Chapter 2

The Experts/Multiplicative Weights Algorithm and Applications

We turn to the problem of *online learning*, and analyze a very powerful and versatile algorithm called the *multiplicative weights update* algorithm. Let's motivate the algorithm by considering a few diverse example problems; we'll then introduce the algorithm and later show how it can be applied to solve each of these problems.

2.1 Some optimization problems

Machine learning: Learning a linear classifier. In machine learning, a basic but classic setting consists of a set of k labeled examples $(a_1, l_1), ..., (a_k, l_k)$ where the $a_j \in \mathbb{R}^N$ are "feature vectors" and the l_j are labels in $\{-1, 1\}$. The problem is to find a *linear classifier*: a unit vector $x \in \mathbb{R}_+^N$ such that $\|x\|_1 = 1$ and $\forall j \in \{1, \ldots, k\}$, $l_j(a_j \cdot x) \geq 0$ (think of x as a distribution over the coordinates of the a_j, the features, that "explains", or correlates with, the labeling indicated by the l_j).

Machine learning: Boosting. Suppose given a sequence of training points $x_1, \ldots, x_N$ sampled from some universe according to some (unknown) distribution $\mathcal{D}$. Each point has an (unknown) label $c(x_i) \in \{-1, 1\}$, where the function c is taken from some restricted set of functions (the "concept class" $\mathcal{C}$). The goal is to find a hypothesis function $h \in \mathcal{C}$ that assigns labels to points, and predicts the function c in the best way possible (on average over $\mathcal{D}$). For instance, in the previous example the concept class is the class of all linear classifiers (or, "hyperplanes"), the distribution is uniform, and the hypothesis function is x.

Call a learning algorithm *strong* if it outputs with probability at least $1 - \delta$ a hypothesis h such that $\mathbf{E}_{i \sim \mathcal{D}} \frac{1}{2}|h(x_i) - c(x_i)\}| \leq \varepsilon$, and *weak* if the error is at most $\frac{1}{2} - \gamma$. Suppose the only thing we have is a weak learning algorithm: for any distribution $\mathcal{D}$, it returns a hypothesis h such that

$$\mathop{\mathbf{E}}_{i \sim \mathcal{D}} \frac{1}{2}\big|h(x_i) - c(x_i)\big| \leq \frac{1}{2} - \gamma.$$

The goal is to use the weak learner in order to construct a strong learner.

Approximation algorithms: Vertex Cover Given a universe $U = \{1, \dots, N\}$ and a collection $\mathcal{C} = \{C_1, \dots, C_m\}$ of subsets of U, the goal is to find the smallest possible number of sets from $\mathcal{C}$ that covers all of U.

Linear programming. Given a set of N linear constraints $a_i^T x \geq b_i$, where $a_i \in \mathbb{R}^m$ and $b_i \in \mathbb{R}$, the goal is to decide if there is an (approximately) feasible solution: does there exist an $x^* \in \mathbb{R}_+^m$ such that $a_i \cdot x^* \geq b_i - \delta$ for all $i \in \{1, \dots, N\}$?

Game theory: zero-sum games. Consider a game between two players, the "row" and "column" players. Each player can choose an action $i, j \in \{1, \dots, N\}$. If the actions are (i, j) the payoff to the row player is $M(i, j) \in \mathbb{R}$, and the payoff to the column player is $-M(i, j)$. A player can decide to randomize and choose a distribution over actions which she will sample from. In particular, if the row player plays according to a distribution $p \in \mathbb{R}_+^N$ over row actions and the column player plays according to a distribution $q \in \mathbb{R}_+^N$ over column actions, then the expected payoff of the row player is $p^T M q$ and the expected payoff of the column player is $-p^T M q$. The goal is to find (approximately) optimal strategies for both players: (p^*, q^*) such that

$$(p^*)^T M q^* \geq \lambda^* - \varepsilon, \qquad \lambda^* = \max_q \min_p p^T M q = \min_p \max_q p^T M q.$$

What do these five problems have in common? For each there is a natural greedy approach. In a zero-sum game, we could pick a random strategy to start with, find the second player's best response, then the first player's best response to this, etc. For the linear programming problem we can again choose a random x, find a constraint that is violated, update x. Same for the linear classifier problem (which can itself be written as a linear program). In the set cover problem we can pick a first set that covers the most possible elements, then a second set covering the highest possible number of uncovered elements, etc. In boosting we can tweak the distribution over points and call a weak learner to improve the performance of our current hypothesis with respect to misclassified points.

In each case it is not so easy to analyze such a greedy approach, as there is always a risk to get stuck in a " local minimum". In this lecture we're going to develop an abstract "greedy-but-cautious" algorithm that can be applied to efficiently solve each of these problems.

