QBoost: Large Scale Classifier Training with Adiabatic Quantum Optimization

Hartmut Neven NEVEN@GOOGLE.COM
Google Inc.

Vasil S. Denchev DENCHEV@GMAIL.COM
Purdue University

Geordie Rose ROSE@DWAVESYS.COM and **William G. Macready** WGM@DWAVESYS.COM
D-Wave Systems Inc.

Editor: Steven C.H. Hoi and Wray Buntine

Abstract

We introduce a novel discrete optimization method for training in the context of a boosting framework for large scale binary classifiers. The motivation is to cast the training problem into the format required by existing adiabatic quantum hardware. First we provide theoretical arguments concerning the transformation of an originally continuous optimization problem into one with discrete variables of low bit depth. Next we propose *QBoost* as an iterative training algorithm in which a subset of weak classifiers is selected by solving a hard optimization problem in each iteration. A strong classifier is incrementally constructed by concatenating the subsets of weak classifiers. We supplement the findings with experiments on one synthetic and two natural data sets and compare against the performance of existing boosting algorithms. Finally, by conducting a quantum Monte Carlo simulation we gather evidence that adiabatic quantum optimization is able to handle the discrete optimization problems generated by QBoost.

Keywords: adiabatic quantum computing, discrete optimization, machine learning, supervised learning, boosting

1. Introduction

We perform binary classifier training using discrete optimization in a formulation adapted to take advantage of emerging hardware that performs adiabatic quantum optimization (AQO). AQO, first introduced by Farhi et al. (2000, 2001), is a quantum computing model with good prospects for scalable and practically useful hardware implementations. Aharonov et al. (2004) showed it to be polynomially equivalent to the gate model of quantum computation. Theoretical and numeric studies of the purported computational superiority of AQO over classical computing have repeatedly given encouraging results, e.g. Santoro et al. (2002); Farhi et al. (2009); Amin and Choi (2009); Dickson and Amin (2011).

Significant investments are underway by the Canadian company *D-Wave Systems* to develop a hardware implementation. A series of rigorous studies of the quantum mechanical properties of the D-Wave processors, culminating in a recent Nature publication (Johnson et al., 2011), have increased the excitement in the quantum computing community for this approach. This was further fueled by news of Lockheed Martin purchasing a D-Wave machine. Most recently, benchmarking tests performed by D-Wave on their current chips (Rose, 2011) have indicated that they are indeed capable of providing dramatic speedups with respect to the best known classical algorithms for certain NP-hard problems.

For machine learning purposes, D-Wave's implementation of AQO can be regarded as a discrete optimization engine that accepts any problems formulated as quadratic unconstrained binary optimization (QUBO), also equivalent to the Ising model and Weighted MAX-2-SAT. It should be noted that the training formulation we propose here is a good format for AQO independently of D-Wave's efforts since it can be physically realized as the simplest possible multi-qubit configuration—an Ising system (Brush, 1967).

2. The Learning Task

We study binary classifiers of the form $y = \operatorname{sign}\left(\boldsymbol{w}^T \boldsymbol{x} + b\right)$, where $\boldsymbol{x} \in \mathbb{R}^N$ is a general input pattern[1] to be classified, $y \in \{-1, 1\}$ is the label associated with $\boldsymbol{x}$, $\boldsymbol{w} \in \mathbb{R}^N$ is a vector of weights to be optimized, and $b \in \mathbb{R}$ is the bias. Training, also known as regularized risk minimization, consists of choosing $\boldsymbol{w}$ and b by simultaneously minimizing two terms: *empirical risk* $R(\boldsymbol{w}, b) = \sum_{s=1}^{S} L\left(m\left(\boldsymbol{x}_s, y_s, \boldsymbol{w}, b\right)\right)/S$ and *regularization* $\Omega(\boldsymbol{w})$.

R, via a *loss function* L, estimates the error that any candidate classifier causes over a set of S training examples $\{(\boldsymbol{x}_s, y_s) | s = 1, \ldots, S\}$. The argument of L is known as the *margin* of example s with respect to the decision hyperplane defined by $\boldsymbol{w}$ and b: $m\left(\boldsymbol{x}_s, y_s, \boldsymbol{w}, b\right) = y_s\left(\boldsymbol{w}^T \boldsymbol{x}_s + b\right)$.

Ω controls the complexity of the classifier and is necessary for good generalization because classifiers with high complexity display overfitting—they can classify the training set with low error but may not do well on previously unseen data. Training amounts to solving

$$(\boldsymbol{w}, b)^* = \arg\min_{\boldsymbol{w}, b} \left\{R\left(\boldsymbol{w}, b\right) + \Omega\left(\boldsymbol{w}\right)\right\} \ . \tag{1}$$

The most natural choice for L is *0-1 loss*, which simply counts misclassifications:

$$L_{\text{0-1}}(m) = \left(1 - \operatorname{sign}\left(m\right)\right)/2 \tag{2}$$

The optimization problem (1) with $L_{\text{0-1}}$ is NP-hard due to non-convexity (Feldman et al., 2010). Also, because $L_{\text{0-1}}$ does not enforce a margin, the generalization of classifiers trained with it is bad even when regularization is applied (Vapnik, 1998). In order to avoid dealing with NP-hard optimization problems and to build large-margin classifiers, in practice $L_{\text{0-1}}$ is replaced by some convex upper bound (e.g. square, logistic, exponential, hinge). This allows arriving at convex optimization problems that can be rigorously analyzed and efficiently solved by classical means. An example of a convex upper bound to $L_{\text{0-1}}$ is *square loss*:

$$L_{\text{square}}(m) = (m - 1)^2 \tag{3}$$

The natural choice for Ω is ℓ_0-norm penalization of $\boldsymbol{w}$ because that explicitly drives weights towards exact zero. This is not only associated with good generalization but also fast execution during the performance phase. However, ℓ_0-norm regularization leads to non-convex optimization problems. Again, to avoid computational hardness, ℓ_0-norm is

1. Any feature vector consisting of raw data, extracted features, kernel or weak classifiers outputs, etc.

frequently replaced by convex alternatives such as the ℓ_2- or ℓ_1-norm. Regularization based on ℓ_2-norm has the nicest mathematical form in the sense that the resulting optimization problems are convex and differentiable. However, when continuous weight variables are used in training, ℓ_2-norm regularization only decreases the magnitude of weights but does not produce exact zeros. The reason for this effect is that in the optimization problem ℓ_2-norm regularization results in vanishing gradient magnitudes in the direction of zero weights. The alternative, ℓ_1-norm, under certain conditions may succeed at enforcing sparsity but leads to more complicated convex optimization problems due to non-differentiability.

