Learning Bit by Bit: Extracting the Essence of Machine Learning

Sascha Mücke, Nico Piatkowski, and Katharina Morik

TU Dortmund, AI Group, Dortmund, Germany
http://www-ai.cs.tu-dortmund.de

Abstract. Data mining and Machine Learning research has led to a wide variety of training methods and algorithms for different types of models. Many of these methods solve or approximate NP-hard optimization problems at their core, using vastly different approaches, some algebraic, others heuristic. This paper demonstrates another way of solving these problems by reducing them to quadratic polynomial optimization problems on binary variables. This class of parametric optimization problems is well-researched and powerful, and offers a unifying framework for many relevant ML problems that can all be tackled with one efficient solver. Because of the friendly domain of binary values, such a solver lends itself particularly well to hardware acceleration, as we further demonstrate in this paper by evaluating our problem reductions using FPGAs.

Keywords: · Machine Learning · Optimization · Hardware Acceleration

1 Introduction

Hardware acceleration for machine learning usually involves GPU implementations that can do fast linear algebra to enhance the speed of numerical computations. Our approach is different to GPU programming in that we make use of a fixed class $\mathcal{C}$ of parametric optimization problems that we solve directly on efficient specialized hardware. Solving a problem instance in $\mathcal{C}$ thus amounts to finding the correct parameters β and feeding them to the solver.

The underlying idea of using a non-universal compute-architecture for machine learning is indeed not new: State-of-the-art *quantum annealers* rely on the very same principle of parametric problem classes. There, optimization problems are encoded as potential energy between *qubits*; the global minimum of a loss function can be interpreted as the quantum state of lowest energy [4]. The fundamentally non-deterministic nature of quantum methods makes the daunting task of traversing an exponentially large solution space feasible. However, their practical implementation is a persisting challenge, and the development of actual quantum hardware is still in its infancy. The latest flagship, the*D-Wave 2000Q*,

can handle problems with 64[1] fully connected bits, which is by far not sufficient for realistic problem sizes.

Nevertheless, the particular class of optimization problems that quantum annealers can solve is well understood which motivates its use for hardware accelerators outside of the quantum world.

2 Boolean Optimization

A *pseudo-Boolean function* (PBF) is any function $f : \mathbb{B}^n \mapsto \mathbb{R}$ that assigns a real value to a fixed-length binary vector. Every PBF on n binary variables can be uniquely expressed as a polynomial of some degree $d \leq n$ with real-valued coefficients [2]. *Quadratic Unconstrained Binary Optimization* (QUBO) is the problem of finding an assignment $\boldsymbol{x}^*$ of n binary variables that is minimal with respect to a Boolean polynomial of degree 2:

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}\in\mathbb{B}^n} \sum_{i=1}^{n} \beta_i \boldsymbol{x}_i + \sum_{i=1}^{n}\sum_{j=1}^{i} \beta'_{ij} \boldsymbol{x}_i \boldsymbol{x}_j, \tag{1}$$

where β_i and β'_{ij} are real-valued parameters that are fixed for a given problem instance. As $x \cdot x = x$ for all $x \in \mathbb{B}$, the linear coefficients β_i can be integrated into the quadratic parameters β'_{ii} by setting $\beta_{ii} = \beta'_{ii} + \beta_i$, which makes for a simpler formula and leaves us with a perfect lower triangle matrix for β:

$$\sum_{i=1}^{n}\sum_{j=1}^{i} \beta_{ij} \boldsymbol{x}_i \boldsymbol{x}_j \tag{2}$$

It has been shown that all higher-degree pseudo-Boolean optimization problems can be reduced to quadratic problems [2]. For this reason a variety of well-known optimization problems like (Max-)3SAT and prime factorization, but also ML-related problems like clustering, maximum-a-posterior (MAP) estimation in Markov Random Fields, and binary constrained SVM learning can be reduced to QUBO or its *Ising* variety (where $\boldsymbol{x} \in \{-1, +1\}^n$).

3 Evolutionary QUBO Solver

If no specialized algorithm is known for a particular hard combinatorial optimization problem, randomized search heuristics, like simulated annealing or *evolutionary algorithms* (EA), provide a generic way to generate good solutions.

