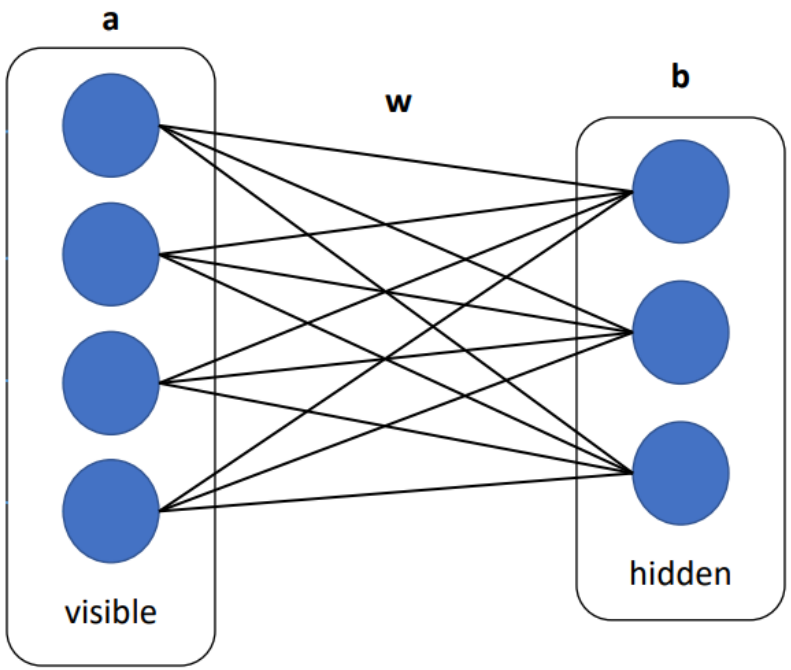


Definition of an RBM

Each node from the visible layer is connected by a weighted connection with each node in the hidden layer.

Each edge connecting nodes i and j corresponds to a weight W_{ij}. Each node in the visible layer corresponds to a bias value a_i and each node in the hidden layer corresponds to a bias value b_j.

Weight matrix $\boldsymbol{W}$ of size $n \times m$ and bias vectors $\boldsymbol{a}$ of size n and $\boldsymbol{b}$ of size m are parameters subject to learning.



Energy function of an RBM

An RBM is an energy-based, probabilistic model, which means that there is a scalar value assigned to each possible state.
The energy function for an RBM is given by the following equation:

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} W_{ij} v_i h_j$$

Probability value of a given state $(\boldsymbol{v}, \boldsymbol{h})$ is described as:

$$p(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z} e^{-E((\boldsymbol{v}, \boldsymbol{h}))}$$

where Z is a partition function, which in general is **hard to compute**.

$$Z = \sum_{(\boldsymbol{v}, \boldsymbol{h})} e^{-E((\boldsymbol{v}, \boldsymbol{h}))}$$

as the number of all possible states grows exponentially with RBM size $(O(2^{n+m}))$.

RBM learning process - classical assumptions

Classically, this limitation has been evaded by assuming independence of variables.

$$p(\boldsymbol{h}|\boldsymbol{v}) = \prod_j p(h_j|\boldsymbol{v})$$

$$p(\boldsymbol{v}|\boldsymbol{h}) = \prod_i p(v_i|\boldsymbol{h})$$

Under this assumption, given the values from the visible layer (e.g. the training data), much simpler formulas for the conditional probabilities of the layers can be derived:

$$p(h_j = 1|\boldsymbol{v}) = \sigma(b_j + \sum_i W_{ij} v_i)$$

$$p(v_i = 1|\boldsymbol{h}) = \sigma(a_i + \sum_j W_{ij} h_j)$$

where σ denotes the Sigmoid function, commonly used as an activation function in feed forward artificial neural networks, and is defined as follows:

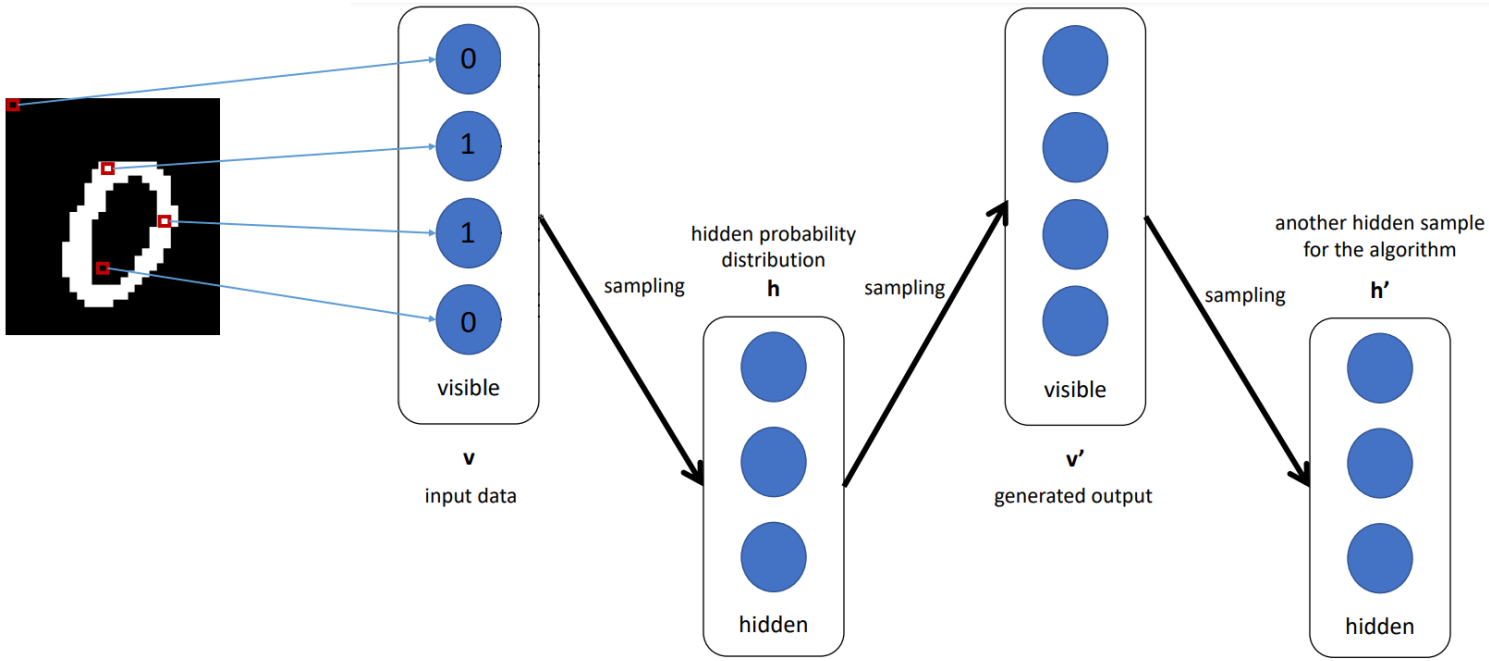
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

RBM learning algorithm - Contrastive Divergence

Using these properties, we can propose a classical learning algorithm, which alternately samples both visible and hidden layers based on the results of the previous iteration.

