Training convolutional neural networks by simulated/quantum annealing

F. Sorba

November 17, 2025

We consider the possibility of training convolutional neural networks (CNN) by simulated or quantum annealing. This is realized by mapping the mean square error (MSE) loss function of each individual network's layer to a set of spin glasses Hamiltonians. Finding the ground state of the spin glass system is then equivalent to minimizing the layer loss function.

1 Linear layer

Let start by considering a network composed by a single linear layer:

$$s_i(q) = \sum_{j=0}^{N-1} \omega_{ij} q_j - \theta_i \,, \tag{1}$$

where q_i, $(i = 0, \dots N-1)$ are the network input neurons, s_i, $(i = 0, \dots M-1)$ the network output neurons, ω_{ij} the synaptic weights, θ_i the biases and $f(x)$ the activation function.

Training the layer means finding the configuration $\{\omega_{ij}, \theta_i\}$ that minimize a loss function over a training dataset. We are interested in the Mean Squared Error (MSE) loss function which is given by:

$$\mathrm{MSE}_i = \frac{1}{Q} \sum_{a=0}^{Q-1} \left(s_i(q^a) - \xi_i^a \right)^2 , \qquad i = 0 \dots M-1 \,, \tag{2}$$

where q^a is the a-th training sample and ξ^a the corresponding expected output. Setting the biases to zero ($\theta_i = 0$) for simplicity, we can insert Eq. (1) into Eq. (2) and expand the square as follow

$$\begin{aligned} \mathrm{MSE}_i &= \frac{1}{Q} \sum_{a=0}^{Q-1} \left[\left(\sum_{j=0}^{N-1} \omega_{ij} q_j^a \right)^2 - 2 \left(\sum_{j=0}^{N-1} \omega_{ij} q_j^a \right) \xi_i^a + (\xi_i^a)^2 \right] = \\ &= \left[\sum_{jl} \left(\frac{1}{Q} \sum_a q_j^a q_l^a \right) \omega_{ij}\omega_{il} + \sum_j \left(-\frac{2}{Q} \sum_a \xi_i^a q_j^a \right) \omega_{ij} + \frac{1}{Q} \sum_a (\xi_i^a)^2 \right] . \end{aligned} \tag{3}$$

By identifying

$$J_{jl} \equiv \frac{1}{Q}\sum_{a=0}^{Q-1} q_j^a q_l^a\,, \qquad h_j^i \equiv -\frac{2}{Q}\sum_{a=0}^{Q-1} \xi_i^a q_j^a\,,$$
$$E_0^i \equiv \frac{1}{Q}\sum_{a=0}^{Q-1} (\xi_i^a)^2\,, \qquad S_j^i \equiv \omega_{ij}\,, \tag{4}$$

the loss function has been mapped to a system of equations that resemble spin glass Hamiltonians:

$$\mathrm{MSE}_i \equiv H_i(\vec{S}) = \sum_{jl=0}^{N-1} J_{jl} S_j^i S_l^i + \sum_{j=0}^{N-1} h_j^i S_j^i + E_0^i\,. \tag{5}$$

However, the weights S_j^i are here continuous variables instead of spin variables ($\sigma_j^i = \pm 1$).

To convert them to spin variables we can choose a range around the current weight values S_j^{0i}, i.e. $S_j^i \in [S_j^{0i} - L, S_j^{0i} + L]$ and map these spans to the interval $[0,1]$[1]

$$S_j^i \in [S_j^{0i} - L, S_j^{0i} + L] \longrightarrow S_j^{'i} = \frac{S_j^i + L - S_j^{0i}}{2L} \in [0,1]\,,$$
$$S_j^i = L(2S_j^{'i} - 1) + S_j^{0i}\,. \tag{6}$$

By casting these normalized weights to an n-bit representation

$$S_j^{'i} = \frac{1}{N_b}\sum_{\alpha=0}^{n-1} 2^\alpha \frac{\sigma_{j\alpha}^i + 1}{2}\,, \qquad \sigma_{j\alpha}^i = \pm 1\,, \qquad N_b = 2^n - 1\,, \tag{7}$$

we can rewrite the Hamiltonian (5) in terms of spin variables $\sigma_{j\alpha}^i$. In this representation, the Hamiltonian's interaction J_{jl} and magnetic field h_j^i become

$$J_{jl} \to J_{jl}^{'\alpha\beta} = \frac{L^2}{N_b^2} J_{jl} 2^\alpha 2^\beta\,,$$
$$h_j^i \to h_j^{'i\alpha} = \frac{L}{N_b}\left(h_j^i + 2\sum_l J_{jl} S_l^{0i}\right) 2^\alpha\,. \tag{8}$$

Note that we do not need to compute the expression of the zero energy $E_0^{'i}$ because it does not affect the system ground state configuration.

Finally, combining the input neuron's index $j = 0 \dots N-1$ with the corresponding bit index $\alpha = 0 \dots n-1$ into a single index $\mu = 0 \dots N_L - 1$ whit $N_L = Nn$, we obtain the system of spin glasses Hamiltonians

$$\mathrm{MSE}_i \equiv H_i(\vec{\sigma}) = \sum_{\mu\nu=0}^{N_L-1} J'_{\mu\nu}\sigma_\mu^i \sigma_\nu^i + \sum_{\mu=0}^{N_L-1} h_\mu^{'i}\sigma_\mu^i + E_0^{'i}\,, \qquad i = 0 \dots M-1\,. \tag{9}$$

[1]This is better than choosing a global range $S_j^i \in [-L, L]$ because it allows us to set different range sizes L at different training epochs (similarly to varying the learning rate in backpropagation).

