23

Multiplicative weights update method

The authors are grateful to Sander Gribling for reviewing this chapter.

Rough overview (in words)

The multiplicative weights update (MWU) method is an algorithmic strategy, sometimes referred to as a "meta-algorithm," with varying applications in classical and quantum algorithms. Reference [58] gives an overview of the MWU strategy. The introductory example problem where the MWU method is used is the problem of making predictions for a binary outcome given advice from a panel of n "experts." The MWU approach assigns a weight to each of the n experts, and the weight is reduced by a multiplicative factor whenever the expert makes an incorrect prediction. The outcome of the process can be shown to give an approximately optimal strategy.

This general approach can be applied to convex programs including linear programs (LPs) and semidefinite programs (SDPs). The SDP version generalizes the MWU method to allow for matrix-valued weights and matrix-valued costs. These weight matrices are positive semidefinite operators with trace equal to one, that is, density matrices. In fact, the states that arise in the SDP-solving algorithm are Gibbs states. Thus, they can be naturally represented as quantum states on a logarithmic number of qubits and generated through the process of Gibbs sampling. The existence of fast Gibbs samplers can lead to a quantum speedup in certain circumstances.

Rough overview (in math)

We present an example problem. Let $\mathbf{1}$ denote the all-ones vector. Consider the following set of linear constraints on the vector $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$

$$\begin{aligned} \langle a^{(j)}, x\rangle &\geq 0 \qquad j = 1, \ldots, m \\ \langle \mathbf{1}, x\rangle &= 1 \\ x_i &\geq 0 \qquad i = 1, \ldots, n \end{aligned}$$

for m fixed vectors $a^{(j)} \in \mathbb{R}^n$ with entries in $[-1, 1]$, for $j = 1, \ldots, m$, where $\langle \cdot, \cdot \rangle$ denotes the standard dot product between vectors. Suppose we are given a value of ϵ and promised either that there is no choice of x that satisfies all the constraints or that there exists an x^* such that $\langle a^{(j)}, x^*\rangle \geq \epsilon$ for all j, with $\langle \mathbf{1}, x^*\rangle = 1$ and $x_i^* \geq 0$ for all i. We wish to determine which is the case and find a vector x^* in the second case. This problem is equivalent to the machine learning problem of finding a linear classifier for a set of m labeled n-dimensional training points, similar to a support vector machine [58, 697]. The problem is also similar to the form of an LP and to the problem of solving for the optimal point of a zero-sum game [58, 46], and the MWU meta-algorithm can also be straightforwardly applied to solve these problems.

A classical solution to this problem is given by the multiplicative weights method [58]. The algorithm iteratively updates the vector x, with initialization $x = \mathbf{1}/n$. At each iteration, the algorithm finds a constraint j for which $\langle a^{(j)}, x\rangle < 0$ (or if no such j exists, it terminates and outputs x). Let $\eta = O(\epsilon)$ be a fixed constant. Once j is found, the entries of the vector x are updated according to

$$x_i \leftarrow \frac{x_i e^{\eta a_{ij}}}{\sum_\ell x_\ell e^{\eta a_{\ell j}}}, \tag{23.1}$$

where a_{ij} denotes entry i of vector $a^{(j)}$, and the denominator works to enforce $\langle \mathbf{1}, x\rangle = 1$. By upweighting x in the direction of the violated constraint $a^{(j)}$, this update rule brings the x closer to satisfying the constraint. The magic of the multiplicative weights method is that the promise problem described above can be solved after only $O(\log(n)/\epsilon^2)$ iterations [58]. By searching for a violated constraint using a Grover search, the runtime of each iteration can be sped up quantumly, giving rise to polynomial speedups for solving zero-sum games and LPs more generally [46].