3. Low-Precision Discrete Variables

The D-Wave quantum optimization processor that we aim to deploy for training requires problems to be discrete and formulated as QUBO. Further, the current hardware generation—Vesuvius (Rose, 2011)—can handle a maximum of 512 binary variables, which imposes the additional requirement of being frugal with the bit-depth of weight variables.

3.1. The Discrete Optimization Problem

We discretize the elements of $\boldsymbol{w}$ to some low bit-depth $d_w < 64$. While this approach is somewhat unconventional, in Subsection 3.2 we argue that the weights do not need high precision. In fact we show a favorable condition $d_w \geq \log(S/N)$ in the case of binary features such as the weak classifiers typically used in boosting. Even though analyzing classifiers constructed out of more general sets of features appears to be more difficult, experiments provide support for using low-precision weights. Finally, we also discretize the bias b with some low bit-depth $d_b < 64$

Given the options for loss function and regularization discussed in Section 2, here we study regularized risk minimization with L_{square} and ℓ_0-norm regularization over discrete variables $\dot{\boldsymbol{w}}$ and $\dot{b}$ of bit depth d_w and d_b respectively. This leads to a QUBO-compatible baseline optimization problem:

$$(\dot{\boldsymbol{w}}, \dot{b})^* = \arg\min_{\dot{\boldsymbol{w}}, \dot{b}} \left\{ \frac{1}{S} \sum_{s=1}^{S} L_{\text{square}} \left(y_s(\dot{\boldsymbol{w}}^T \boldsymbol{x}_s + \dot{b}) \right) + \lambda \|\dot{\boldsymbol{w}}\|_0 \right\}, \tag{4}$$

where $\lambda \in \mathbb{R}_{>0}$ controls the relative importance of regularization. The full QUBO derivation for a similar learning formulation is illustrated in Appendix A of Denchev et al. (2012).

3.2. Bit-Depth of Weight Variables

It is evident from (2) that $L_{0\text{-}1}$ enforces an inequality constraint per training example:

$$y_s \left(\boldsymbol{w}^T \boldsymbol{x}_s + b \right) \geq 0 \text{ for } s = 1, \ldots, S \tag{5}$$

Each training example brings about an inequality, which demands to choose weights that are on one side of a diagonal hyperplane in N-dimensional space. If we restrict each $\boldsymbol{x}_s \in \{-1, 1\}^N$, the hyperplane corresponding to example s is defined by a set of ± 1 coefficients depending on $\boldsymbol{x}_s$. Fig. 1 illustrates the situation for $N = 3$. The number of regions created

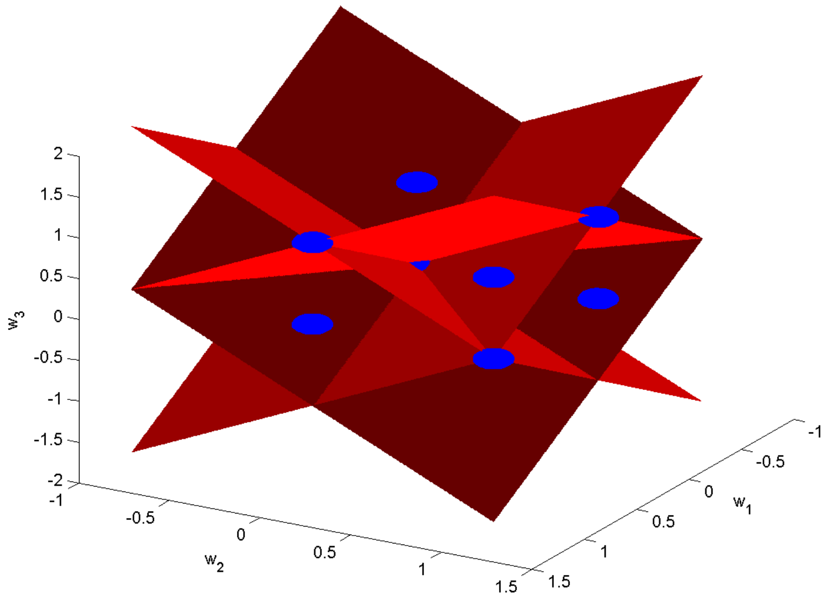

Figure 1: Arrangement of the diagonal hyperplanes that define the solution spaces for selecting $\boldsymbol{w}^*$. Depicted is the situation for $N = 3$, which yields 14 regions. The number of solution regions grows rapidly: $N = 4$ leads to 104 and $N = 5$ to 1882 regions. Here all possible hyperplanes are shown. However, in practice S training examples invoke only a small subset of the 2^{N-1} possible hyperplanes. The blue dots are the vertices of a cube placed in the positive quadrant with one vertex coinciding with the origin. They correspond to weight configurations that can be represented with one bit. Multi-bit weights give rise to a cube-shaped lattice.

by S hyperplanes is calculated using their characteristic polynomial (Orlik and Terao, 1992):

$$N_{\text{regions}} = (-1)^N \sum_{S_k} (-1)^k (-1)^{dim(\bigcap S_k)} \ , \tag{6}$$

where S_k designates the k-element subsets of the S hyperplanes and $dim(\bigcap S_k)$ is the dimension of the intersection of S_k.[2] Due to linear dependencies among the hyperplanes, which occur for $N \geq 4$, we are not able to find a closed form expression for $dim(\bigcap S_k)$ and instead have to resort to an upper bound for N_{regions} (Orlik and Terao, 1992; Sauer, 1972):

$$N_{\text{regions}} \leq \sum_{k=0}^{N} \binom{S}{k} \tag{7}$$

2. In this calculation we ignored the fact that a hyperplane or parts of it can become a solution space itself. This can occur when there are two training examples s' and s'' for which $\boldsymbol{x}_{s'}$ and $\boldsymbol{x}_{s''}$ differ by a global sign but have the same label $y_{s'} = y_{s''}$. Since this case is exceedingly unlikely, the probability being $O(S/2^{(2N)})$, we can afford not to consider this situation.

It is possible that multiple training examples generate identical inequality constraints for $\boldsymbol{w}$. So, (7) is a conservative estimate as the actual number of solution spaces is often lower.