1.1 Biases

To keep things simple, we have set the biases θ_i to zero (and we will keep them so for the rest of this document). However, we want to point out that the above derivation is valid also in the presence of biases which can be interpreted as additional spin variables.

When expanding the MSE loss function, the biases contribution is

$$\mathrm{MSE}_i(\theta) = \frac{1}{Q}\sum_a \left(\theta_i - 2\sum_{j=0}^{N-1}\omega_{ij}\sigma_j^a + 2\xi_i^a\right)\theta_i\,. \tag{10}$$

By defining

$$S_N^i = \theta_i\,, \qquad J_{NN} = 1\,, \qquad J_{Nj} = -\frac{1}{Q}\sum_a \sigma_j^a\,, \qquad h_N^i = \frac{2}{Q}\sum_a \xi_i^a\,, \tag{11}$$

The total MSE loss function translates to the Hamiltonian

$$\mathrm{MSE}_i \equiv H_i(\vec{S}) = \sum_{jl=0}^{N} J_{jl}S_j^iS_l^i + \sum_{j=0}^{N} h_j^iS_j^i + E_0^i\,, \tag{12}$$

where the sums run now over $N+1$ weight variables instead of the N in Eq. (5).

2 Convolutional layer

The same approach outlined above for a linear layer can be carried out for convolutional layers as well. Let us consider a 2 dimensional convolution with C_1 input channels C_2 output channels and kernel c_{kl}^{pr} of size K:

$$s_{ij}^r = \sum_{p=0}^{C_1-1}\sum_{kl=0}^{K-1} c_{kl}^{rp} q^p_{i-\frac{K}{2}+k,j-\frac{K}{2}+l}\,, \qquad r = 0,\dots C_2-1\,, \tag{13}$$

where q_{ij}^p is the input layer and s_{ij}^r the output layer, and $i \in [\frac{K}{2}, N-\frac{K}{2}]$, $j \in [\frac{K}{2}, M-\frac{K}{2}]$. The MSE loss function is

$$\mathrm{MSE}_r = \frac{1}{Q}\sum_{a=0}^{Q-1}\sum_{i=\frac{K}{2}}^{N-\frac{K}{2}}\sum_{j=\frac{K}{2}}^{M-\frac{K}{2}}\left(s_{ij}^r(q^a) - \xi_{ij}^{ra}\right)^2\,, \qquad r = 0\dots C_2-1\,, \tag{14}$$

where we have introduced a sum over the i,j input sites because the convolutional kernel does not depend on them. Expanding the loss function and defining

$$\begin{aligned}
J_{kl,mn}^{pp'} &\equiv \frac{1}{Q}\sum_a\sum_{ij} q^{pa}_{i-\frac{K}{2}+k,j-\frac{K}{2}+l}\, q^{p'a}_{i-\frac{K}{2}+m,j-\frac{K}{2}+n}\,, \qquad p,p' \in [0, C_1-1]\,,\\
h_{kl}^{rp} &\equiv -\frac{2}{Q}\sum_a\sum_{ij} q^{pa}_{i-\frac{K}{2}+k,j-\frac{K}{2}+l}\,\xi_{ij}^{ra}\,, \qquad r\in[0,C_2-1]\,, p\in[0,C_1-1]\,,\\
E_0^r &\equiv \frac{1}{Q}\sum_a\sum_{ij}(\xi_{ij}^{ra})^2\,, \qquad r\in[0,C_2-1]\,,
\end{aligned} \tag{15}$$

the loss function is mapped to the set of Hamiltonians

$$\mathrm{MSE}_r \equiv H_r(c) = \sum_{pkl} \sum_{p'mn} J^{pp'}_{kl,mn} c^{rp}_{kl} c^{rp'}_{mn} + \sum_{pkl} h^{rp}_{kl} c^{rp}_{kl} + E^r_0 \,. \tag{16}$$

As before, we can convert the continuous kernel variables c^{rp}_{kl} to spin variables:

$$H_r(\sigma) = \sum_{pkl\alpha} \sum_{p'mn\beta} J'^{pp'}_{kl\alpha,mn\beta} \sigma^{rp}_{kl\alpha} \sigma^{rp'}_{mn\beta} + \sum_{pkl\alpha} h'^{rp}_{kl\alpha} \sigma^{rp}_{kl\alpha} + E'^r_0 \,, \tag{17}$$

where

$$\begin{aligned} J'^{pp'}_{kl\alpha,mn\beta} &= \frac{L^2}{N_b^2} J^{pp'}_{kl,mn} 2^\alpha 2^\beta \,, \\ h'^{rp}_{kl\alpha} &= \frac{L}{N_b} \left(h^{rp}_{kl} + 2 \sum_{p'mn} J^{pp'}_{kl,mn} c^{rp'}_{mn} \right) 2^\alpha \,. \end{aligned} \tag{18}$$

Finally by combining the kernel indices $k, l \in [0, K-1]$, the input channel index $p \in [0, C_1 - 1]$ and the bit index $\alpha \in [0, n-1]$ into a single index $\mu \in [0, N_C - 1]$ with $N_C = K^2 C_1 n$ we obtain the system of spin glass Hamiltonians

$$\mathrm{MSE}_r \equiv H_r(\vec{\sigma}) = \sum_{\mu\nu=0}^{N_C-1} J'_{\mu\nu} \sigma^r_\mu \sigma^r_\nu + \sum_{\mu=0}^{N_C-1} h'^r_\mu \sigma^r_\mu + E'^r_0 \,, \qquad r = 0 \ldots C_2 - 1 \,. \tag{19}$$