Inspired by biological evolution, EAs employ recombination or mutation on a set of "parent" solutions to produce a set of "offspring" solutions. A loss function, also called *fitness function* in the EA-context, is used to select those solutions which will constitute the next parent generation. This process is repeated until convergence or a pre-specified time-budget is exhausted [8].

[1] `https://www.dwavesys.com/sites/default/files/mwj_dwave_qubits2018.pdf`

Motivated by the inherently parallel nature of digital circuits, we developed a highly customizable $(\mu + \lambda)$-EA FPGA hardware architecture implemented in VHDL. Here, customizable implies that different types and sizes of FPGA hardware can be used.

Moreover, our hardware synthesizer allows the end-user to customize the maximal problem dimension n, the number of parent solutions μ, the number of offspring solutions λ, and the number of bits per coefficient β_{ij}. In case of low-budget FPGA, this allows us to either allocate more FPGA resources for parallel computation (μ and λ) or for the problem size (n and β).

We used this hardware optimizer for evaluating the QUBO and Ising model embeddings we are going to present in the following sections.

4 Exemplary Learning Tasks

With an efficient solver for QUBO and the Ising model at hand, solving a specific optimization problem reduces to finding an efficient embedding into the domain of n-dimensional binary vectors and devising a method for calculating β such that the minimizing vector x^* corresponds to an optimal solution of the original problem. The actual optimization step then amounts to uploading β to the hardware solver which approximates a global optimum that yields an approximately optimal solution to the original problem, once decoded back into the original domain. As we will show, it is possible for multiple valid embeddings to exist for the same problem class, but their effectiveness differs from instance to instance.

4.1 Clustering

A prototypical data mining problem is k-means clustering which is already NP-hard for $k = 2$. To derive the coefficients β for an Ising model solving the 2-means clustering problem, we can use the method devised in [1], where each binary variable $\sigma_i \in \{+1, -1\}$ indicates whether a corresponding data point $\boldsymbol{x}^i \in \mathcal{D}$ belongs to cluster $+1$ or cluster -1 – the problem dimension is thus $n = |\mathcal{D}|$.

First, we assume that $\mathcal{D}$ has zero mean, such that $\sum_i \boldsymbol{x}^i = 0$. Next, we compute the Gramian matrix $\boldsymbol{G}$, where every entry corresponds to an inner product of two data points:

$$G_{ij} = \langle \boldsymbol{x}^i, \boldsymbol{x}^j \rangle$$

As shown in [1], minimizing an Ising model using $\boldsymbol{G}$ as coupling matrix corresponds to maximizing the *between cluster scatter* under the assumption that the clusters are approximately of equal size. Optionally a kernel function can be used instead of the inner product, and the resulting kernel matrix can be centered to maintain the zero mean property in the resulting feature space.

For simplicity we will stick to the linear kernel and assume that $\boldsymbol{G}$ is already centered. As our hardware optimizer encodes β as signed b-bit integers (with $b \leq 32$), we have to round the entries of $\boldsymbol{G}$, which are real-valued. To minimize

loss of precision, we scale the parameters to use the full range of b bits before rounding. As the overall objective function is a linear combination of β, scaling all coefficients by α is the same as multiplying the objective function by α, which means that the position of the optimum is unaffected. The final formula for a single coefficient comes out to

$$\beta_{ij} = \lfloor \alpha G_{ij} + 0.5 \rfloor \text{ with } \alpha = \frac{2^{b-1} - 1}{\max_{i,j} |G_{ij}|}.$$

Notice that there is no practical difference in precision between integer and fixed-point representation, as the latter is nothing more than a scaled integer representation to begin with.

Exemplary optimization results using this method on the UCI datasets *Iris* and *Sonar* are shown in Fig. 1.