Discrete weight configurations with a finite bit-depth d_w lie on an N-dimensional hypercubic lattice with edges that have 2^{d_w} vertices. This gives a total of $2^{d_w N}$ vertices on the lattice. If each solution region contains a lattice vertex then all classifiers that can be attained with real-valued weights can also be realized by the discrete weight configurations. Thus, one obtains the necessary bit-depth by demanding that the number of vertices on the lattice is at least as large as N_{regions}, the number of solution regions created by the hyperplanes:

$$\begin{aligned}
&\frac{\text{Vertices on Lattice}}{\text{Regions in Positive Quadrant}} \approx \frac{(2^{d_w})^N}{N_{\text{regions}}} \geq \frac{2^{d_w N}}{\sum_{k=0}^{N}\binom{S}{k}} \geq \frac{2^{d_w N}}{(\frac{eS}{N})^N} = \frac{2^{d_w N} N^N}{(eS)^N} \overset{!}{\geq} 1 \\
&\Rightarrow \left(\frac{2^{d_w} N}{eS}\right)^N = \left(\frac{2^{d_w} N}{efN}\right)^N = \left(\frac{2^{d_w}}{ef}\right)^N \overset{!}{\geq} 1 \\
&\Rightarrow d_w \geq \log_2(f) + \log_2(e) \ ,
\end{aligned} \tag{8}$$

where e is the Euler number and $f = S/N$. In (8) we used a standard result regarding binomial coefficients: $\sum_{k=0}^{N}\binom{S}{k} \leq (\frac{eS}{N})^N$. This holds in the case of $S \geq N$. Smaller numbers of training examples lead to even better bounds than (8). This is an important result as it shows that the required bit depth for weight variables only grows logarithmically with the ratio of the number of training examples to the number of features. Thus for many problems that arise in practice we can get away with very few bits, and often only a single bit may suffice.

4. Comparison to AdaBoost

In the case of a finite dictionary of weak classifiers $\{h_i(\boldsymbol{x}) | i = 1, \ldots, N\}$, AdaBoost can be seen as a greedy minimization of the exponential loss (Zhang, 2008):

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} \left(\sum_{s=1}^{S} \exp\left(-y_s \sum_{i=1}^{N} \alpha_i h_i(x_s) \right) / S \right) \ , \tag{9}$$

with $\alpha_i \in \mathbb{R}_{>0}$. There are two differences between the baseline objective (4) and the one employed by AdaBoost. The first is that we use ℓ_0-norm regularization. Second, we employ square loss, while AdaBoost works with the exponential loss.

It can be shown that including ℓ_0-norm regularization in the objective (4) leads to improved generalization as compared to using square loss only. An upper bound for the *Vapnik-Chervonenkis dimension* of a strong classifier of the form $H(\boldsymbol{x}) = \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})$ is

$$VC_H = 2(VC_{\{h_t\}} + 1)(T + 1)\log_2(e(T + 1)) \ , \tag{10}$$

where $VC_{\{h_t\}}$ is the VC dimension of the weak classifiers (Freund and Shapire, 1995)—a complexity measure of the class of functions the dictionary can represent. Then the strong classifier's generalization error has the upper bound (Vapnik and Chervonenkis, 1971):

$$\text{Error}_{\text{test}} \leq \text{Error}_{\text{train}} + \sqrt{\frac{VC_H \ln(\frac{2S}{VC_H} + 1) + ln(\frac{9}{\delta})}{S}} \ . \tag{11}$$

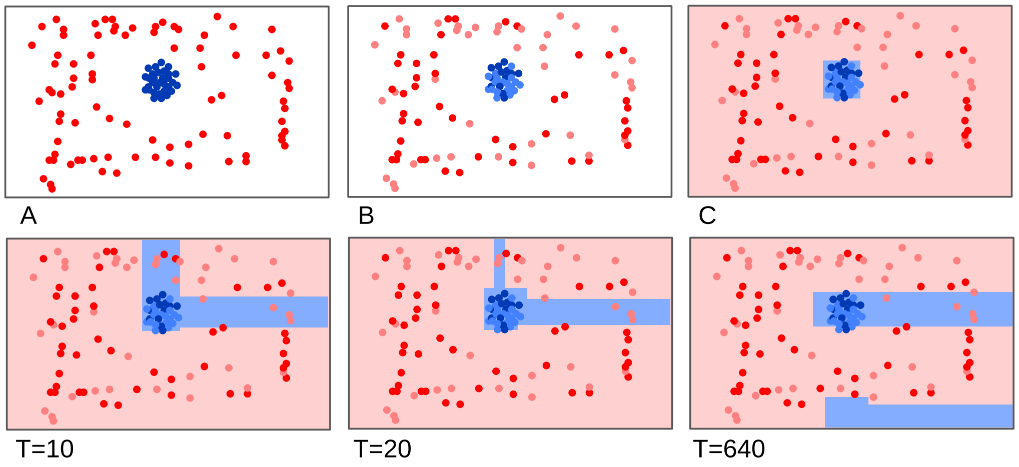

Figure 2: AdaBoost applied to a simple classification task. **A** shows the data, a separable case consisting of a two-dimensional cluster of positive examples (blue) surrounded by negative ones (red). **B** shows the random division into training (saturated colors) and test data (light colors). The dictionary of weak classifiers is constructed out of axis-parallel one-dimensional hyperplanes. **C** shows the optimal classifier for this situation, which employs four weak classifiers to partition the input space into positive and negative areas. The lower row shows partitions generated by AdaBoost after 10, 20, and 640 iterations. The configuration at $T = 640$ is the asymptotic one, which does not change anymore in subsequent iterations. The breakout regions outside the bounding box of the positive cluster occur in areas in which the training set does not contain negative examples. This problem becomes more severe for higher-dimensional data.

Apparently a more compact strong classifier that achieves a given $\text{Error}_{\text{train}}$ with a smaller number T of weak classifiers—hence, with a smaller VC_H—comes with a guarantee for lower generalization error. Looking at the optimization problem (4), one can see that if the regularization parameter λ is chosen weak enough, i.e. $\lambda < \frac{2}{N} + \frac{1}{N^2}$, then the effect of regularization is merely to thin out the strong classifier without sacrificing training accuracy. One arrives at the condition for λ by demanding that the reduction of the regularization term $\Delta\Omega(\dot{\boldsymbol{w}})$ that can be obtained by switching a $\dot{w}_i$ to zero is smaller than the smallest associated increase in empirical risk $\Delta R(\dot{\boldsymbol{w}})$ that comes from incorrectly classifying a training example. This condition guarantees that weak classifiers are not eliminated at the expense of a higher training error. Therefore regularization keeps only a minimal set of components—those which are needed to achieve the minimal training error that can be obtained when using the loss term only. In this regime the VC bound of the resulting strong classifier is lower than or equal to the VC bound of a classifier trained without regularization.

AdaBoost does not explicitly enforce sparsity, so the classifier may use a richer set of weak classifiers than needed for the minimal training error, which in turn leads to degraded generalization. Fig. 2 illustrates this fact for hand-crafted data with simple structure. Due to AdaBoost's greedy approach, the optimal configuration for this data is not found despite the fact that the weak classifiers necessary to construct the ideal classifier are generated.