3 Activation function

Including an activation function in the above approach can be easily done at least for a piecewise linear function. Consider for example the LeakyReLU activation function:

$$f(x) = \gamma(x) x \,, \qquad \gamma(x) = \begin{cases} 1 \,, & x \geq 0 \\ \alpha \,, & x < 0 \end{cases} \,.$$

By inserting it in Eq. (1), the activated layer output becomes

$$f\left(s_i(q^a)\right) = \sum_{j=0}^{N-1} \gamma^a_i \omega_{ij} q^a_j \,, \qquad \gamma^a_i = \gamma(s_i(q^a)) \,. \tag{20}$$

Expanding the MSE (2) with the activated neurons, the spin glass Hamiltonian parameters (Eq. (4)) become

$$\begin{aligned} J^i_{jl} &\equiv \frac{1}{Q} \sum_{a=0}^{Q-1} (\gamma^a_i)^2 q^a_j q^a_l \,, \qquad & h^i_j &\equiv -\frac{2}{Q} \sum_{a=0}^{Q-1} \gamma^a_i \xi^a_i q^a_j \,, \\ E^i_0 &\equiv \frac{1}{Q} \sum_{a=0}^{Q-1} (\xi^a_i)^2 \,, & S^i_j &\equiv \omega_{ij} \,. \end{aligned} \tag{21}$$

This allows us to carry on the simulated annealing by updating the Hamiltonian parameters as the spin configuration evolves.

4 Multilayer networks

The equivalence outlined above between MSE loss functions and spin glass Hamiltonians can be used directly to train single layer networks for which inputs and outputs are given by the elements of the training dataset.

But when considering multilayer networks, the ideal inputs and outputs of the internal hidden layers are unknown.

We assume then that it is possible to proceed in steps where at each step the dataset samples are propagated forward and backward through the network generating the input/output configurations for each layer, the layers Hamiltonians are minimized for the given configuration and the procedure is repeated until eventually the system converges to an optimal solution.

Note that in this approach, backward propagation requires to compute the true inverse of the network. Since the layers are in general not bijective, we have to resort to use the Moore-Penrose pseudoinverse. Furthermore, the inversion of convolutional layers requires to express the convolution in Fourier space where it acts as a matrix multiplication and the pseudoinverse can be computed.

To be more specific, the procedure we use to train multilayer networks is outlined by the following steps:

1. Initialize the network weights randomly
2. Given the training set $\{q^a, \xi^a\}$, propagate the output sample ξ^a backward through the (pseudo)inverse network.
3. For each layer in the network:
   - Propagate the training input samples q^a forward through the network up to the layer in question
   - Transform the layer to the equivalent system of spin glasses Hamiltonians.
   - Find the spin configuration that minimize the Hamiltonian
   - Convert back the spin configuration and update the layer's weights
4. Once all layers weights have been updated, recalculate the network (pseudo)inverse.
5. Repeat from step 2 until the system's energy reaches a minimum.

As an example, consider a 2 layers network composed of an output linear layer s_i with weights ω^1_{ij} and an input linear layer h_j with weights ω^0_{jk} activated by a leakyReLU function. The MSE is

$$\mathrm{MSE}_i = \frac{1}{Q}\sum_{a=0}^{Q-1}(T_i^a)^2 = \frac{1}{Q}\sum_{a=0}^{Q-1}\left(\sum_j \omega^1_{ij}\gamma_i^{1a}h_j^a - \xi_i^a\right)^2 , \tag{22}$$

where we have introduced the quantity T_i^a:

$$T_i^a = f(s_i(f(h(q^a)))) - \xi_i^a = \left(\sum_j \omega^1_{ij}\gamma_i^{1a}\sum_k \omega^0_{jk}\gamma_j^{0a}q_k^a\right) - \xi_i^a . \tag{23}$$

By applying the pseudo-inverse of the output layer $(\omega^1_{ij})^{-1}$, one gets

$$T^a_j = \sum_i (\omega^1)^{-1}_{ji} \frac{T^a_i}{\gamma^{1a}_i} = \sum_k \omega^0_{jk} \gamma^{0a}_j q^a_k - \xi'^a_j \,, \tag{24}$$

where

$$\xi'^a_j = \sum_i (\omega^1)^{-1}_{ji} \frac{\xi^a_i}{\gamma^{1a}_i} \,.$$

One can then use T^a_j to minimize the MSE_j for the input layer weights ω^0_{jk}. Then, use this weight configuration to calculate the input layer output

$$h^a_j = \sum_k \omega^0_{jk} \gamma^{0a}_j q^a_k \,,$$

and use it to minimize the MSE_i for the output layer weights ω^1_{ij}.[2]

Note that when minimizing the weight configuration ω^0_{jk} of the input layer, Eq. (24) shows that the input samples q^a_k are affected only by the activation factors γ^{0a}_j relative to the input layer. On the other hand, in the gradient descent approach, the gradient component relative to the input layer weight $\omega^0_{j'k'}$ contains contributions from both layers activation factors:

$$\frac{\partial \mathrm{MSE}_i}{\partial \omega^0_{j'k'}} = \frac{2}{Q} \sum_a \left(\sum_j \omega^1_{ij} \gamma^{1a}_i \sum_k \omega^0_{jk} \gamma^{0a}_j q^a_k - \xi^a_i \right) \omega^1_{ij'} \gamma^{1a}_i \gamma^{0a}_{j'} q^a_{k'} \,. \tag{25}$$