Starting from the input of the learning data $\boldsymbol{x}$ given to the layer $\boldsymbol{v}$, we are able to sample the layer $\boldsymbol{h}$, then on its basis the layer $\boldsymbol{v'}$ and further the layer $\boldsymbol{h'}$.

Continuing this process, one can iteratively reach $\boldsymbol{v^{(t)}}, \boldsymbol{h^{(t)}}$ for any number of iterations t.



RBM learning algorithm - training objective

Our goal is for the RBM to return results similar to data points from the training set in subsequent iterations t.

For this purpose, we need to define a loss function which maximizes the probability that each pixel in the generated vector $\boldsymbol{v}^{(t)}$ is equal to the training example $\boldsymbol{x}$.

Equivalently minimizing the the negative log likelihood function seems to be the natural choice for the cost function L:

$$L(\boldsymbol{x}) = \frac{1}{T} \sum_{t}^{T} - \log p(\boldsymbol{v}^{(t)} = \boldsymbol{x})$$

RBM learning algorithm - Stochastic Gradient Descent

To minimize the cost function we used a well known algorithm called Stochastic Gradient Descent (SGD).

The idea of SGD is to take repeated small steps in the opposite direction of the gradient of the function at the current point, because this is the direction of the steepest descent.

The gradient of the loss function must be computed w.r.t. model parameters θ. In this case, the model parameters are $\boldsymbol{W}$, $\boldsymbol{a}$ and $\boldsymbol{b}$.

$$\frac{\partial L}{\partial \theta} = \frac{\partial \left(-\log p\left(\boldsymbol{v}\right)\right)}{\partial \theta} = \mathbb{E}_{\boldsymbol{h}}\left[\frac{\partial E(\boldsymbol{v}, \boldsymbol{h})}{\partial \theta} \,\middle|\, \boldsymbol{v} = \boldsymbol{x}\right] - \mathbb{E}_{\boldsymbol{v},\boldsymbol{h}}\left[\frac{\partial E(\boldsymbol{v}, \boldsymbol{h})}{\partial \theta}\right]$$

Parameter update rule

$$\frac{\partial\left(-\log p\left(\boldsymbol{v}\right)\right)}{\partial\theta} = \mathbb{E}_{\boldsymbol{h}}\left[\frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial\theta}\,\middle|\,\boldsymbol{v}=\boldsymbol{x}\right] - \mathbb{E}_{\boldsymbol{v},\boldsymbol{h}}\left[\frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial\theta}\right]$$

The equation for the gradient consists of two parts called positive and negative phases.

The first part dependents on the input data and for parameter $\theta = W_{ij}$ can be calculated as $h_j \cdot v_i$.

The second part dependents on the model and is hard to compute classically. However, using the Gibbs sampling method introduced earlier we can approximate this value by the expression $h_j^{(t)} \cdot v_i^{(t)}$ after t sampling steps.

Parameter update rule

The update rule for the $\boldsymbol{W}$ matrix is then:

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k + \alpha \left(\boldsymbol{h} \cdot \boldsymbol{v}^T - \boldsymbol{h}^{(t)} \cdot \boldsymbol{v}^{(t)\,T} \right)_k$$

where α is the learning rate. Analogously, the update rule for the bias vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ are:

$$\boldsymbol{a}_{k+1} = \boldsymbol{a}_k + \alpha \left(\boldsymbol{v} - \boldsymbol{v}^{(t)} \right)_k$$

$$\boldsymbol{b}_{k+1} = \boldsymbol{b}_k + \alpha \left(\boldsymbol{h} - \boldsymbol{h}^{(t)} \right)_k$$

Further extensions

In practice $t = 1$ is often used to reduce computation time, as it is empirically shown that it is often enough for the training process, however this take may take more iterations to achieve convergence.

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k + \alpha \left(\boldsymbol{h} \cdot \boldsymbol{v}^T - \boldsymbol{h}' \cdot \boldsymbol{v}'^T \right)_k$$

$$\boldsymbol{a}_{k+1} = \boldsymbol{a}_k + \alpha \left(\boldsymbol{v} - \boldsymbol{v}' \right)_k$$