In practice we do not operate in the weak λ regime but rather determine the regularization strength by cross-validation. We measure the performance of the classifier for different values of λ on a validation set and then choose the one with minimal validation error. In this regime the optimization may perform a trade-off and accept some higher empirical loss if the classifier can be kept more compact. In other words, it may choose to misclassify training examples if the classifier can be kept simpler. This leads to increased robustness in the case of noisy data, and indeed we observe the most significant gains over AdaBoost for noisy data sets when the Bayes error is high. The fact that boosting in its standard formulation with convex loss and no explicit regularization is not robust against label noise has drawn attention recently (Long and Servedio, 2010; Freund, 2009). We also experimented with a version of AdaBoost with explicit ℓ_1 regularization (Duchi and Singer, 2009), but that did not perform better than plain AdaBoost on our data.

The second difference to the baseline objective, namely that it employs quadratic loss while AdaBoost works with exponential loss, is of smaller importance. In fact, the discussion above about the role of regularization does not change if we replace square loss by exponential loss. The literature agrees that the use of exponential loss in AdaBoost is not essential and that other loss functions could yield classifiers with similar performance (Friedman et al., 1998; Wyner, 2002). From a statistical perspective, square loss is satisfactory since a classifier that minimizes it is *Bayes consistent*, i.e. with increasing numbers of training examples it asymptotically approaches the Bayes-optimal classifier (Zhang, 2008).

5. Large Scale Classifiers by QBoost

The disadvantage of the baseline (4) is that it assumes a small enough fixed dictionary of weak classifiers, so that all weight variables can be considered in a single AQO run. This approach needs to be modified if the goal is to train a large-scale classifier. *Large scale* here means that either the dictionary contains more weak classifiers than what can be considered in a single AQO run, or the final strong classifier consists of a number of weak classifiers that exceeds the number of variables that can be handled at once. Typical problem sizes usually satisfy both conditions. The state-of-the-art commercial solver CPLEX operates in heuristic mode for the problems we study (mixed integer quadratic programs) and can solve problems in reasonable time for up to 10^3 variables. Existing quantum hardware currently can handle 512-variable problems. In order to train a strong classifier we often sift through millions of features. Moreover, dictionaries of weak learners are often dependent on various continuous parameters, which makes their cardinality effectively infinite. We estimate that typical classifiers employed in vision-based products today use thousands of weak learners. Therefore, it is not possible to determine all weights in a single AQO run, but rather it is necessary to break the problem into smaller chunks.

Let T denote the size of the final strong classifier and Q the number of variables that a single optimization run can handle. Q is determined by the number of available qubits, or if classical solvers such as CPLEX or Tabu search (Palubeckis, 2004) are employed, then Q denotes the maximum problem size for which optimal solutions can be obtained in reasonable time. QBoost Algorithms 1 and 2 consider two cases: $T \leq Q$ and $T > Q$.

Algorithm 1 $T \leq Q$ (QBoost Inner Loop)

```
Require: Training and validation data $\{\boldsymbol{x}_s\}$, dictionary of weak classifiers $\{h_i(\boldsymbol{x})\}$, regularization parameters $\lambda_{\text{min}}$, $\lambda_{\text{step}}$, and $\lambda_{\text{max}}$
Ensure: Strong classifier $H_{\dot{\boldsymbol{w}}^*}(\boldsymbol{x})$
 1: Initialize: $\forall s, d_{\text{inner}}(s) = \frac{1}{S}$; $T_{\text{inner}} = 0$; empty strong classifier $H_{\dot{\boldsymbol{w}}^*}(\boldsymbol{x})$; storage for a pool of $Q$ candidate weak learners $\{h_q\}$
 2: repeat
 3:    Optimize the members of the dictionary $\{h_i\}$ according to the current $d_{\text{inner}}$
 4:    From $\{h_i\}$ select the $Q - T_{\text{inner}}$ weak classifiers that have the smallest training error rates weighted by $d_{\text{inner}}$ and add them to the pool $\{h_q\}$
 5:    for $\lambda = \lambda_{\text{min}} : \lambda_{\text{step}} : \lambda_{\text{max}}$ do
 6:       Optimize $\dot{\boldsymbol{w}}^* = \arg\min_{\dot{\boldsymbol{w}}} \left\{ \sum_{s=1}^S \left( y_s \sum_{q=1}^Q \dot{w}_q h_q(\boldsymbol{x}_s) - 1 \right)^2 + \lambda \|\dot{\boldsymbol{w}}\|_0 \right\}$
 7:       Set $T_{\text{inner}} = \|\dot{\boldsymbol{w}}^*\|_0$
 8:       Construct strong classifier $H_{\dot{\boldsymbol{w}}^*}(\boldsymbol{x}) = \text{sign}\left( \sum_{q=1}^Q \dot{w}_q^* h_q(\boldsymbol{x}) \right)$
 9:       Measure validation error $\text{Error}_{\text{val}}$ of $H_{\dot{\boldsymbol{w}}^*}(\boldsymbol{x})$ on unweighted validation set
10:    end for
11:    Save $\dot{\boldsymbol{w}}^*$, $T_{\text{inner}}$, $H_{\dot{\boldsymbol{w}}^*}(\boldsymbol{x})$ and $\text{Error}_{\text{val}}$ from the optimization run that has yielded the lowest validation error so far
12:    Update $d_{\text{inner}}(s) = d_{\text{inner}}(s) \left( y_s \sum_{q=1}^Q \dot{w}_q^* h_q(\boldsymbol{x}_s) - 1 \right)^2$
13:    Normalize $d_{\text{inner}}(s) = \frac{d_{\text{inner}}(s)}{\sum_{s=1}^S d_{\text{inner}}(s)}$
14:    Delete from the pool $\{h_q\}$ the $Q - T_{\text{inner}}$ weak learners for which $\dot{w}_q^* = 0$
15: until validation error $\text{Error}_{\text{val}}$ stops decreasing
```