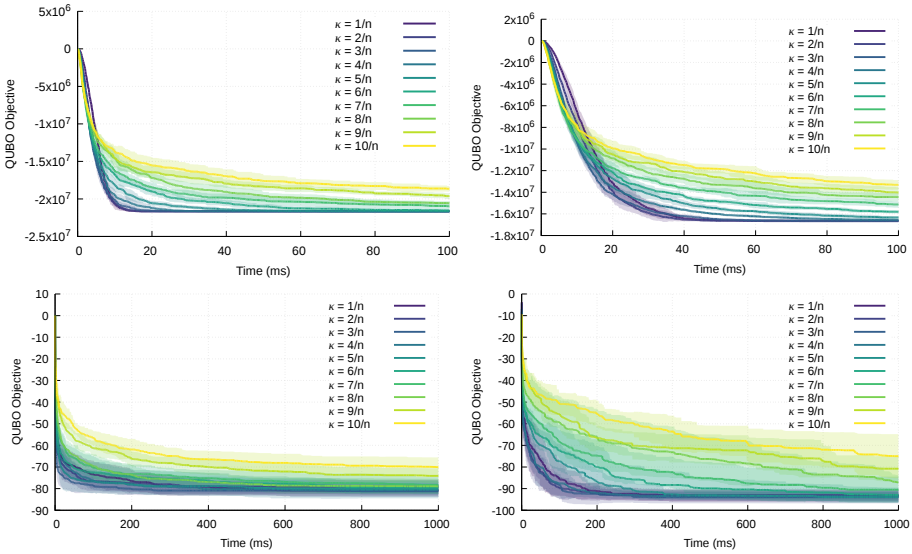


Fig. 1. QUBO loss value over time with different mutation rates, each averaged over 10 runs. Uncertainty indicated by transparent areas. Top-left: 2-means on *Iris* ($n = 150$, $d = 4$). Top-right: 2-means on *Sonar* ($n = 208$, $d = 61$). Bottom-left: MRF-MAP with edge encoding. Bottom-right: MRF-MAP with vertex encoding.

4.2 Support Vector Machine

A linear Support Vector Machine (SVM) is a classifier that – in the simplest case – takes a labeled dataset $\mathcal{D} = \{(\boldsymbol{x}^i, y_i) \mid 1 \leq i \leq n\} \subseteq \mathcal{X} \times \mathcal{Y}$ of two classes ($\mathcal{Y} = \{+1, -1\}$) and tries to separate them with a hyperplane [3]. As there may be infinitely many such hyperplanes, an additional objective is to maximize the

margin, which is the area around the hyperplane containing no data points, in an attempt to obtain best generalization. The hyperplane is represented as a normal vector $\boldsymbol{w}$ and an offset b. To ensure correctness the optimization is subject to the requirement that every data point be classified correctly, i.e. every data point lies on the correct side of the plane:

$$(\langle \boldsymbol{w}, \boldsymbol{x}^i \rangle - b) \cdot y_i \geq 1$$

As for real-world data perfect linear separability is unlikely, each data point is assigned a *slack variable* ξ_i such that $\xi_i > 0$ indicates that $\boldsymbol{x}^i$ violates the correctness condition. This yields the second objective, which is to minimize the total slack, so that the entire minimization problem comes out to be

$$\begin{aligned} &\text{minimize } \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_i \xi_i \\ &\text{s.t. } \forall i.\ (\langle \boldsymbol{w}, \boldsymbol{x}^i \rangle - b) \cdot y_i \geq 1 - \xi_i \end{aligned}$$

The optimization is done by adjusting $\boldsymbol{w}$, b and ξ_i, while C is a free parameter controlling the impact of wrong classification on the total loss value. In fact, C is nothing more than an inverse regularization factor, as the loss function can be rewritten as $\sum_i \xi_i + \lambda\|\boldsymbol{w}\|_2^2$, which illustrates that SVM learning is merely a regularized hinge loss minimization. Typically this problem is solved using its Lagrange dual, which is

$$\begin{aligned} &\text{maximize } \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}^i, \boldsymbol{x}^j \rangle \\ &\text{s.t. } \forall i.\ 0 \leq \alpha_i \leq C \text{ and } \sum_i \alpha_i y_i = 0. \end{aligned} \tag{3}$$