In order to keep a similar contribution in the spin glass approach, one can re-express Eq. (23) as

$$T^a_i = \sum_j T^a_{ij} \,, \tag{26}$$

where

$$T^a_{ij} = \omega^1_{ij} \gamma^{1a}_i \sum_k \omega^0_{jk} \gamma^{0a}_j q^a_k - \omega^1_{ij} \gamma^{1a}_i \sum_{i'} (\omega^1)^{-1}_{ji'} \frac{\xi^a_{i'}}{\gamma^{1a}_{i'}} \,. \tag{27}$$

When minimizing the input layer weight configuration, one can then use the expression

$$T^a_j = \sum_i T^a_{ij} = \sum_k \omega^0_{jk} \beta^{1a}_j \gamma^{0a}_j q^a_k - \xi''^a_j \,, \tag{28}$$

where

$$\beta^{1a}_j = \sum_i \omega^1_{ij} \gamma^{1a}_i \,, \qquad \xi''^a_j = \beta^{1a}_j \sum_{i'} (\omega^1)^{-1}_{ji'} \frac{\xi^a_{i'}}{\gamma^{1a}_{i'}} \,. \tag{29}$$

While now the input samples are affected by the activation factors from both layers, the approach does not seem to produce better results than the previous one.

[2]Note that this derivation is based on the assumption that $\sum_i (\omega^1)^{-1}_{ji} \omega^1_{ij'} = \delta_{jj'}$ which is not generally true for non-invertible matrices. However, tests shows that the procedure works well independently of the shape of the weight matrices.

5 Tests and results

In order to assess the performance of the approach described above, we compare it to standard backpropagation in training some simple network architectures for classification[3].

To minimize the spin glass Hamiltonians, we use simulated annealing and quantum annealing (when possible).

Simulated annealing is based on the Metropolis-Hastings algorithm where the system temperature is slowly lowered during the iteration. In the Metropolis algorithm, to find the spin configuration that minimizes the Hamiltonian (9), each spin σ_μ is flipped if it decreases the system energy, i.e. if

$$dE^i_\mu = H_i(\sigma'_\mu) - H_i(\sigma_\mu) = (\sigma'_\mu - \sigma_\mu)\left(2\sum_{\nu\neq\mu} J^i_{\mu\nu}\sigma_\nu + h^i_\mu\right) < 0\,, \tag{30}$$

or if a random number $r \in [0,1]$ is smaller than the acceptance probability

$$P_a = E^{-\beta dE^i_\mu}\,, \tag{31}$$

where $\beta \propto \frac{1}{T}$ is the inverse temperature.

Quantum annealing is performed on the D-wave quantum annealing machines.

The code of the tests is available here.

5.1 Single linear layer

Let's start by considering a network consisting of a single linear layer:

$$\boxed{\begin{array}{c}\text{Input}\\(1,32,32)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Flatten}\\(1024)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Linear}\\(1024\times 10)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Output}\\(10)\end{array}}\,. \tag{32}$$

When performing simulated annealing, we use a 8bit representation to convert the synaptic weights to spin variables so that the network is mapped to a system of 10 spin glasses each containing 8192 spins.

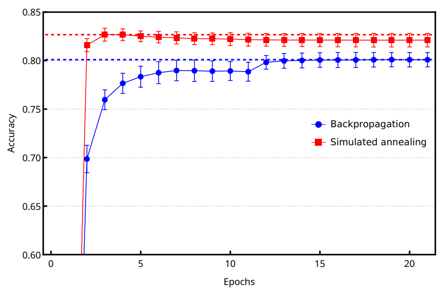
We train this network using a subset of the MNIST dataset consisting on 1000 samples.

In Fig. 1(a), we compare the mean accuracy over 20 epochs of training by backpropagation and simulated annealing (the model is too large to be trained by quantum annealing). The result shows that the two methods reach the same level of accuracy.

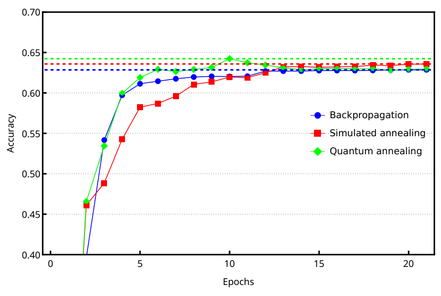
Since this network does not fit on the current D-wave architectures, we consider a smaller version in order to observe how quantum annealing performs. By inserting a pooling layer after the input we can reduce the number of weights in the Linear layer:

$$\boxed{\begin{array}{c}\text{Input}\\(1,32,32)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Spectralpool}\\(1,4,2)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Flatten}\\(8)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Linear}\\(8\times 10)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Output}\\(10)\end{array}}\,. \tag{33}$$

[3]For classification tasks, Cross Entropy loss function is usually preferred over MSE. However, for such simple datasets and architectures, the use of MSE loss function does not seem to negatively affect the training performance.



(a)



(b)

Figure 1: a) Comparison of accuracies obtained by training the 1-layer model (32) by backpropagation (blue) and by simulated annealing (red). b) Comparison of accuracies obtained by training the 1-layer model (33) by backpropagation (blue), simulated annealing (red) and quantum annealing (green).