Algorithm 2 $T > Q$ (QBoost Outer Loop)

```
Require: Training and validation data $\{\boldsymbol{x}_s\}$, dictionary of weak classifiers $\{h_i(\boldsymbol{x})\}$, regularization parameters $\lambda_{\text{min}}$, $\lambda_{\text{step}}$, and $\lambda_{\text{max}}$
Ensure: Strong classifier $H_{\dot{\boldsymbol{w}}^*_{\text{outer}}}(\boldsymbol{x})$
 1: Initialize: $\forall s, d_{\text{outer}}(s) = \frac{1}{S}$; $T_{\text{outer}} = 0$; empty strong classifier $H_{\dot{\boldsymbol{w}}^*_{\text{outer}}}(\boldsymbol{x})$; storage for selected weak learners $\{h_t\}$
 2: repeat
 3:    Run Algorithm 1 with $\{\boldsymbol{x}_s\}$, $\{h_i(\boldsymbol{x})\}$, $\lambda_{\text{min}}$, $\lambda_{\text{step}}$, $\lambda_{\text{max}}$, initializing $d_{\text{inner}}$ from $d_{\text{outer}}$, and using an objective function that takes into account current $H_{\dot{\boldsymbol{w}}^*_{\text{outer}}}(\boldsymbol{x})$:
       $\dot{\boldsymbol{w}}^* = \arg\min_{\dot{\boldsymbol{w}}} \left\{ \sum_{s=1}^S \left( y_s \left( \sum_{t=1}^{T_{\text{outer}}} \dot{w}^*_{\text{outer},t} h_t(\boldsymbol{x}_s) + \sum_{q=1}^Q \dot{w}_q h_q(\boldsymbol{x}_s) \right) - 1 \right)^2 + \lambda \|\dot{\boldsymbol{w}}\|_0 \right\}$
 4:    $H_{\dot{\boldsymbol{w}}^*_{\text{outer}}}(\boldsymbol{x}) = \text{sign}\left( \sum_{t=1}^{T_{\text{outer}}} \dot{w}^*_{\text{outer},t} h_t(\boldsymbol{x}) + \sum_{q=1}^Q \dot{w}_q^* h_q(\boldsymbol{x}) \right)$
 5:    Set $\dot{\boldsymbol{w}}^*_{\text{outer}} = [\dot{\boldsymbol{w}}^*_{\text{outer}}, \dot{\boldsymbol{w}}^*]$
 6:    Set $\{h_t\} = \{\{h_t\}, \{h_q\}\}$
 7:    Set $T_{\text{outer}} = T_{\text{outer}} + T_{\text{inner}}$
 8:    Update $d_{\text{outer}}(s) = d_{\text{outer}}(s) \left( y_s \sum_{t=1}^{T_{\text{outer}}} \dot{w}^*_{\text{outer},t} h_t(\boldsymbol{x}) - 1 \right)^2$
 9:    Normalize $d_{\text{outer}}(s) = \frac{d_{\text{outer}}(s)}{\sum_{s=1}^S d_{\text{outer}}(s)}$
10: until validation error $\text{Error}_{\text{val}}$ stops decreasing
```

A way to think about Algorithm 1 is to see it as an enrichment process. In the first iteration the algorithm selects those T_{inner} weak classifiers out of subset of Q that produce the optimal validation error. The subset of Q weak classifiers is preselected from a dictionary with cardinality possibly much larger than Q. In the next step the algorithm fills the remaining $Q - T_{\text{inner}}$ empty slots in the solver with the best weak classifiers drawn from a modified dictionary that was adapted by taking into account for which examples the strong classifier constructed in the first iteration is already good and where it still makes errors. This is the boosting idea (Freund and Schapire, 1999). Under the assumption that the solver always finds the global minimum, it is guaranteed that for a given λ the solutions found in the subsequent round have lower or equal objective value, i.e. they achieve lower empirical risk, represent a more compact strong classifier, or stay the same. The fact that the algorithm considers groups of Q weak classifiers simultaneously (instead of augmenting the strong classifier by one) and then tries to find a subset that produces low empirical risk, allows it to find good configurations more efficiently. If the validation error cannot be decreased any further using the inner loop, one may conclude that more weak classifiers are needed for constructing the strong one. In this case QBoost Algorithm 2 "freezes" the classifier obtained so far and adds another partial classifier trained by the inner loop.

6. Experimental Results

In this section we present experimental results from training with QBoost on three different data sets. The first two can be considered toy problems of limited practical importance, so we train classifiers for them by a classical heuristic algorithm serving as a stand-in for quantum hardware. The third data set represents a truly difficult large-scale practical problem, and for it we use an actual quantum chip consisting of 52 operational qubits.

6.1. Optimization by Classical Heuristic

We assess the performance of classifiers trained by QBoost Algorithm 2 on synthetic and natural data sets. The synthetic data was sampled in 30 dimensions from
$P(\boldsymbol{x}, y) = \frac{1}{2}\delta(y-1)N(\boldsymbol{x}|\boldsymbol{\mu}_+, \boldsymbol{I}) + \frac{1}{2}\delta(y+1)N(\boldsymbol{x}|\boldsymbol{\mu}_-, \boldsymbol{I}/2)$, where $N(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is a spherical Gaussian having mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$. This data captures generic characteristics of classification problems via the usage of an overlap coefficient that determines the separation of the two Gaussians. The natural data consists of 30-dimensional vectors of Gabor wavelet amplitudes extracted at eye locations in images showing faces. The data sets consist of $2 \cdot 10^4$ input vectors, which we divide evenly into training, validation, and test sets. We use Tabu search as a classical heuristic stand-in for quantum hardware, discretize the weight variables with $d_w = 1$, and employ a dictionary of decision stumps:

$$h^1_{l,p}(\boldsymbol{x}) = \text{sign}\left((-1)^p x_l - \Theta_{p,l}\right) \text{ for } l = 1:M; p \in \{0,1\}$$
$$h^2_{l,p}(\boldsymbol{x}) = \text{sign}\left((-1)^p x_i x_j - \Theta_{p,i,j}\right) \text{ for } l = 1:M(M-1)/2; i,j = 1:M; i<j; p \in \{0,1\}$$

Here M is the dimensionality of the input vector $\boldsymbol{x}$; x_l, x_i, x_j are elements of $\boldsymbol{x}$; and $\Theta_{p,l}$, $\Theta_{p,i,j}$ are optimal thresholds. For the 30-dimensional input data the dictionary employs 930 weak classifiers, and for the 96-dimensional data it consists of 9312 weak learners.