Note that eq. (3) has striking similarity to eq. (1), the QUBO objective function, only a) it is a maximization instead of a minimization problem, b) index j goes from 1 to n instead to i, which makes for a full matrix instead of a lower triangle matrix, and c) α_i is continuous between 0 and C instead of a boolean. Points a) and b) are easy to deal with by flipping the sign and multiplying all entries except for the main diagonal of the triangle matrix by 2. Point c) requires a radical simplification: Instead of a continuous $\alpha_i \in [0, C]$ we assume that α_i is either 0 or C, reducing it to a boolean indicator whether $\boldsymbol{x}^i$ is a support vector or not. Thus we can substitute α_i for $C \cdot \boldsymbol{z}_i$ with $\boldsymbol{z}_i \in \mathbb{B}$ and end up with the following QUBO formulation:

$$\sum_{i=1}^{n} -C\boldsymbol{z}_i + \sum_{i=1}^{n}\left(\sum_{j=1}^{i-1} C^2 t_{ij}\boldsymbol{z}_i\boldsymbol{z}_j + \frac{1}{2}C^2 t_{ii}\boldsymbol{z}_i\right)$$

$$\text{where } t_{ij} = y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$$

In terms of eq. (2) we can express β as

$$\beta_{ij} = \begin{cases} \frac{1}{2}C^2 t_{ii} - C & \text{if } i = j \\ C^2 t_{ij} & \text{otherwise} \end{cases}$$

4.3 Markov Random Field

Another typical NP-hard ML problem is to determine the most likely configuration of variables in a Markov Random Field, known as the MAP prediction problem [9]. Given the weight parameters $\theta_{uv=xy} \in \mathbb{Z}$ of an MRF with graphical structure $G = (V, E)$ and variables $(X_v)_{v \in V}$ from finite state spaces $(\mathcal{X}_v)_{v \in V}$, we demonstrate two different QUBO encodings to maximize the joint probability over all possible variable assignments. These two encodings ultimately have the same objective, but lead to significantly different runtimes on our optimizer hardware, depending on the given MRF's structure. We assume θ to be integer-valued to fit our optimizer's architecture, and also because integer MRF's have been shown to be generally well-performing alternatives to MRFs with real-valued weights, especially when resources are constrained [7, 6].

For the first QUBO embedding of the MAP problem we encode the assignments of all X_v as a concatenation of one-hot vectors

$$(X_1 = x_1^{i_1}, \ldots, X_m = x_m^{i_m}) \;\mapsto\; \underbrace{0 \ldots 010 \ldots 0}_{|\mathcal{X}_1|} \ldots \underbrace{0 \ldots 010 \ldots 0}_{|\mathcal{X}_m|},$$

where $m = |V|$ and x_k^i is the i-th value in $\mathcal{X}_k$. The QUBO problem dimension is equal to the total number of variable states, $n = \sum_{v \in V} |\mathcal{X}_v|$. The weights are encoded into the quadratic coefficients: Let $\iota : V \times \mathcal{X} \mapsto \mathbb{N}$ be a helper function that, given a vertex and variable state, returns the corresponding bit index within the concatenated one-hot vector. Given two indices $i = \iota(u, x)$ and $j = \iota(v, y)$ with $u \neq v$, the corresponding coefficient is $\beta_{ij} = -\theta_{uv=xy}$; the negative sign turns MAP into a minimization problem. If $u = v$ and $i \neq j$, the indices belong to the one-hot encoding of the same variable; as a variable can only be in one state at a time, these two bits cannot both be 1, as this would lead to an invalid one-hot encoding. Therefore the corresponding coefficient must be set to an arbitrary positive penalty value $P \gg \theta$, large enough to ensure that even the worst valid encoding has a lower loss value than P. On the other hand, if $i = j$ then $\beta_{ij} = 0$, because the variable assignments have no weights on their own.

$$\beta_{ij} = \begin{cases} 0 & \text{if } i = j \\ P & \text{if } i \neq j \text{ and } u = v \\ -\theta_{uv=xy} & \text{otherwise} \end{cases}$$
$$\text{where } i = \iota(u, x),$$
$$j = \iota(v, y)$$

The dimension can be reduced by removing bits with index i where all β_{ij} are either 0 or P for all $j \leq i$, as those can never be part of an optimal solution.