By using a 4bit representation for the synaptic weights, the network is then mapped to system of 10 spin glasses each consisting of 32 spin variables.

Fig. 1(b) compares the accuracies over 20 epochs of training by backpropagation, simulated annealing and quantum annealing on 10000 samples of the MNIST dataset. The result shows that the three methods reach the same level of accuracy. It appears that no evident advantage arises by using quantum annealing (at least for such a simple system).

5.2 Multiple linear layers

We consider here a multilayer network consisting of 3 linear layers:

$$\begin{array}{c}\boxed{\begin{matrix}\text{Input}\\(1,32,32)\end{matrix}} \rightarrow \boxed{\begin{matrix}\text{Spectralpool}\\(1,8,8)\end{matrix}} \rightarrow \boxed{\begin{matrix}\text{Flatten}\\(64)\end{matrix}} \rightarrow \boxed{\begin{matrix}\text{Linear}\\(64\times 128)\end{matrix}}\\ \rightarrow \boxed{\begin{matrix}\text{Linear}\\(128\times 64)\end{matrix}} \rightarrow \boxed{\begin{matrix}\text{Linear}\\(64\times 10)\end{matrix}} \rightarrow \boxed{\begin{matrix}\text{Output}\\(10)\end{matrix}}\end{array} \tag{34}$$

The network is trained on a subset of the MNIST dataset of 1000 samples for 20 epochs. The accuracies for backpropagation and simulated annealing are plotted in Fig. 2(a).

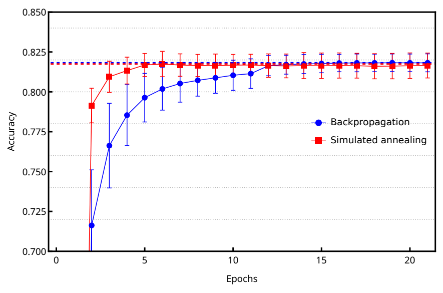
In order to test the performance of quantum annealing for multiple layers, again we consider a simplified architecture consisting of only 2 linear layer:

$$\boxed{\begin{matrix}\text{Input}\\(1,32,32)\end{matrix}} \rightarrow \boxed{\begin{matrix}\text{Maxpool}\\(1,4,2)\end{matrix}} \rightarrow \boxed{\begin{matrix}\text{Flatten}\\(8)\end{matrix}} \rightarrow \boxed{\begin{matrix}\text{Linear}\\(8\times 8)\end{matrix}} \rightarrow \boxed{\begin{matrix}\text{Linear}\\(8\times 10)\end{matrix}} \rightarrow \boxed{\begin{matrix}\text{Output}\\(10)\end{matrix}}. \tag{35}$$

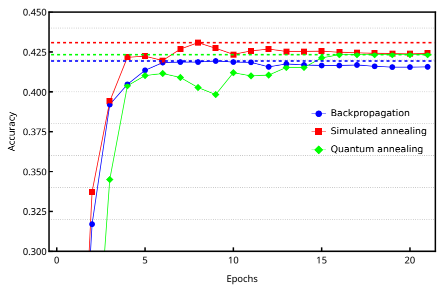
The results are shown in Fig. 2(b).

5.3 Convolutional layer

For testing how the training approach performs with convolutional layers, we consider a single layer and a three layers architecture. The single layer network



(a)



(b)

Figure 2: a) Comparison of accuracies obtained by training the 5-layers model (34) by backpropagation (blue) and simulated annealing (red). b) Comparison of accuracies obtained by training the 2-layers network (35) by backpropagation (blue), simulated annealing (red) and quantum annealing (green).

is

$$\boxed{\begin{array}{c}\text{Input}\\(1,32,32)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{SpectralPool}\\(1,8,8)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Conv2d}(8\times 8)\\(10,1,1)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Flatten}\\(10)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Output}\\(10)\end{array}} . \tag{36}$$

Again, we use a 8bit representation for the synaptic weights which translates to a system of 10 spin glasses each containing 512 spins and the network is trained on a subset of the MNIST dataset consisting on 1000 samples.

The three layer network is defined as

$$\begin{aligned}&\boxed{\begin{array}{c}\text{Input}\\(1,32,32)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Conv2d}(5\times 5)\\(10,28,28)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{SpectralPool}\\(10,14,14)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Conv2d}(5\times 5)\\(30,10,10)\end{array}}\\ &\rightarrow \boxed{\begin{array}{c}\text{SpectralPool}\\(30,5,5)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Conv2d}(5\times 5)\\(10,1,1)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Flatten}\\(10)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Output}\\(10)\end{array}} .\end{aligned} \tag{37}$$

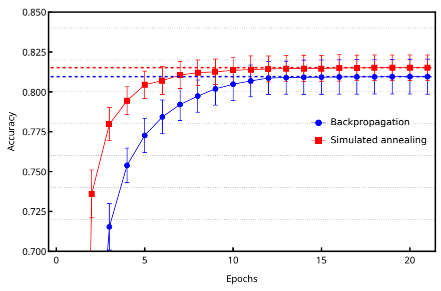
Note that in this case, we apply an amplification of the signal in the inverse network before each convolutional layer. This improves the performance of the simulated annealing procedure.

In Fig. 3, we compare the accuracies obtained by training the two networks by backpropagation and simulated annealing. As for the linear layer cases, the simulated annealing approach seems to perform well and give results comparable to the ones obtained by standard backpropagation.