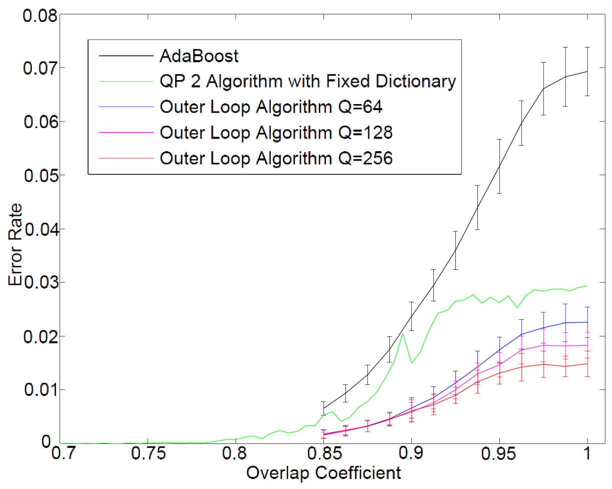

Figure 3: Test errors for the synthetic data set. We ran the outer loop algorithm for three different values of Q: 64, 128, 256. The plots show means over 100 runs and the error bars indicate the corresponding standard deviations. All three versions outperform AdaBoost. The gain increases as the classification problem gets harder i.e. as the overlap between the two Gaussians increases. "Overlap Coefficient" = 1 means that $\boldsymbol{\mu}_+ = \boldsymbol{\mu}_-$, and the Bayes error rate for this case is ≈ 0.05. One can also see that there is a benefit to being able to run larger optimizations since the error rate decreases with increasing Q. For comparison we also included results for a classifier trained on a fixed dictionary (QP 2) for which the training was performed as per (4).

Table 1: Results obtained from the eye data. Similarly to the synthetic data, we compare the outer loop algorithm for three different optimization sizes ($Q = 64, 128, 256$) to AdaBoost. The means and standard deviations are obtained from 10^3 runs. Here QBoost leads to only sightly lower test errors but obtains those with a significantly reduced number of weak classifiers. Also, the number of iterations needed in training is more than 4 times lower than what is required by AdaBoost.

	AdaBoost	QBoost Outer Loop		
		$Q = 64$	$Q = 128$	$Q = 256$
Test error	0.258± 0.006	0.254± 0.010	0.249± 0.010	0.246± 0.009
Weak classifiers	257.8± 332.1	116.8± 139.0	206.1± 241.8	356.3± 420.3
Reweightings	658.9± 209.3	130.8± 65.3	145.6± 65.5	159.8± 63.7
Training error	0.038± 0.054	0.185± 0.039	0.170± 0.039	0.158± 0.040
Outer loops		11.9± 5.0	12.2± 4.6	12.6± 4.4

Test results for the synthetic data are shown in Fig. 3, and Table 1 displays results obtained from the natural data. We use a validation set to determine the optimal size T of the strong classifier generated by AdaBoost.

6.2. Optimization by Quantum Hardware

In this experiment we applied QBoost to the training of a detector for cars in digital images and employed an early-generation D-Wave chip (Rainier) to solve optimization problems of size $Q = 52$ by AQO. The training and test sets consist of $2 \cdot 10^4$ images with roughly half the images in each set containing cars in side-view positions and the other half containing city streets and buildings without cars. The images containing cars are human-labeled ground truth data with tight bounding boxes drawn around each car. The actual data seen by the training system is obtained by randomly sampling subregions of the input training and test data to obtain about 10^5 patches.

Starting from a highly tuned training system implementing cascaded confidence-rated boosting (Schapire and Singer, 1998; Viola and Jones, 2004), we replaced its feature-selection routine by QBoost. For each stage the original system calls the feature-selection routine, requesting a fixed number T of features for that stage. The numbers of features for different stages are determined by experience and typically involve smaller numbers for earlier stages and larger numbers for later stages in order to achieve the optimal tradeoff between accuracy and processing time during evaluation of the final detector. The guiding principle is to reduce the false positives by 50% at each stage of the cascade.

Consequently, for each stage of the cascade QBoost is invoked with a request for selecting T weak classifiers out of the several million weak classifiers that compose the dictionary for this problem. We keep the optimization size Q fixed at 52, which is the maximum problem size that the available quantum hardware could handle. Since T can be both smaller and larger than 52, early stages utilize only Algorithm 1, while later stages invoke Algorithm 2.

We found that training using the quantum hardware led to a classifier that produces useful classification accuracy. But more importantly, as shown in Fig. 4, the accuracy obtained by the hardware solver is better than the accuracy of the classifier trained with Tabu search. However, the classifier obtained with the hardware solver—even though it is sparser by about 10%—has a false negative rate approximately 10% higher than the boosting framework we employed as a point of departure. A possible reason for that is the extremely small Q relative to the number of weak learners as here we do selection from a dictionary consisting of several million weak classifiers. Future training runs with larger hardware are expected to produce drastically better results.

7. Scaling Analysis via Quantum Monte Carlo Simulation

We used the quantum Monte Carlo (QMC) simulator of Farhi et al. (2009) to estimate the time complexity of AQO on the optimization problems that QBoost generates. According to the adiabatic theorem (Messiah, 1999), the ground state of the problem Hamiltonian H_P is found with high probability by AQO, provided that the evolution time $\mathcal{T}$ from the initial Hamiltonian H_B to H_P is $\Omega(g_{\text{min}}^{-2})$, where g_{min} is the *minimum gap*. Here $H_B = \sum_{i=1}^{N}(1 - \sigma_i^x)/2$, where σ^x is a Pauli matrix. The minimum gap is the smallest energy gap between the ground state E_0 and first excited state E_1 of the time-dependent Hamiltonian

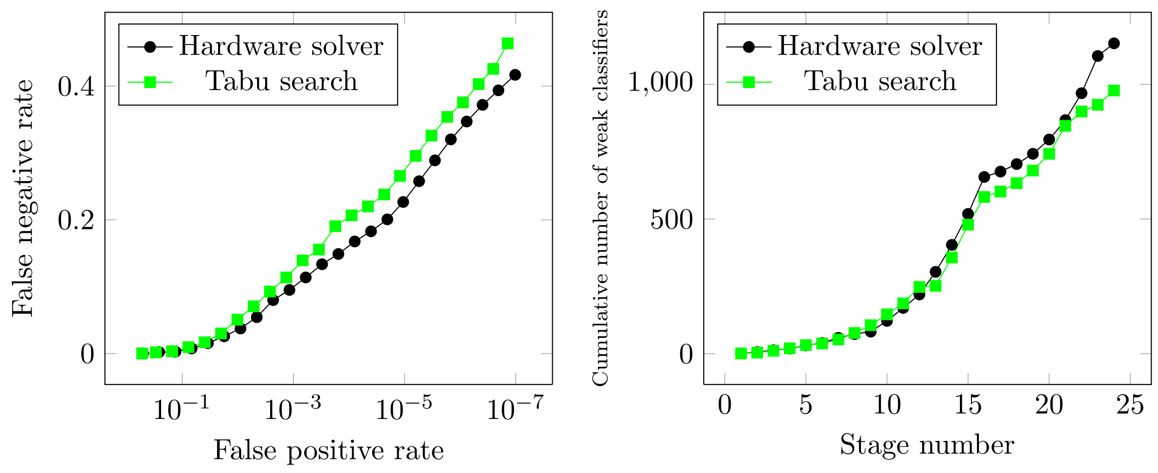

Figure 4: Error rates and sparsity levels for Tabu search as compared to the quantum hardware solver. The training consisted of 24 cascaded stages. Each stage is trained on the false negatives of the previous stage as its positive examples complemented with true negative examples unseen so far (Viola and Jones, 2004). The error rates of the hardware solver are superior to the ones obtained by Tabu search across all stages. The reported error rates are per pixel errors. The cumulative number of weak classifiers employed at a certain stage does not suggest a clear advantage for either method.