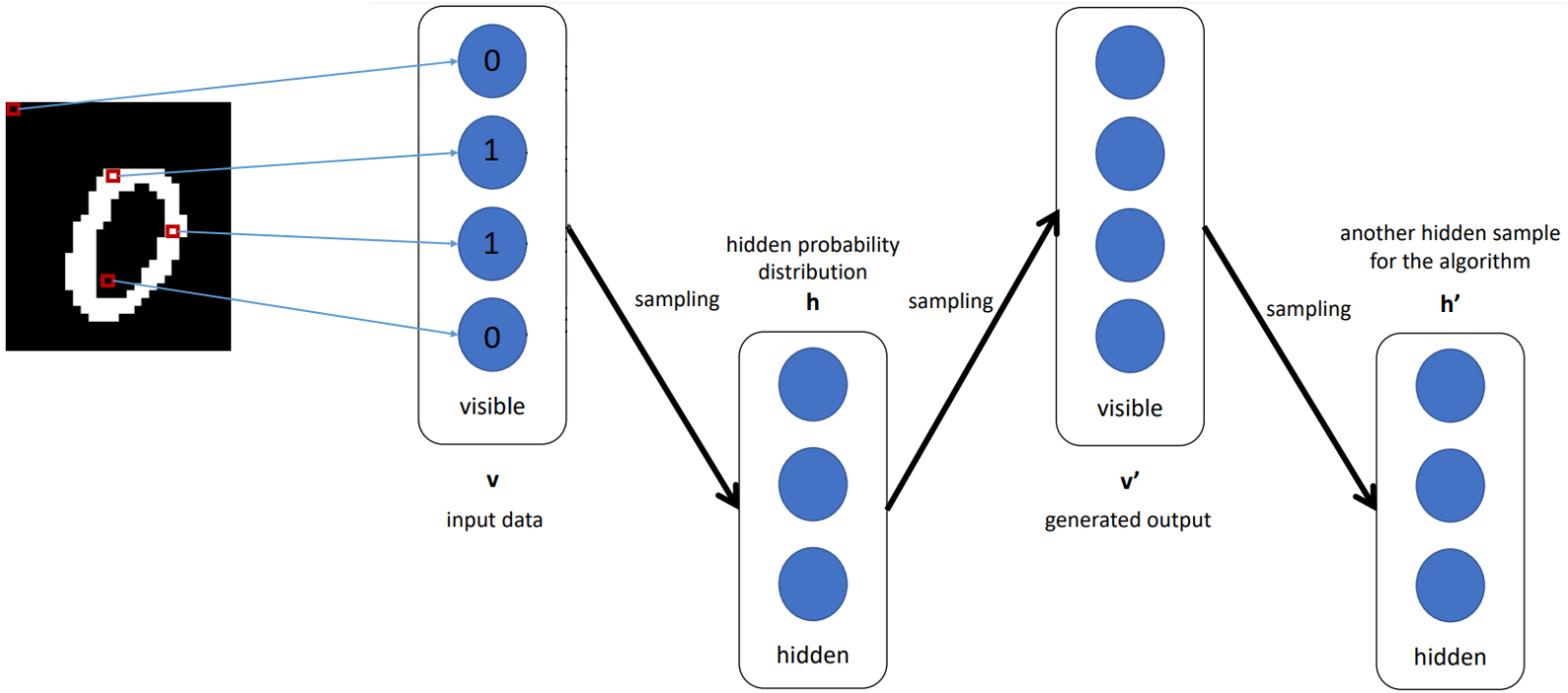
$$\boldsymbol{b}_{k+1} = \boldsymbol{b}_k + \alpha \left(\boldsymbol{h} - \boldsymbol{h}' \right)_k$$

Furthermore, one can use more advance machine learning techniques to enhance the learning process such as:

- gradient with momentum,
- learning rate decay,
- early stopping,
- persistent-CD.

RBM learning algorithm - summary

1. Enter input data $\boldsymbol{x}$ onto the visible layer $\boldsymbol{v}$.
2. Conditionally sample $\boldsymbol{h}$, conditionally sample $\boldsymbol{v'}$, conditionally sample $\boldsymbol{h'}$.
3. Calculate the positive and negative phases of the gradient.
4. Update model parameters W, a and b.



Quantum Extension to the RBM (QRBM)

Problems of the classical RBM:

- Partition function Z hard to compute.
- Assume independence of variables.
- Inaccurate sampling using classical methods.

The main idea is to replace the sampling step with a quantum annealing based sampling. This should be a natural step as RBM is an energy based model, so quantum annealing fits perfectly into the basic concept of the algorithm.

- Sample from the actual probability distribution.
- No additional assumptions.
- An ideal random number generator.

QUBO

RBM's energy function:

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} W_{ij} v_i h_j$$

D-wave's Quantum Annealing device provides an efficient way to find minimum of such a function. In order to do that, a QUBO (Quadratic Unconstrained Binary Optimization) equation needs to be constructed:

$$f(x) = \sum_i Q_{i,i} x_i + \sum_{i<j} Q_{i,j} x_i x_j$$

The quantum annealing is a physical process that can efficiently traverse the energy landscape by taking advantage of effects such as quantum tunneling to find the global minimum of an energy function.

RBM - changes to the algorithm

Instead of classically sampling $\boldsymbol{h}$, $\boldsymbol{v'}$, $\boldsymbol{h'}$ using the conditional probabilities

$$p(h_j = 1|\boldsymbol{v}) = \sigma(b_j + \sum_i W_{ij} v_i)$$

$$p(v_i = 1|\boldsymbol{h}) = \sigma(a_i + \sum_j W_{ij} h_j)$$

we can sample the probability distribution with the quantum annealer (D-Wave). In each step we define the Q matrix depending on the model parameters.
To sample $\boldsymbol{h}$:

$$Q = -\sum_j^m \left(\left(\sum_i^n (W_{ij} \cdot v_i) + b_j \right) \cdot h_j \right)$$

To sample $\boldsymbol{v}$:

$$Q = -\sum_i^n \left(\left(\sum_j^m (W_{ij} \cdot h_j) + a_i \right) \cdot v_i \right)$$

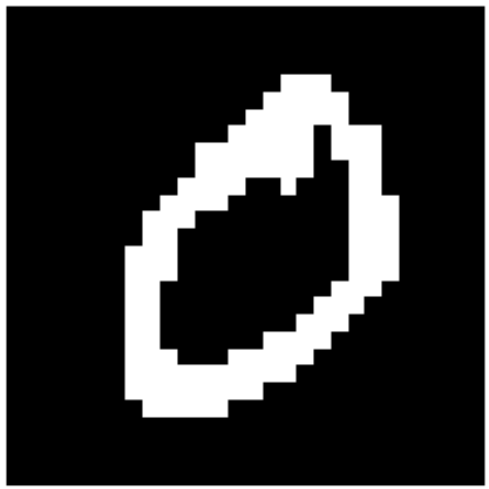
MNIST dataset

We tested our model on the well-known MNIST dataset. MNIST is a dataset of handwritten digit images with 60000 training samples and 10000 testing samples. It is a popular machine learning benchmark dataset which consists of images of size 28×28 pixels each.

Preprocessing

Before feeding the data to the model, we had to perform some pre-processing steps, due to the fact that an RBM can only process binary data.

We transformed the original MNIST images to binary representation, using a threshold of 100 pixel brightness out of $0 \div 255$ greyscale.