For a different QUBO embedding, we may assign bits $\boldsymbol{z}_{uv=xy}$ to all non-zero weights $\theta_{uv=xy}$ between specific values x and y of two variables X_u and X_v, indicating that $X_u = x$ and $X_v = y$ (see fig. 2). Again, to avoid multiple or

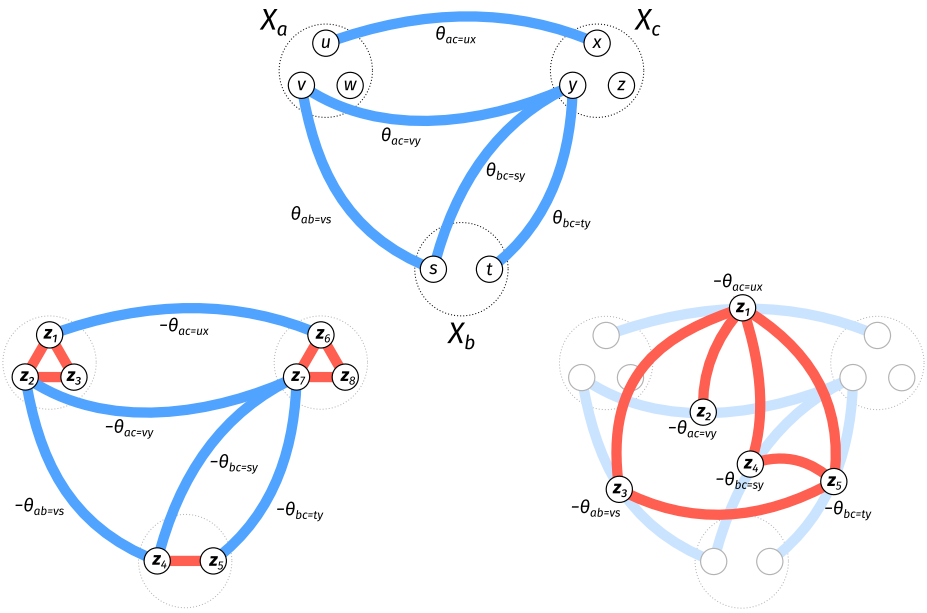


Fig. 2. Schemes for embedding an MRF into QUBO: Variables $(X_v)_{v \in V}$ are represented as dotted circles containing their possible states, weights θ as blue edges between individual variable states (top). For the first embedding (bottom left), variable states become binary variables $\boldsymbol{z}$ with their respective quadratic coefficients $\beta_{ij} = -\theta_{uv=xy}$ if $i = \iota(u, x)$ and $j = \iota(v, y)$. For the second embedding (bottom right), all non-zero parameters θ become binary variables with $-\theta$ as their respective linear coefficient. For both embeddings, invalid variable combinations receive a penalty weight P, represented as red edges.

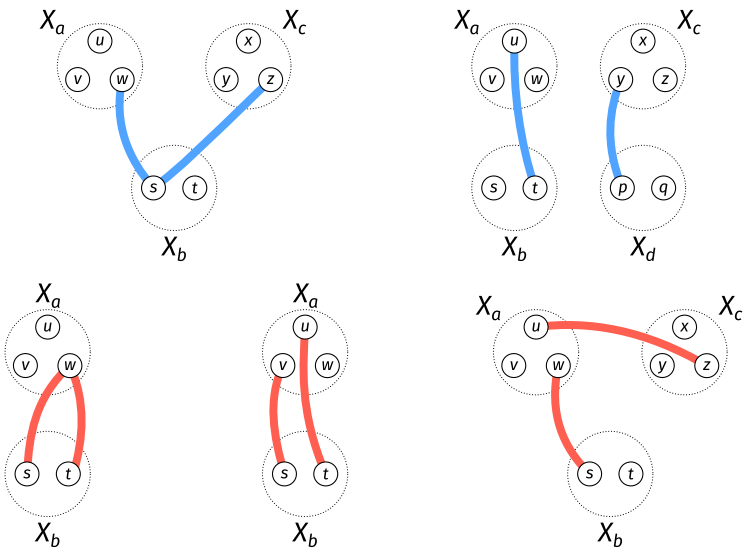


Fig. 3. Possible edge configurations. Top: Valid configurations, where two edges are either disjoint or, if they share an edge, agree on the variable state. Bottom: Invalid configurations, where in case both edges were "active" at least one variable would simultaneously be in two states. These pairs of edges require a penalty weight between their corresponding binary variables within the second embedding.

conflicting variable assignments we introduce penalty weights between pairs of edges that share a variable but disagree in its specific value (see fig. 3).