$H(t) = (1 - t/\mathcal{T})H_B + (t/\mathcal{T})H_P$ at any $0 \le t \le \mathcal{T}$. For notational convenience we also use $\tilde{H}(s) = (1 - s)H_B + sH_P$ with $0 \le s \le 1$. More details can be found in the seminal work Farhi et al. (2000).

To find the time complexity of AQO for a given objective function, one needs to estimate the asymptotic scaling of the minimum gap as observed on a collection of average-case instances of corresponding problems. As noted in Amin and Choi (2009), the task of analytically extracting the minimum gap scaling has been extremely difficult in practice, except for a few special cases. The only alternative is to resort to numerical methods, which consist of diagonalization and QMC simulation. Unfortunately, diagonalization is currently limited to about $N = 30$, and QMC to about $N = 256$, where N is the number of binary variables (Young et al., 2010). Hence, the best that can be done with the currently available tools is to compute the minimum gap via QMC simulations on small problem instances and attempt to extrapolate the scaling of the time complexity for larger instances.

Using the QMC simulator of Farhi et al. (2009), we indirectly measured the minimum gap by estimating the magnitude of the second derivative of the ground state energy with respect to s, which is related to the minimum gap (Young et al., 2008). This quantity is an upper bound on $2|V_{01}|^2/g_{\min}$, where $V_{01} = \langle\Psi_0|\, d\tilde{H}/ds\, |\Psi_1\rangle$ and Ψ_0, Ψ_1 are the eigenstates corresponding to the ground and first excited states of $\tilde{H}$. However, the quantity we seek for the time scaling of AQO is $|V_{01}|/g^2_{\min}$. Nevertheless, assuming that the matrix element V_{01} is

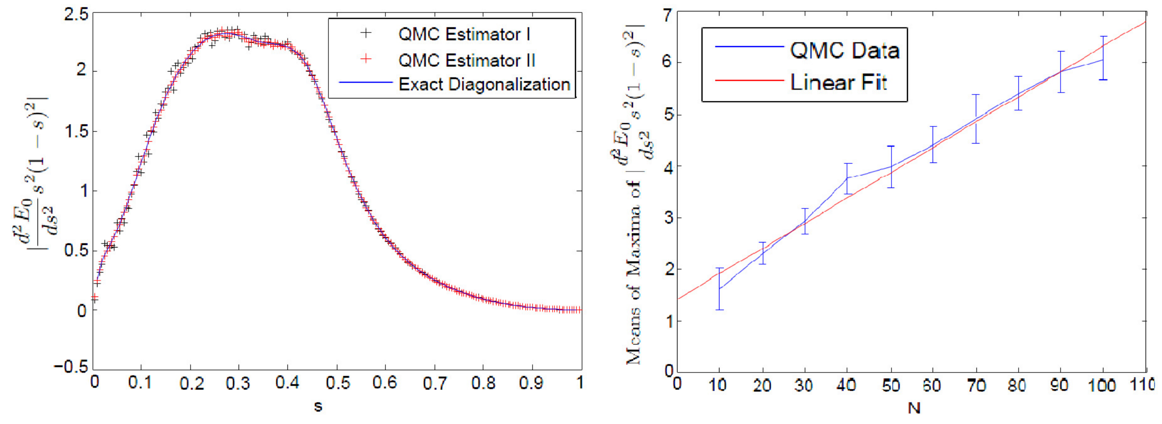

Figure 5: **Left:** The quantity $|\frac{d^2E_0}{ds^2}s^2(1-s)^2|$ determined by quantum Monte Carlo simulation as well as exact diagonalization for a training problem with 20 weight variables. For small problem instances with less than 30 variables we can determine this quantity via exact diagonalization of the Hamiltonian $\tilde{H}(s)$. As one can see, the results obtained by diagonalization coincide very well with the ones determined by QMC. The training objective is given by (4) using the synthetic data set with an overlap coefficient of 0.95. **Right**: A plot of the peaks of the mean of $|\frac{d^2E_0}{ds^2}s^2(1-s)^2|$ against the number of qubits for values in $[10, 100]$. The error bars indicate standard deviation. Each point of the plot represents 20 QMC runs. The data is well fitted by a linear function. From the fact that the scaling is at most polynomial in the problem size we can infer that the minimum gap and hence the run time of AQO are scaling polynomially as well.

not extremely small, the scaling of the second derivative—polynomial or exponential—can be used to infer if the time complexity of AQO is polynomial or exponential in N.

Fig. 5 shows our scaling analysis for the synthetic data set. The result is encouraging as the maxima of $|\frac{d^2E_0}{ds^2}s^2(1-s)^2|$ seem to only scale linearly with the problem size. This implies that the runtime of AQO on this data set is likely to scale at most polynomially.

It is not possible to judge how typical this data set and scaling behavior are. We know from related experiments with known optimal solutions that Tabu search often fails to obtain the optimal solution for training problems of sizes as small as 64 variables. Obviously, scaling should depend on the input data. In fact, it is possible to take a hard problem known to defeat AQO (Amin and Choi, 2009) and encode it as a training problem, which would cause the scaling to become exponential. Nevertheless, the minimum gap between ground and first excited state energies shrinking exponentially with increasing problem size only implies that the system is unlikely to be found in its ground state at the end of the adiabatic evolution if completed in polynomial time. But this does not say anything about the quality of the resulting approximate solutions. Due to quantum tunneling, approximate solutions found by AQO can still be significantly better than those found by classical solvers. Further, results from approximability theory (Zuckerman, 2006) imply that obtaining such approximate solutions may still be NP-hard, i.e. finding a low-lying excited state of H_P may still be difficult for classical algorithms but not for AQO. Finally, alternative versions of AQO—e.g. Farhi et al. (2009)—may be overcoming the known failure cases.

8. Conclusion

We presented AQO as a discrete optimization method for training binary classifiers. The proposed algorithm, QBoost, enables handling training tasks of large sizes as they occur in production systems today. QBoost offers gains over standard AdaBoost in three aspects: i. lower generalization error; ii. faster classification (employing fewer weak classifiers); iii. smaller computational effort for training (fewer boosting iterations).