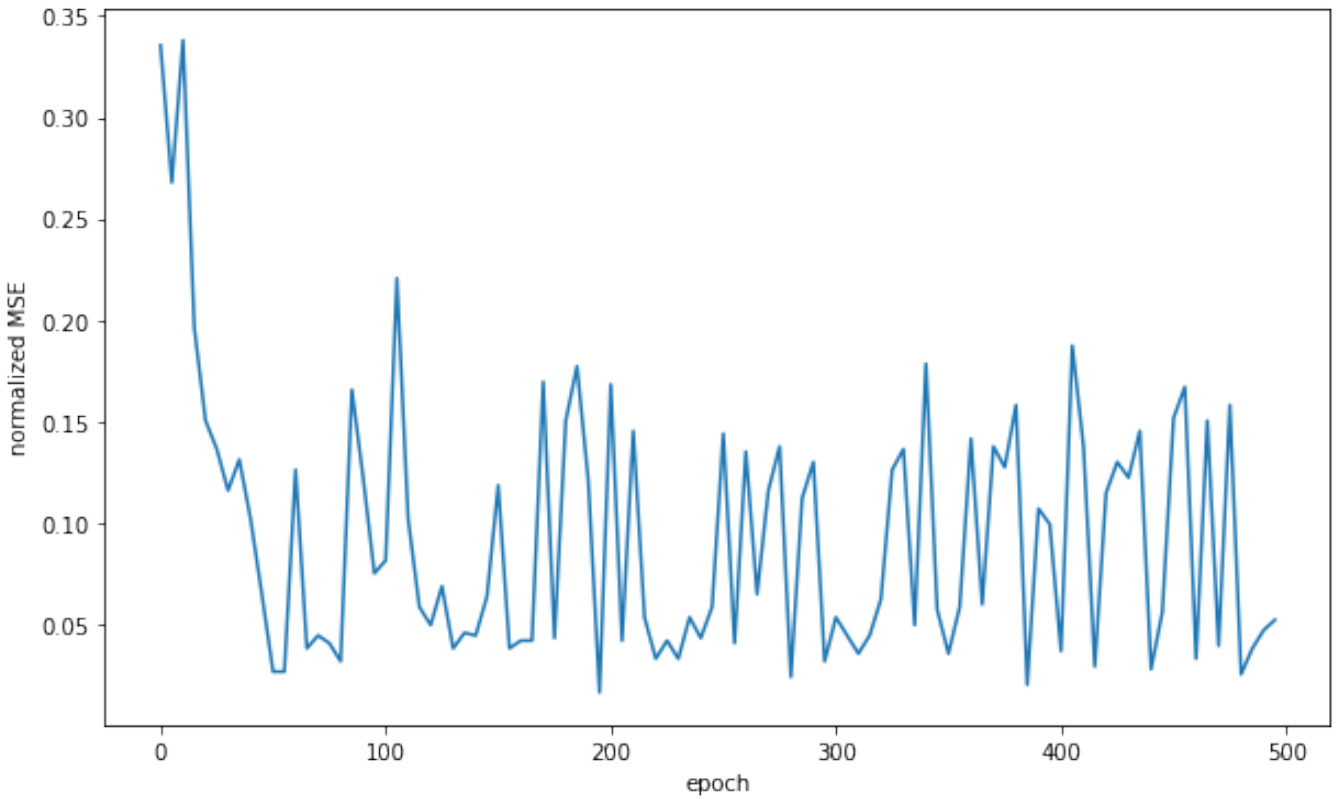
Next, we had to flatten the images, so they could fit onto the 1-dimensional RBM visible layer of length $n = 28 \times 28 = 784$.

Result quality measure - MSE

To monitor the quality of training processes, we used a popular MSE (Mean Squared Error) measure between the input image which activates the sampling process and the generated image. The two images were compared pixel by pixel and the MSE value was then normalized by dividing it by the image size to fit between 0 and 1.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

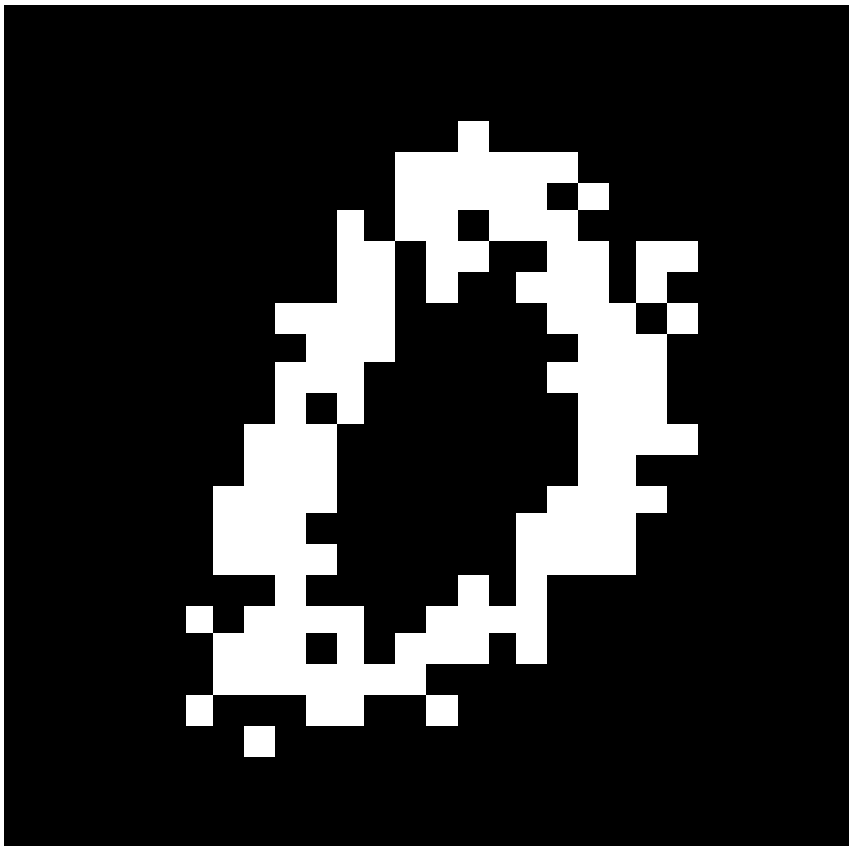
Learning process



The plot has an L-shaped curve, typical for machine learning problems, which drops rapidly at the beginning of the learning process and then gradually decreases in the remaining epochs. However, there are some fluctuations because the MSE is not an exact measure of the quality of the model.