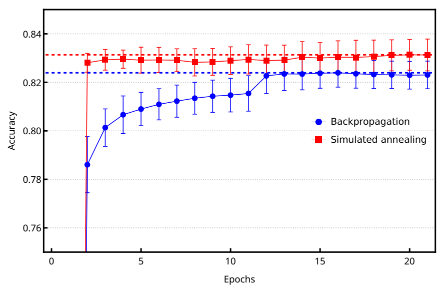
As before, in order to test the quantum annealing approach, we use smaller networks. For the single convolutional layer case we consider the network:

$$\boxed{\begin{array}{c}\text{Input}\\(1,32,32)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{SpectralPool}\\(1,4,4)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Conv2d}(4\times 4)\\(10,1,1)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Flatten}\\(10)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Output}\\(10)\end{array}} , \tag{38}$$

so that, by using a 4bit representation for the convolutional kernel, we end up with a system of 10 spin glasses each containing 64 spin variables.

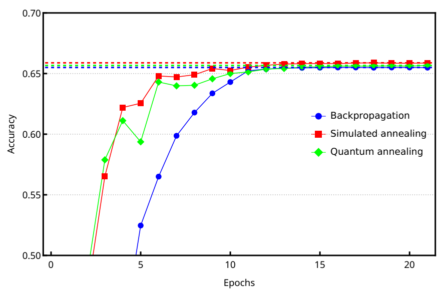


(a)

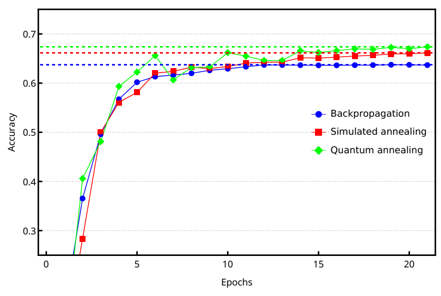


(b)

Figure 3: a) Comparison of accuracies obtained by training the 1 convolutional layer model (36) by backpropagation (blue) and by simulated annealing (red). b) Comparison of accuracies obtained by training the 3 convolutional layers model (37) by backpropagation (blue) and simulated annealing (red).



(a)



(b)

Figure 4: Comparison of accuracies obtained training by backpropagation (blue), simulated annealing (red) and quantum annealing (green) the 1 convolutional layer model (38) (a) and the 2 convolutional layers model (39) (b).

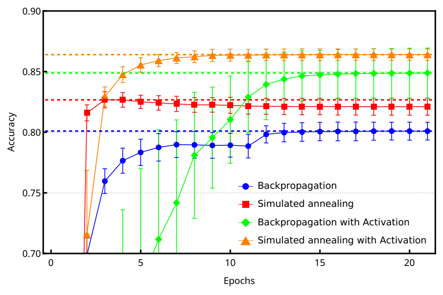
For the multi layers case, we consider the 2 layers architecture:

$$\begin{array}{c} \boxed{\begin{array}{c}\text{Input}\\(1,32,32)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{SpectralPool}\\(1,4,4)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Conv2d}(3\times 3)\\(4,2,2)\end{array}} \\ \rightarrow \boxed{\begin{array}{c}\text{Conv2d}(2\times 2)\\(10,1,1)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Flatten}\\(10)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Output}\\(10)\end{array}}, \end{array} \tag{39}$$

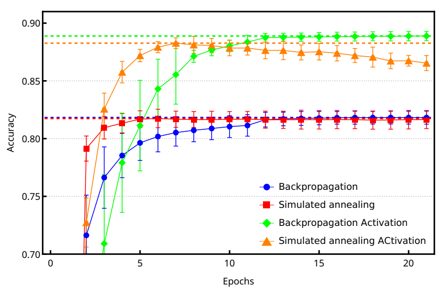
where the first layer is mapped to a system of 4 spin glasses of 32 spin variables and the second layer is mapped to a system of 10 spin glasses of 64 spin variables. Results are shown in Fig. 4.

5.4 Activation function

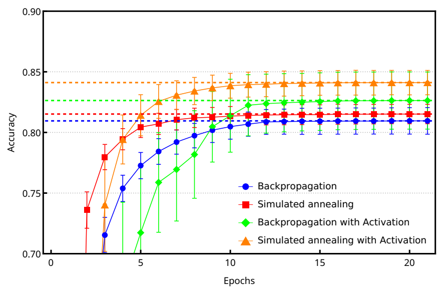
As discussed in Sec. 3, an activation function like LeakyReLU can be easily incorporated in the spin glass model by including the activation factors γ_i^a in the Hamiltonian parameters J_{jl}^i and h_j^i, Eq. (21).



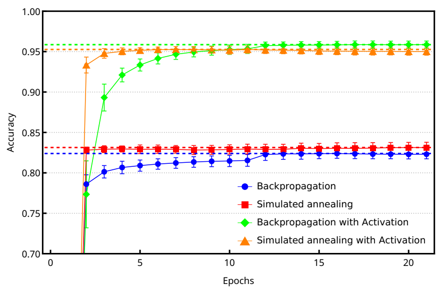
(a)



(b)



(c)



(d)

Figure 5: Comparison of the effect of introducing a LeakyReLu ($\alpha = 0.001$) activation function (after each linear and convolutional layer) when training by backpropagation and simulated annealing the 1 linear layer network (32) (a), the 3 linear layers network (34) (b), the 1 convolutional layer network (36) (c) and the 3 convolutional layer network (37).