The result $\boldsymbol{z}^*$ of the optimization is a vector of indicators for "active" edges indexed by the set of all non-zero weights θ, from which we can derive an assignment for every variable, provided it has at least one incident edge of non-zero weight:

$$X_v = \bigcup \{\tilde{y} \mid u \in V \backslash \{v\},\ u \in \mathcal{X}_u,\ \tilde{y} \in \mathcal{X}_v,\ \boldsymbol{z}^*_{uv=x\tilde{y}} = 1\}$$

For a valid encoding, the above set is always a singleton, such that its union yields the contained element.

5 Evaluation

We evaluated our embeddings using our optimizer hardware on multiple UCI datasets.

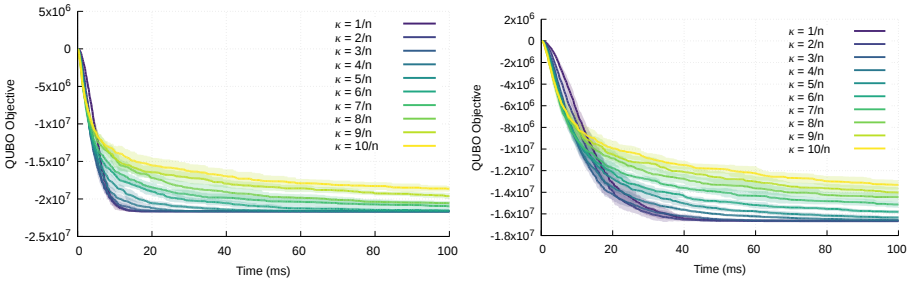
For 2-means clustering we took five datasets and compared the resulting k-means loss values of the Ising model embedding to Lloyd's algorithm [5] as implemented by R's `kmeans` method[2]. For calculating β we chose the simplest case of a linear kernel. The results are listed in table 1; the resulting clusterings

[2] `https://stat.ethz.ch/R-manual/R-patched/library/stats/html/kmeans.html`

are similar to those obtained through Lloyd's algorithm, though our loss values were mostly slightly higher. Our values seem to be better the higher the data dimension d is, surpassing Lloyd's algorithm on *Sonar* containing 60 numerical variables per data point. We assume that the generally slightly higher loss values are due to the simplifying assumption that the clusters are about equal in size, not due to insufficient convergence of our optimizer, as we found that we reached identical optima in every run, leading to a standard deviation of 0 for every loss value. Convergence plots of our hardware optimizer for two exemplary datasets are shown in fig. 4.

Table 1. 2-means loss values (sums of cluster variances) on different datasets; each optimization run on the Ising model was repeated 10 times, the EA parameters were $\mu = 2$ and $\lambda = 30$. The columns "variables" and "data points" contain the data dimension d and number of points n used for each experiment.

dataset	variables	data points	Lloyd	Ising model
Iris	4	150	152.3480	163.1659 ± 0.0
Abalone	8	500	1947.141	2121.1438 ± 0.0
Sonar	60	208	280.5821	280.5696 ± 0.0
Wilt	5	500	14663360	14734828 ± 0.0
Ionosphere	32	351	2387.2917	2388.3108 ± 0.0



Iris ($d = 4$, $n = 150$) *Sonar* ($d = 60$, $n = 208$)

Fig. 4. Convergence plots for the 2-means Ising model optimization performed on our hardware optimizer using different mutation rates κ, averaged over 10 runs.

The SVM embedding was tested extensively on the *Sonar* dataset, where we created ten random splits of 2/3 training data and 1/3 test data. Using our hardware optimizer, we then trained the binary SVM on each test set of each split with $C = 10^k$ for $k \in \{-3, -2, -1, 0, 1, 2\}$ and calculated the accuracy on the respective test set; see table 2 for the full results. For $C = 10^{-3}$, our

optimizer found the optimum $\boldsymbol{z}^* = \mathbf{1}$, which means that every data point was a support vector. For bigger C the number of support vectors decreased, until for $C = 10^2$ the optimum was $\boldsymbol{z}^* = \mathbf{0}$, which is why 10^2 is missing from the table, as $\boldsymbol{w}$ and b could not be calculated without at least one support vector. The best accuracy was achieved with $C = 1$ and came out to be about 75%. On the same splits we trained a conventional SVM (*libsvm* using the `svm` method from R's `e1071` package[3]) and found that the test accuracies were in fact very similar, the difference being less than one percent on average (see table 3), which is a promising result considering the radical simplification steps taken for this model.