Basic experiments - single digit '0'

After 500 epochs the MSE error was not decreasing further, so we assumed that the Quantum RBM (QRBM) has been trained. Example digit '0' generated by the trained QRBM with the MSE $= 0.067$.



Classical comparison

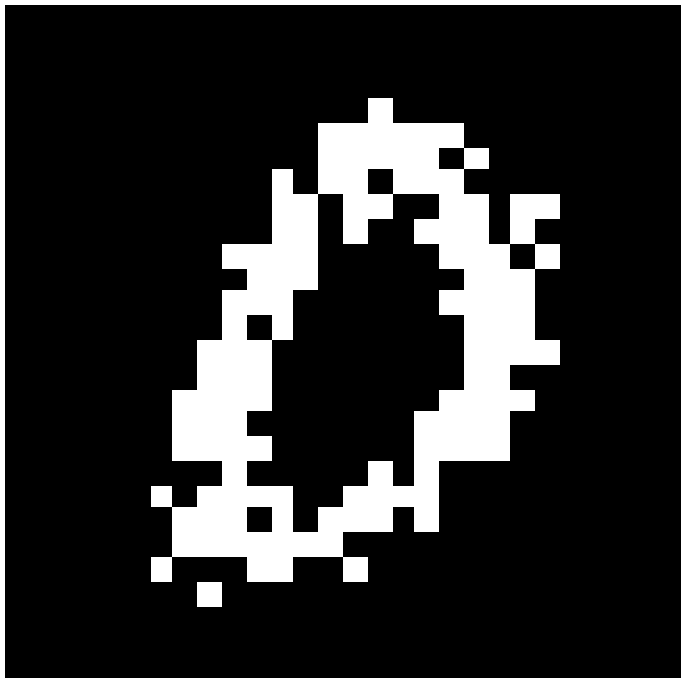


Image generated by the QRBM.

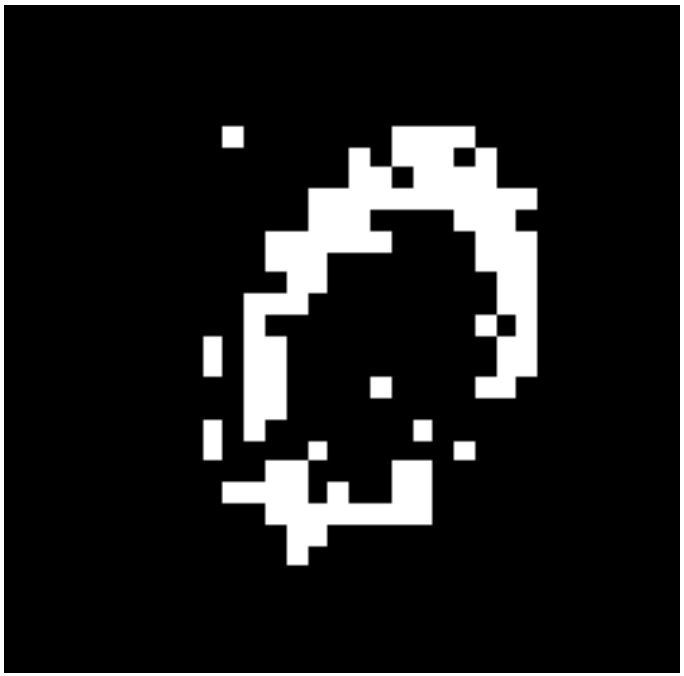


Image generated by the classical RBM (from the scikit-learn library) of the same size and hyperparameters.

Further experiments - two digits

After obtaining promising results with low MSE errors, we wanted to further enhance the model by adding more digits to the training dataset. For example, for training the QRBM on a dataset containing 2 digits '0' and '1' we got results.