In Fig. 5 we shows the effect of introducing a LeakyReLu activation function with negative slope $\alpha = 0.001$ after each layer for some of the network tested so far.

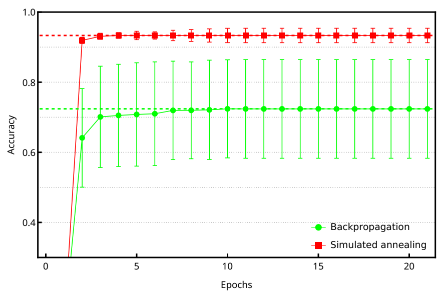
5.5 CNN

After having verified that the annealing training procedure works in a number of scenarios, let us test it on a full convolutional network architecture. Specifically, we consider the following Lenet-like architecture

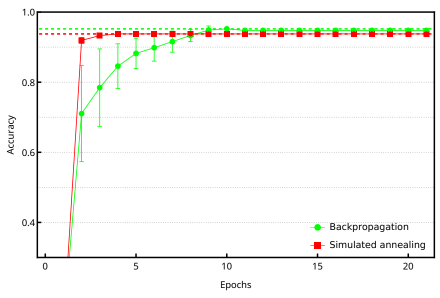
$$\boxed{\begin{matrix}\text{Input}\\(1,32,32)\end{matrix}} \to \boxed{\begin{matrix}\text{Conv2d}(5\times 5)\\(6,28,28)\end{matrix}} \xrightarrow{ACT} \boxed{\begin{matrix}\text{SpectralPool}\\(6,14,14)\end{matrix}} \to \boxed{\begin{matrix}\text{Conv2d}(5\times 5)\\(16,10,10)\end{matrix}}$$
$$\xrightarrow{ACT} \boxed{\begin{matrix}\text{SpectralPool}\\(16,5,5)\end{matrix}} \to \boxed{\begin{matrix}\text{Conv2d}(5\times 5)\\(120,1,1)\end{matrix}} \xrightarrow{ACT} \boxed{\begin{matrix}\text{Flatten}\\(120)\end{matrix}} \tag{40}$$
$$\to \boxed{\begin{matrix}\text{Linear}\\(120\times 84)\end{matrix}} \xrightarrow{ACT} \boxed{\begin{matrix}\text{Linear}\\(84\times 10)\end{matrix}} \xrightarrow{ACT} \boxed{\begin{matrix}\text{Output}\\(10)\end{matrix}},$$

where ACT indicates a LeakyReLU activation function.

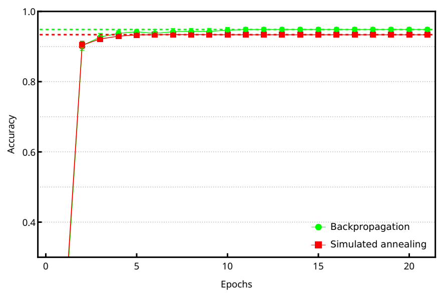
In Fig. 6, we compare the performance of the simulated annealing approach



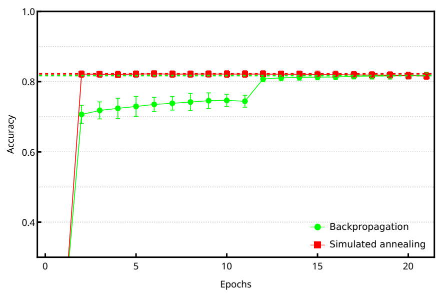
(a)



(b)



(c)



(d)

Figure 6: Comparison of accuracies obtained by training by backpropagation and simulated annealing the CNN (40) on the MNIST dataset with 1000 samples for different values of the LeakyRelu negative slope α. (a) $\alpha = 0$ (Relu), (b) $\alpha = 0.0001$, (c) $\alpha = 0.1$, (d) $\alpha = 1$.

with standard backpropagation when training the network over 1000 samples of the MNIST dataset for different values of the LeakyRelu negative slope α.

It appears that the annealing approach can provide good results also for larger architectures.

In Fig. 7 we compare the accuracy of simulated quantum annealing with that of simulated annealing for a scaled down version of the network (40).

5.6 Gate-based quantum computing

The approach outlined above has been tested on quantum annealing machines. However a number of algorithms (QAOA, Groover adaptive search (GAS),...) exist that allow to perform the training on gate based quantum computers.

Here, we consider the Imaginary Time Evolution Mimicking Circuit (ITEMC) approach. Compared to the alternatives, this approach requires less resources (no extra qubits like in GAS and, at least in first approximation, no extra measurements needed to estimate parameters like in QAOA). Experiments show that this result in faster execution times and more accurate results.

The idea behind the algorithm is to use rotation gates to approximate the imaginary time evolution of the spin glasses Hamiltonians (9) so that the spin configuration is evolved into its ground state.