In the analogy to a panel of experts, we may view the above problem as follows. Each expert $i \in [n]$ produces a prediction for each of $j \in [m]$ data points, denoted by a_{ij}. We wish to produce a weighting x over the n experts such that the weighted majority of the n experts yields a positive value for all m data points. The multiplicative weights method solves this iteratively by be-

ginning with a uniform weighting over the n experts, and repeatedly observing an index j where the weighted majority misclassifies (i.e., predicts a negative value for) data point j, assessing a multiplicative penalty of $e^{\eta a_{ij}}$ to the weight of expert i. In a machine learning context, the weighting of experts produced by the algorithm can then be used as a classifier that allows us to predict a label for a new data point, by following the opinion of the weighted majority of the experts.

The *matrix* MWU method generalizes the n-dimensional vector x to an $n \times n$ symmetric matrix X. An example problem generalizing the above is

$$\begin{aligned} \langle A^{(j)}, X \rangle &\geq 0 \qquad j = 1, \ldots, m \\ \langle \mathbf{I}, X \rangle &= 1 \\ X &\geq 0 \,, \end{aligned}$$

where $A^{(j)}$ are fixed symmetric constraint matrices and the notation $\langle U, V \rangle := \mathrm{Tr}(UV)$ generalizes the dot product from vectors to matrices. Here $\mathbf{I}$ denotes the identity matrix, and $X \geq 0$ denotes that X is positive semidefinite. The problem above is related to the general form of an SDP, and the matrix MWU approach can be applied to solve SDPs. Note that we recover the vector example if we specify that the matrices $A^{(j)}$ and X are diagonal. The final two constraints indicate that X is a density matrix and is associated with a quantum state on $\log_2(n)$ qubits. When X is updated by a generalization of the rule in Eq. (23.1), then at every iteration of the MWU method, X will be a Gibbs state for a certain Hamiltonian that is a weighted sum of the symmetric constraint matrices $A^{(j)}$. Thus, the quantum state X can be prepared on a quantum computer using algorithms for Gibbs sampling. Taking this approach, quantum algorithms can achieve guaranteed polynomial speedups for performing an iteration of the MWU method compared to classical approaches, and it is conceivable that larger speedups could be available if the associated quantum systems admit faster-than-worst-case Gibbs sampling.

Dominant resource cost (gates/qubits)

The MWU method, both in the classical and quantum setting, consists of some number T of iterations, where each iteration updates a classical data structure. In typical applications, $T = \mathrm{poly}(\log(n)/\epsilon)$, where n is the problem size and ϵ is a precision parameter related to how close to optimal the solution has to be. This contrasts with other approaches to solving optimization problems, such as interior point methods, for which the number of iterations can scale as $O(\mathrm{poly}(n)\log(1/\epsilon))$.

Each iteration typically takes $\text{poly}(n, m, 1/\epsilon)$ time and is carried out with subroutines that can often be sped up with quantum algorithms. These subroutines can include Grover search / amplitude amplification and, in the case of the matrix MWU method, Gibbs sampling, which end up dominating the quantum cost of the algorithm.

Here it is important to point out that, especially in the quantum case, the MWU method can benefit from keeping an implicit representation of the n-dimensional vector x (or in the case of matrix MWU, the $n \times n$ matrix X). For instance, in the example problem above, we need not explicitly write down the vector x; rather, we can keep track of the indices $j_1, j_2, \ldots, j_t$ corresponding to the penalties assessed at iterations $1, 2, \ldots, t$. These indices can be organized into a t-sparse vector $y \in \mathbb{R}^m$ from which x is defined implicitly by $x_i \propto e^{\eta \sum_{j=1}^{m} a_{ij} y_j}$. In the context of optimization problems like LPs and SDPs, the vector y can often be related to the *dual* version of the optimization problem (see, e.g., [45]), where the goal is to find an optimal y, and each value y_j may be interpreted as the weight assigned to decision $j \in [m]$. Given y, the Gibbs sampling primitive can then produce a quantum state on $O(\log(n))$ qubits encoding the vector x. At iteration $t + 1$, this quantum state is used to find an index j_{t+1} corresponding to a violated constraint without ever explicitly writing down the vector x. This implicit representation is essential if the quantum algorithm is to achieve complexity sublinear in n. The same situation arises in algorithms for SDP based on matrix MWU, where Gibbs sampling is used to produce a $O(\log(n))$-qubit mixed quantum state $\rho = e^{-H}/\text{tr}(e^{-H})$, where on iteration $t+1$, the Hamiltonian $H = \sum_j y_j A^{(j)}$ is given by a weighted sum of at most t distinct input matrices. By keeping track only of the sparse vector $y \in [m]$, one avoids needing to write down the n^2 entries of ρ. In fact, ignoring dependence on ϵ, the Gibbs state ρ can typically be prepared using only $\widetilde{O}(s\sqrt{n})$ queries to the input data, where $s \leq n$ is the sparsity of the $n \times n$ input matrices $A^{(j)}$ (see, e.g., [48, 45]). This represents a polynomial speedup in the per-iteration cost compared to classical methods.