Image reconstructed after training

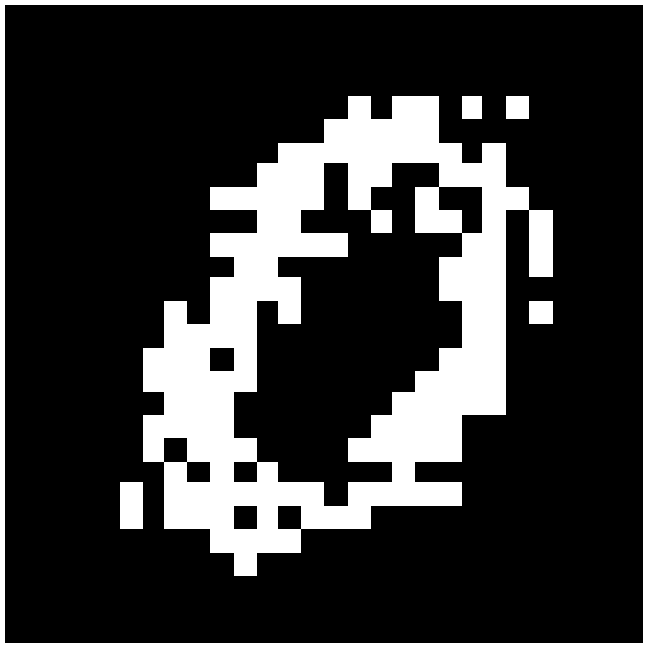
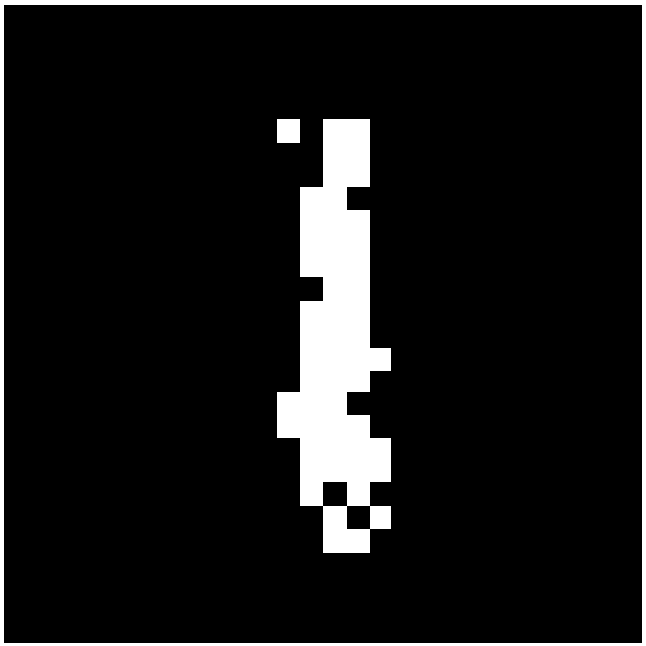
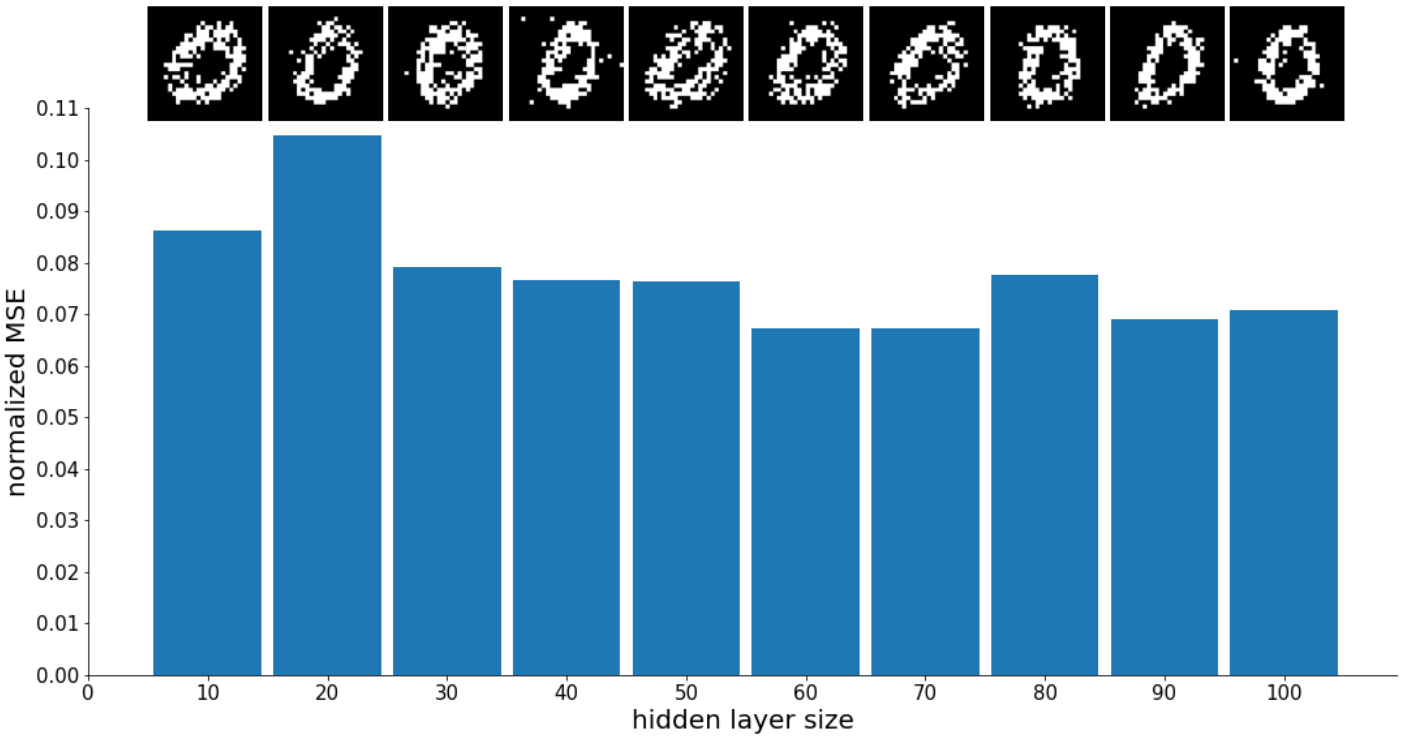


Image reconstructed after training



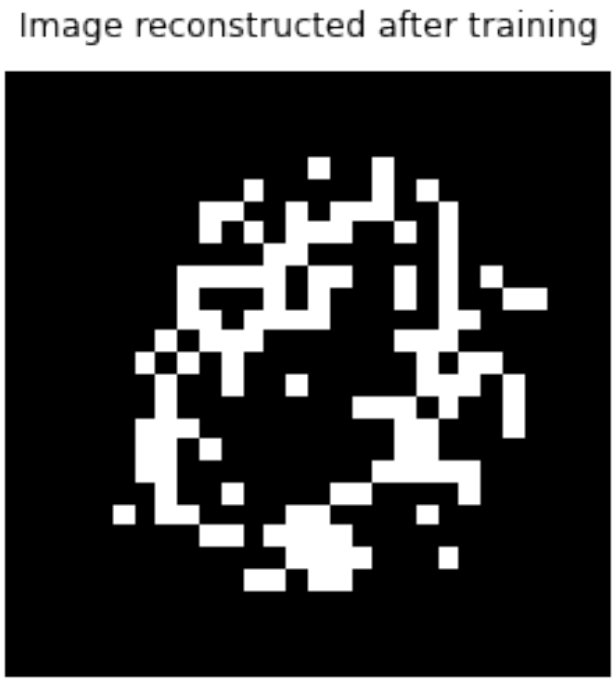
Further experiments - hidden layer size

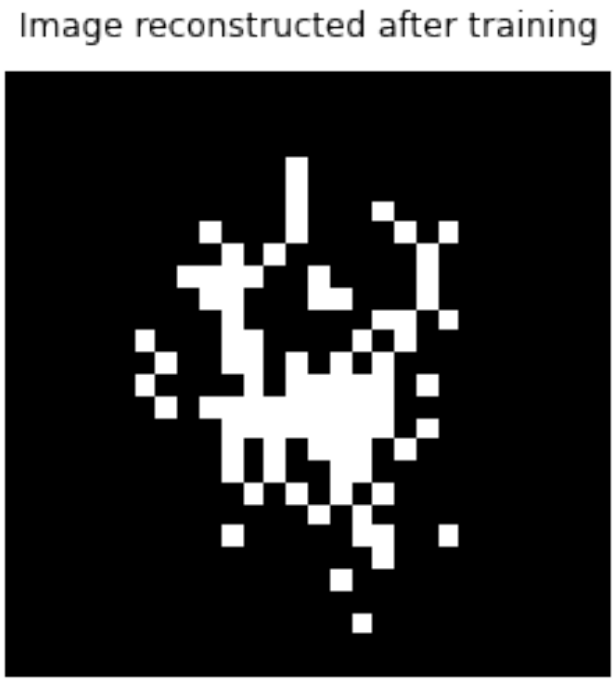
For the 2 digit dataset we experimented with the size of the hidden layer. For next experiments, we chose $m = 60$ for the hidden layer size, as it returned the best normalised MSE rate.