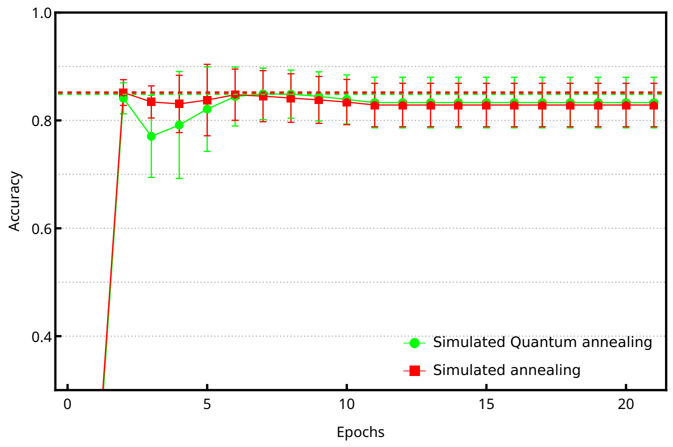


Figure 7: Comparison of accuracies obtained training by simulated quantum annealing (green) and simulated annealing (red) a scaled down version of (40) using a 4-bit representation for the network's weights.

The imaginary time evolution of the Hamiltonian (9) is

$$|\psi(\tau)\rangle = e^{-\tau\hat{H}}|\psi(0)\rangle = \prod_{\mu\nu} e^{-\tau J_{\mu\nu}\hat{Z}_\mu\hat{Z}_\nu} \prod_{\mu} e^{-\tau h_\mu \hat{Z}_\mu}|\psi(0)\rangle \,, \tag{41}$$

that can be approximated by the unitary operator

$$|\psi(\tau)\rangle \approx \prod_{\mu\nu} U_{\mu\nu}(\overline{\theta}_{\mu\nu}) \prod_{\mu} R_y(\overline{\theta}_\mu)|\psi(0)\rangle \,, \tag{42}$$

where

$$R_y(\overline{\theta}_\mu) = e^{-\frac{i}{2}\overline{\theta}_\mu \hat{Y}_\mu} \,, \qquad U_{\mu\nu}(\overline{\theta}_{\mu\nu}) = e^{-\frac{i}{2}(\overline{\theta}^1_{\mu\nu}\hat{Z}_\mu\hat{Y}_\nu + \overline{\theta}^0_{\mu\nu}\hat{Y}_\mu\hat{Z}_\nu)} \,. \tag{43}$$

When the initial state is in a uniform superposition

$$|\psi(0)\rangle = \bigotimes_\mu |+\rangle_\mu \,, \qquad |+\rangle_\mu = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right)_\mu \,, \tag{44}$$

the optimal rotation parameters are given by

$$\overline{\theta}_\mu = 2\arctan(\tanh(\tau h_\mu)) \,, \qquad (\overline{\theta}^1_{\mu\nu}, \overline{\theta}^0_{\mu\nu}) = \underset{(\theta^1_{\mu\nu},\theta^0_{\mu\nu})}{\operatorname{argmax}} f(\theta^1_{\mu\nu}, \theta^0_{\mu\nu}) \,, \tag{45}$$

and, in first approximation, f is

$$\begin{aligned} f(\theta^1_{\mu\nu}, \theta^0_{\mu\nu}) = \cos\left(\frac{\theta^0_{\mu\nu}}{2}\right)\left[a\cos\left(\frac{\theta^1_{\mu\nu}}{2}\right) + c\sin\left(\frac{\theta^1_{\mu\nu}}{2}\right)\right] + \\ + \sin\left(\frac{\theta^0_{\mu\nu}}{2}\right)\left[b\cos\left(\frac{\theta^1_{\mu\nu}}{2}\right) - d\sin\left(\frac{\theta^1_{\mu\nu}}{2}\right)\right] \,, \end{aligned} \tag{46}$$

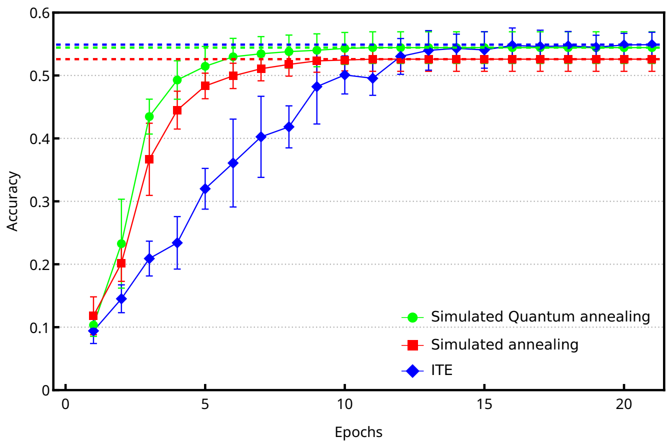


Figure 8: Comparison of accuracies obtained training by simulated quantum annealing (green), simulated annealing (red) and imaginary time evolution (blue) a single linear layer using a 2-bit representation for the network's weights.

where

$$\begin{aligned}
a &= \cosh(\tau J_{\mu\nu}) - \sinh(\tau J_{\mu\nu})\sin(\overline{\theta}_\mu)\sin(\overline{\theta}_\nu)\,, \\
b &= \sinh(\tau J_{\mu\nu})\cos(\overline{\theta}_\mu)\,, \\
c &= \sinh(\tau J_{\mu\nu})\cos(\overline{\theta}_\nu)\,, \\
d &= \cosh(\tau J_{\mu\nu})\cos(\overline{\theta}_\mu)\cos(\overline{\theta}_\nu)\,.
\end{aligned} \tag{47}$$

We tested the ITEMC algorithm (on simulated backend) to train a small linear layer for classification on the MNIST dataset with 1000 samples using a 2-bit representation for the weight variables. In Fig. 8 the training accuracy from the ITEMC algorithm is compared with those from simulated annealing and simulated quantum annealing showing that the approach performs equally well.