Additionally, there is also an appealing possibility that, for specific cases, the Gibbs sampling step for the $\log_2(n)$-qubit system could be accomplished in $\text{polylog}(n)$ time if the system thermalizes rapidly, allowing quantum algorithms based on the matrix MWU method to have faster runtime, perhaps as fast as $\text{poly}(\log(n), 1/\epsilon)$, representing an exponential speedup over their $\text{poly}(n, 1/\epsilon)$-time classical counterparts.

Caveats

One caveat is that the best outlook for quantum advantage occurs when the constraint matrices $A^{(j)}$ that appear in applications are sparse matrices (and

especially if they correspond to physical local Hamiltonians). However, this sparsity constraint may not be satisfied often in practice. There can in principle still be a speedup for dense matrices, but in this case, access to a large quantum random access memory might be required, which has its own caveats.

Another caveat to achieving a practically useful algorithm with either the classical or the quantum version of the MWU method is that the theoretical dependence of the runtime on the error parameter ϵ may lead to poor practical runtimes. The original quantum SDP solver based on MWU had $O(\epsilon^{-18})$ dependence [181], and this was later improved to $O(\epsilon^{-5})$ [45]. While this is technically poly$(1/\epsilon)$ scaling, the large power would likely lead the algorithm to be worse than alternatives, such as classical or quantum interior point methods which have polylog$(1/\epsilon)$ scaling, unless it is tolerable for ϵ to be essentially constant. In the case of zero-sum games, the quantum algorithm based on the MWU method has a slightly more tolerable $O(\epsilon^{-3})$ dependence.

Example use cases

- The MWU method can be used to gain an asymptotic quantum speedup in solving zero-sum games, and relatedly, solving LPs [46, 178]. This speedup is generated by Grover-like methods and does not require Gibbs sampling of quantum states. Many interesting optimization problems can be reduced to an LP.
- The MWU method was used in [686] to give an algorithm for online portfolio optimization, relevant to applications in finance.
- The matrix MWU method can be used to gain an asymptotic speedup for solving SDPs in the regime where the precision parameter ϵ to which the program should be optimized is large. Many interesting optimization problems can be reduced to an SDP. One notable example is that approximate solutions to (discrete) binary optimization problems can be found by solving the (continuous) SDP relaxation of the problem and performing a rounding procedure on the solution (see, e.g., [183, 71]).

Further reading

- See Arora, Hazan, and Kale [58] for an overview of the MWU method from a classical perspective, including its matrix generalization.
- The quantum algorithm for SDP based on the MWU method was introduced by Brandão and Svore [181]. This was improved in subsequent works [182, 48, 45]. The method was applied to the specific application of solving SDP relaxations of binary optimization problems in [183, 71], and to the specific application of computing optimal strategies of zero-sum games in [46].

- In [178], a "dynamic Gibbs sampling" method is proposed to improve the complexity of the MWU algorithm for zero-sum games. It would be interesting if this method can be extended to other applications of the MWU method.
- For an overview of multiplicative weights methods within quantum algorithms, see [520].