Further experiments - more digits

However, after increasing the number of digits in the dataset the output images are becoming more and more blurry. The result aquired from the model trained on a dataset containing all 10 digits were hard to identify.

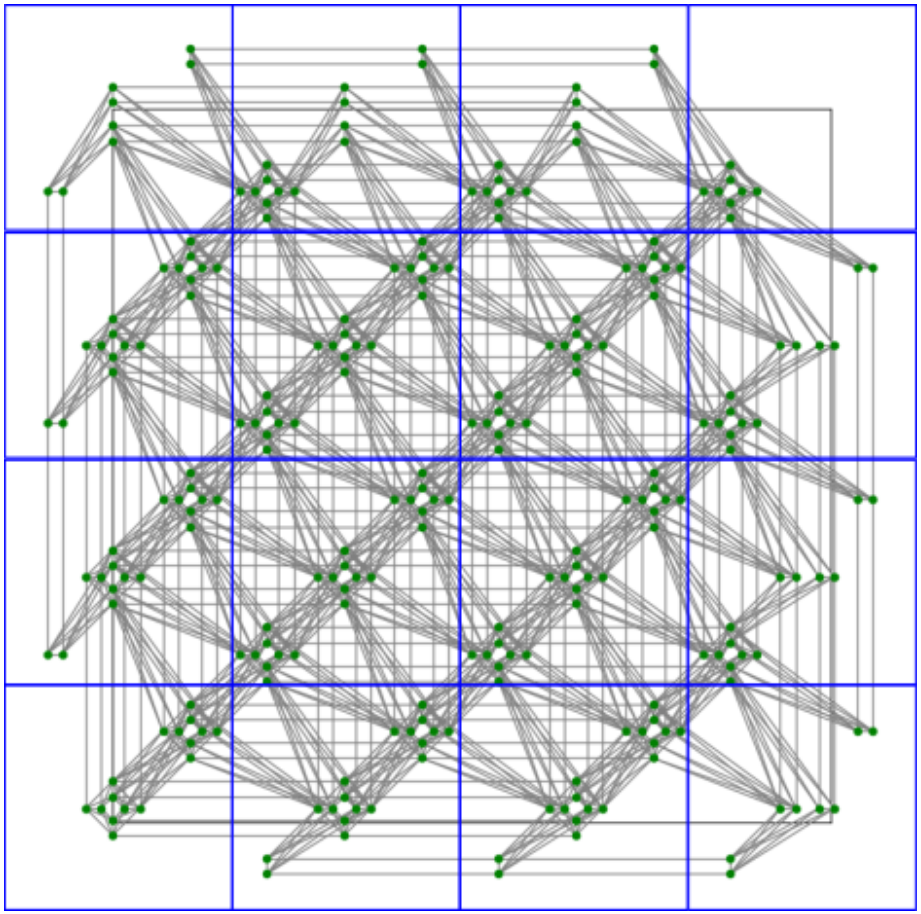




The reason behind this was that the model was overfitting, as choosing the sample with the lowest energy was correlated with an image which is a mix of all 10 digits.

Solution - chain strength

The *chain strength* is a crucial control parameter available in D-Wave architecture responsible for ensuring that results follow given restrictions. When a chain connects two qubits, they are supposed to have the same binary value. If the opposite is true, the chain is broken, which may lead to suboptimal results.