In all experiments we find that the classifier constructed with discrete optimization is significantly more compact. The gain in accuracy however is more varied. The good performance of a large scale binary classifier trained using iterated discrete optimization in a form amenable to AQO but solved with classical heuristics shows that it is possible to map the training of a classifier to AQO with negative translation costs. Any improvements by AQO to the solution of the training problem will directly increase the advantage of the algorithm proposed here over conventional approaches. Access to emerging hardware that realizes AQO is needed to establish the size of the gain over classical solvers. We already completed the first such experiment with 52 operational qubits, but processors with larger qubit numbers will be needed in order to see a more pronounced effect.

Further, we conclude that bit-constrained learning machines can be expected to exhibit better generalization than what is obtained when the weight variables are represented with higher precision. To the best of our knowledge this has not been studied before. Bit-constraining can be regarded as intrinsic regularization that contributes to keeping the model complexity low. Our finding that the bit-depth needed for realizing the optimal classifier only grows logarithmically with the ratio of training examples to features, supplies insight into why few-bit learning machines work. The competitive performance of bit-constrained classifiers suggests that training benefits from being treated as an integer program. This has a twofold implication. First, it is good news for hardware-constrained implementations such as cell phones, sensor networks, or early quantum chips with small numbers of qubits. Second, this renders the training problem manifestly NP-hard, thus further motivating the application of quantum algorithms that may generate better approximate solutions than classically available.

We close with the remark that the study of bit-constrained learning may have implications for understanding plasticity in the nervous system, where it is still unknown how a synapse can store information reliably over a long period of time (Kandel et al., 2000).

Acknowledgments

We would like to thank Hartwig Adam, Xiaowei Li, and Jiayong Zhang for help with data preparation; Alessandro Bissacco, Ulrich Buddemeier, and Jiayong Zhang for assistance with MapReduce and boosting code; Edward Farhi and David Gosset for support with QMC simulations; Hartwig Adam, Boris Babenko, Ulrich Buddemeier, Vint Cerf, Corinna Cortes, Edward Farhi, Yoram Singer, and Jiayong Zhang for useful comments.

References

D. Aharonov, W. van Dam, J. Kempe, Z. Landau, S. Lloyd, and O. Regev. Adiabatic quantum computation is equivalent to standard quantum computation. *Foundations of Computer Science, IEEE Annual Symposium on*, 0:42–51, 2004. ISSN 0272-5428.

M. H. S. Amin and V. Choi. First order quantum phase transition in adiabatic quantum computation, 2009. arXiv: quant-ph/0904.1387.

S. Brush. History of the Lenz-Ising model. *Reviews of Modern Physics*, 39:883–893, 1967.

V. S. Denchev, N. Ding, S. V. N. Vishwanathan, and H. Neven. Robust classification with adiabatic quantum optimization. 2012. arXiv:1205.1148v2.

N. G. Dickson and M. H. S. Amin. Algorithmic approach to adiabatic quantum optimization, 2011. arXiv:1108.3303v1.

J. Duchi and Y. Singer. Boosting with structural sparsity. In *Proceedings of the 26th International Conference on Machine Learning*, 2009.

E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser. Quantum computation by adiabatic evolution. 2000. arXiv:quant-ph/0001106v1.

E. Farhi, J. Goldstone, S. Gutmann, J. Lapan, A. Lundgren, and D. Preda. A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem. *Science*, 292:472–476, 2001.

E. Farhi, J. Goldstone, D. Gosset, S. Gutmann, H. B. Meyer, and P. Shor. Quantum adiabatic algorithms, small gaps, and different paths. 2009. arXiv: quant-ph/0909.4766v1.

V. Feldman, V. Guruswami, P. Raghavendra, and Y. Wu. Agnostic learning of monomials by halfspaces is hard. 2010. arXiv:1012.0729v1.

Y. Freund. A more robust boosting algorithm. 2009. arXiv:0905.2138v1.

Y. Freund and R. E. Schapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780, 1999.

Y. Freund and R. E. Shapire. A decision-theoretic generalization of online learning and an application to boosting. *AT&T Bell Labs Technical Report*, 1995.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28:2000, 1998.

M. W. Johnson, M. H. S. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, E. M. Chapple, C. Enderud, J. P. Hilton, K. Karimi, E. Ladizinsky, N. Ladizinsky, T. Oh, I. Perminov, C. Rich, M. C. Thom, E. Tolkacheva, C. J. S. Truncik, S. Uchaikin, J. Wang, B. Wilson, and G. Rose. Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198, May 2011.

E. R. Kandel, J. H. Schwartz, and T. M. Jessell. *Principles of Neural Science.* McGraw-Hill, 2000.

P. M. Long and R. A. Servedio. Random classification noise defeats all convex potential boosters. *Machine Learning Journal*, 783:287–304, 2010.

A. Messiah. *Quantum mechanics: Two volumes bound as one.* Dover, Mineola, NY, 1999. Trans. of : Mecanique quantique, t.1. Paris, Dunod, 1959.

P. Orlik and H. Terao. *Arrangements of hyperplanes.* Fundamental Principles of Mathematical Sciences. Springer-Verlag, Berlin, Germany, 1992. ISBN 3-540-55259-6.

G. Palubeckis. Multistart tabu search strategies for the unconstrained binary quadratic optimization problem. *Annals of Operations Research*, 131:259–282, 2004.

G. Rose. The D-Wave One quantum computer: Technology and applications. Presentation at International Conference for High Performance Computing, Networking, Storage and Analysis (SC). http://dwave.files.wordpress.com/2011/11/20111117_sc11_rose.pdf, November 2011.

G. E. Santoro, R. Martonak, E. Tosatti, and R. Car. Theory of quantum annealing of an ising spin glass. *Science*, 295(5564):2427–2430, 2002.

N. Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13:145–147, 1972.

R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. In *COLT*, pages 80–91, 1998.

V. Vapnik. *Statistical learning theory.* Wiley, 1998.

V. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

P. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

A. J. Wyner. Boosting and the exponential loss. *Proceedings of the Ninth Annual Conference on AI and Statistics*, 2002.

A. P. Young, S. Knysh, and V. N. Smelyanskiy. Size dependence of the minimum excitation gap in the quantum adiabatic algorithm. *Physical Review Letters*, 101:170503, 2008.

A. P. Young, S. Knysh, and V. N. Smelyanskiy. First order phase transition in the quantum adiabatic algorithm. *Physical Review Letters*, 104:020502, 2010.

T. Zhang. Forward-backward greedy algorithm for learning sparse representations. *Rutgers Statistics Department Technical Report*, 2008.

D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *STOC*, pages 681–690, 2006.