Table 2. Test accuracies for varying C values on ten random splits (2/3 train, 1/3 test) of the *Sonar* dataset. (Values of the individual splits are $\times 10^{-4}$)

	split										
C	1	2	3	4	5	6	7	8	9	10	*mean* ± *sd*
0.001	6115	6619	6547	5899	6619	6619	6259	6331	6547	5899	0.6345 ± 0.0291
0.01	6547	6619	6619	5899	6835	6619	6763	6187	6547	6043	0.6468 ± 0.0313
0.1	7410	6403	6403	6165	7324	6835	7331	6547	6619	7770	0.6881 ± 0.0540
1	7712	7748	6691	7957	6986	7554	7540	7942	7432	7180	0.7474 ± 0.0413
10	6129	6209	6151	5676	5849	6129	6187	6777	6849	6511	0.6247 ± 0.0371

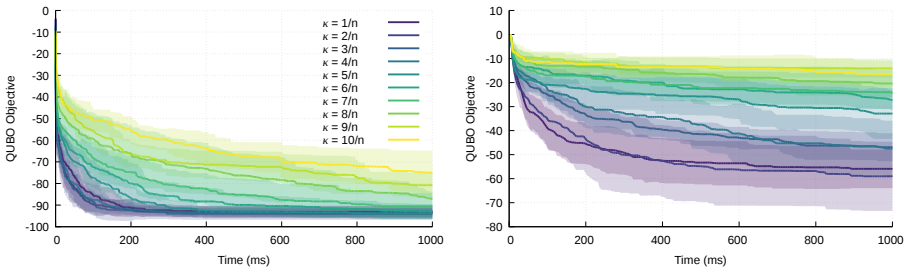
Table 3. Test accuracies for $C = 1$ on the same ten random splits achieved by *libsvm*. (Values of the individual splits are $\times 10^{-4}$)

split										
1	2	3	4	5	6	7	8	9	10	*mean* ± *sd*
7101	7246	7826	7826	7536	7681	7681	6667	8551	7536	0.7565 ± 0.0501

The two MRF embeddings were tested on the *Sonar* and *Mushroom* datasets, using weights θ from pre-trained integer MRFs with Chow-Liu tree structures. Let $|\theta|$ denote the number of non-zero weights of a trained MRF: While the MRF on *Sonar* had only few non-zero weights ($|\theta| = 102$), *Mushroom* yielded a much more densely connected MRF ($|\theta| = 679$). Convergence plots for *Mushroom* can be seen in fig. 5. Obviously the state embedding (where each bit encodes one variable state as a one-hot vector) is superior on *Mushroom*, as it converges much more quickly to a very good optimum, whereas the edge embedding (where each bit encodes one value from θ) takes much longer and does not converge to

[3] `https://www.rdocumentation.org/packages/e1071/versions/1.7-1/topics/svm`

nearly as good an optimum, even after several minutes. On the contrary, the edge embedding on *Sonar* is much more efficient than the state embedding (see fig. 6). The different convergence rates are due to the different QUBO dimensions of the embeddings: The state embedding on *Mushroom* has dimension $n_s = \sum_{v \in V} |\mathcal{X}_v| = 127$, which can be further reduced to 107 by removing unnecessary bits as described in section 4.3. The edge embedding on the other hand leads to a dimension $n_e = |\theta| = 679$. As our hardware optimizer needed about $\mathcal{O}(n^2)$ to perform the optimization, the latter encoding lead to a performance about 40 times slower. The *Sonar* MRF has $n_s = 662$, which the reduction step improves significantly to 117. As $|\theta| = 102$ (and $n_e = 102$, consequently) for the *Sonar* MRF, the edge embedding is even better, though.



reduced state embedding edge embedding

Fig. 5. Convergence plots for both QUBO embeddings of MRFs on the *Mushroom* dataset; the state embedding clearly converges much faster on this MRF.