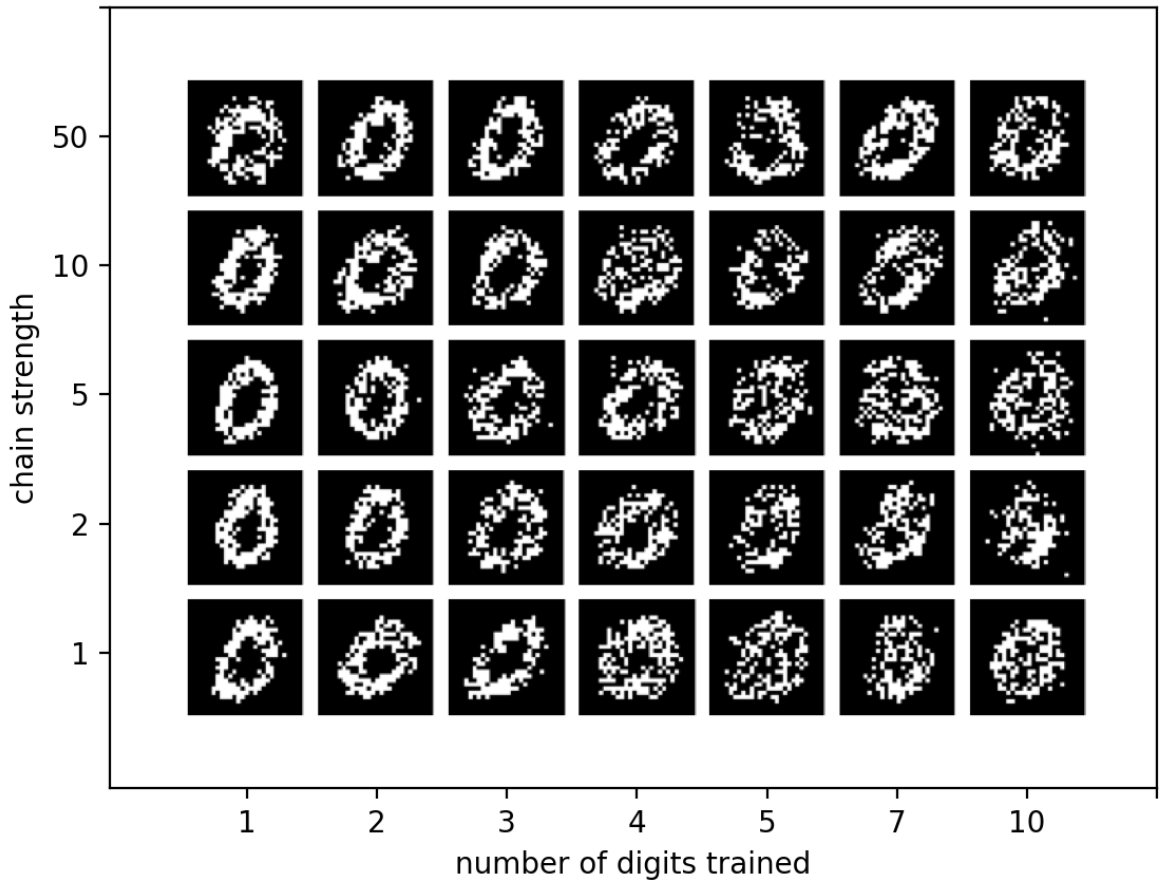


Chain strength

All QUBO weights are autoscaled to values between -1 and 1 together with the chain strength value. Increasing the chain strength may result in a more distributed range of solutions, as original punishments become less precise and more prone to thermal noise.

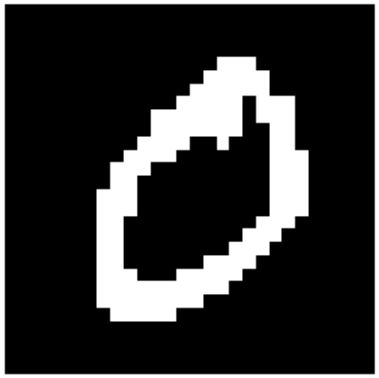
Usually, D-Wave's quantum annealer is used as a solver for optimization problems, which can quickly find a (sub)optimal solution by choosing a sample with the lowest energy. However, we are interested in the whole energy landscape describing the probability distribution rather than just the lowest energy eigenstate.

Chain strength - experimental results

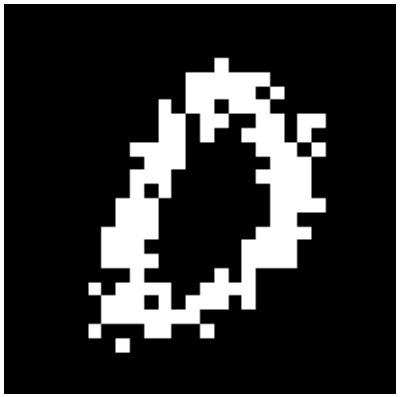


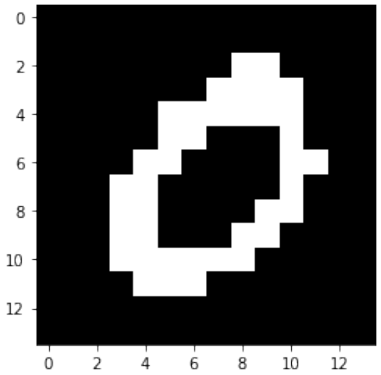
When trained on a low number of digits, the algorithm yields the best results with a low chain strength value. When the number of digits grows, it turns out that higher chain strength values perform better.

Experimental results D-Wave 2000Q vs Advantage

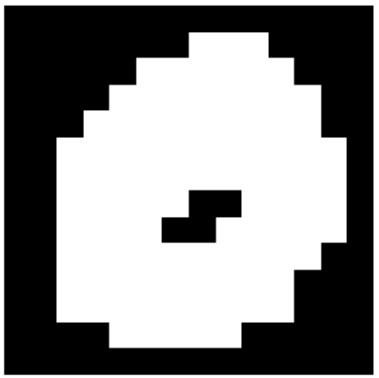


Standard resolution image (28×28) on D-Wave Advantage.





Low resolution image (14×14) on D-Wave 2000Q. The hidden layer size was also restricted to $m \approx 50$.



Conclusions

- We proposed, implemented, trained and validated experimentally a Quantum Enhanced RBM (QRBM) on the MNIST dataset.
- The advantage of using the D-Wave's quantum annealer, is that we can quickly acquire samples from a complicated probability distribution, without further assumptions for the model.
- For low number of different digits in the training set, the QRBM gave promising results as it generated images of digits with low MSE score.
- However, increasing the number of digits caused the model to overfit. We solved this problem, by tuning the chain strength parameter, which effected in getting a more evenly distributed sample.
- Our research additionally dives into the topic of tuning other RBM hyperparameters, e.g. by conducting experiments on the best hidden layer size, or applying advanced machine learning techniques like momentum or learning rate decay.
- We also compared the differences in capabilities of D-Wave's architecture between the old 2000Q and the new Advantage systems. The Advantage system allows more variables to fit on a quantum computer, thanks to the greater connectivity of the Pegasus graph.

Kurowski Krzysztof, Slysz Mateusz, Subocz Marek, Różycki Rafał.

Applying a quantum annealing based restricted boltzmann machine for mnist handwritten digit classification.

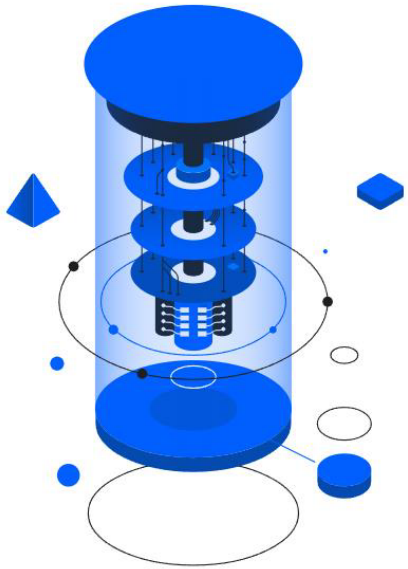
CMST, 27(3):99–107, 2021.

QRBM - GitHub repo - code and notebook.

`https://github.com/mareksubocz/QRBM`.

PSNC IBM Quantum HUB

https://quantum.psnc.pl



First Workshop on Quantum Computing and Communication at PPAM'22 (September 11-14, 2022, Gdańsk, Poland), Call for papers/presentations, extended deadline 20 May 2022. More information at: https://quantum.psnc.pl/wqcc22/