As a general rule we observe that for relatively dense MRFs, the (reduced) state encoding leads to faster convergence, while for sparse MRFs with $|\theta| < \sum_{v \in V} |\mathcal{X}_v^\theta|$ (where $\mathcal{X}_v^\theta$ is the set of variable states that have at least one non-zero θ associated with them) the edge encoding is preferable.

6 Conclusion and Future Work

QUBO and Ising models constitute simple but versatile optimization problems that capture the essence of many problems highly relevant to Machine Learning, and in this work we have summarized some interesting embeddings of NP-hard Machine Learning problems into both models. A general hardware-based solver for either of these models is thus a powerful tool for Machine Learning, and the domain of bit vectors makes for an efficient problem representation even when resources are restricted, e.g. in embedded systems. Further we showed that there can exist multiple embeddings of the same problem that perform differently depending on properties of the problem instance.

It would be interesting to uncover more embeddings of ML problems in the future, both with conventional hardware acceleration like we did, but also with the advent of quantum annealing in mind.

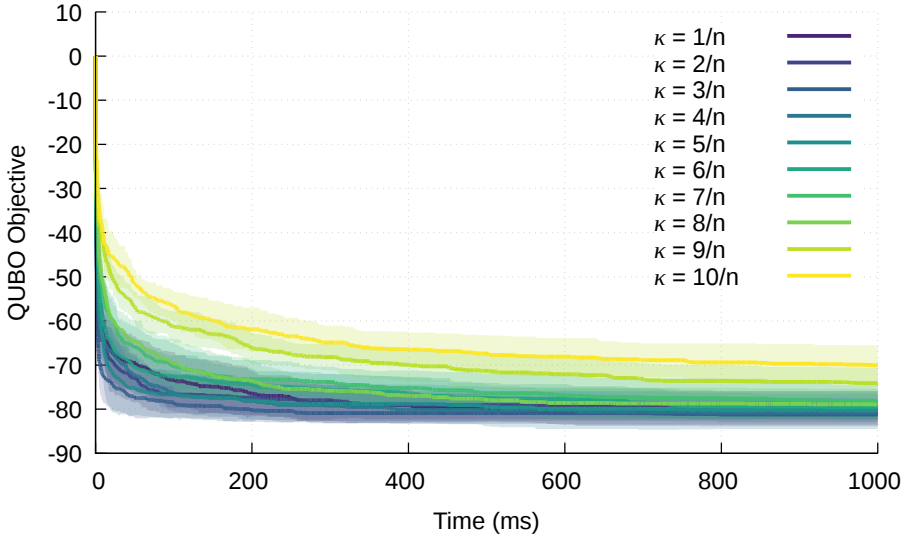


Fig. 6. Convergence plot for the edge embedding on *Sonar*; unlike on *Mushroom*, this embedding leads to quick convergence to a good optimum.

Acknowledgement This research has been funded by the Federal Ministry of Education and Research of Germany as part of the competence center for machine learning ML2R (01|S18038A)

References

1. Bauckhage, C., Ojeda, C., Sifa, R., Wrobel, S.: Adiabatic quantum computing for kernel k=2 means clustering. In: Proceedings of the LWDA 2018. pp. 21–32 (2018)
2. Boros, E., Hammer, P.L.: Pseudo-boolean optimization. Discrete applied mathematics **123**(1-3), 155–225 (2002)
3. Cortes, C., Vapnik, V.: Support vector machine. Machine learning **20**(3), 273–297 (1995)
4. Kadowaki, T., Nishimori, H.: Quantum annealing in the transverse ising model. Physical Review E **58**(5), 5355 (1998)
5. Lloyd, S.: Least squares quantization in pcm. IEEE transactions on information theory **28**(2), 129–137 (1982)
6. Piatkowski, N.: Exponential families on resource-constrained systems. Ph.D. thesis, Technical University of Dortmund, Germany (2018)
7. Piatkowski, N., Lee, S., Morik, K.: Integer undirected graphical models for resource-constrained systems. Neurocomputing **173**, 9–23 (2016)
8. Schwefel, H.P., Rudolph, G.: Contemporary evolution strategies. In: European conference on artificial life. pp. 891–907. Springer (1995)
9. Wainwright, M.J., Jordan, M.I., et al.: Graphical models, exponential families, and variational inference. F+Trends in Machine Learning **1**(1–2), 1–305 (2008)