2.2 Online learning

Consider the following situation. There are T steps, $t = 1, \dots, T$. At each step t,

- The player chooses an action $x^{(t)} \in \mathcal{K} \subseteq \mathbb{R}^N$.
- She suffers a loss $f^{(t)}(x^{(t)})$, where $f^{(t)} : \mathcal{K} \to [-1, 1]$ is the *loss function.*

The player's goal is to devise a strategy for choosing her actions $x^{(1)}, \dots, x^{(T)}$ which minimizes the *regret*,

$$R_T = \sum_{t=1}^{T} f^{(t)}(x^{(t)}) - \min_{x \in \mathcal{K}} \sum_{t=1}^{T} f^{(t)}(x). \tag{2.1}$$

The regret measures how much worse the player is doing compared to a player who is told all loss functions in advance, but is restricted to play the same action at each step. Since the player may be adaptive a piori regret could be positive or negative. In virtually all cases the set $\mathcal{K}$ as well as the functions $f^{(t)}$ will be convex, so that setting $x = \frac{1}{T}(x^{(1)} + \cdots + x^{(T)})$ we can see that under this convexity assumption regret is always a non-negative quantity.

Exercise 1. What if regret were defined, not with respect to the best *fixed* action in hindsight, but by comparison with an offline player who is allowed to switch between experts at every step? Construct an example where the loss suffered by such an offline player is 0, but the expected loss of *any* online learning algorithm is at least $T \cdot (1 - 1/N)$.

In online learning we can distinguish between the cases where the player is told what the function $f^{(t)}$ is, or only what her specific loss $f^{(t)}(x^{(t)})$ is. The former scenario is called the "full information model", and it is the model we'll focus on. The latter is usually referred to as the "multi-armed bandit" problem, but we won't discuss it further. In addition we are going to make the following assumptions:

- The set of allowed actions for the player is the probability simplex $\mathcal{K}_N = \{p \in \mathbb{R}_+^N, \sum_i p_i = 1\}$,
- Loss functions are *linear*, $f^{(t)}(p) = m^{(t)} \cdot p$ where $m^{(t)} \in \mathbb{R}^N$ is such that $|m_i^{(t)}| \le 1$ for all $i \in \{1, \dots, N\}$. Note that this ensures that $f^{(t)}(p) \in [-1, 1]$ for any $p \in \mathcal{K}_N$.

This setting has an interpretation in terms of "experts": think of each of the coordinates $i = 1, \dots, N$ as an expert. At each time t, the experts make recommendations. The player has to bet on an expert, or more generally on a distribution over experts. Then she gets to see how well the expert's recommendations turned out: to each expert i is associated a loss $m_i^{(t)}$, and the player suffers an average loss $\sum_i p_i^{(t)} m_i^{(t)}$. The player's goal is to minimize her regret R_T, which in this case boils down to

$$\begin{aligned} R_T &= \sum_{t=1}^{T} f^{(t)}(p^{(t)}) - \min_{p \in \mathcal{K}_N} \sum_{t=1}^{T} f^{(t)}(p) \\ &= \sum_{t=1}^{T} m^{(t)} \cdot p^{(t)} - \min_{i \in \{1,\dots,N\}} \sum_{t=1}^{T} m_i^{(t)}, \end{aligned}$$

i.e. she is comparing her loss to that of the best expert in hindsight.

How would you choose which expert to follow so as to minimize your total regret? A natural strategy is to always choose the expert who has performed best so far. Suppose for simplicity that experts are always either right or wrong, so that the losses $m_i^{(t)} \in \{0, 1\}$: 0

for right and 1 for wrong. Suppose also that there is at least one expert who always gets it right. Finally suppose at each step we "hedge" and choose an expert at random among those who have always been correct so far. Then whenever we suffer a loss $r_t \in [0,1]$ it precisely means that a fraction r_t of the remaining experts (those who were right so far) got it wrong and are thus eliminated. Since $N \prod_t (1-r_t) \geq 1$ (there are N experts to start with and there must be at least 1 remaining), taking logarithms we see $\sum_t r_t \leq \ln N$, which is not bad; in particular it doesn't grow with T.

But now let's remove our assumption that there is an expert that always gets it right — in practice, no one is perfect. Suppose at the first step all experts do equally well, so $m^{(1)} = (1/2, \dots, 1/2)$. But now experts get it right or wrong on alternate days, $m^{(2t)} = (1, \dots, 1, 0, \dots, 0)$ and $m^{(2t+1)} = (0, \dots, 0, 1, \dots, 1)$ for $t \geq 1$. In this case, it turns out we're always going to pick the wrong expert! Our regret increases by 1/2 per iteration on average, and we never "learn". In this case a much better strategy would be to realize that no expert is consistently good, so that by picking a random expert at each step we suffer an expected loss of 1/2, which is just as good as the best expert — our regret would be 0 instead of $\sim T/2$.

Follow the Regularized Leader. The solution is to consider a strategy which does not jump around too quickly. Instead of selecting the absolute best expert at every step, we're going to add a "risk penalty" which penalizes distributions that concentrate too much on a small number of experts — we'll only allow such high concentration if there is really a high confidence that the experts are doing much better than the others. Specifically, instead of playing the distribution p such that

$$p^{(t+1)} = \arg\min_{p \in \mathcal{K}_N} \sum_{j \leq t} m^{(j)} \cdot p,$$

we'll choose

$$p^{(t+1)} = \arg\min_{p \in \mathcal{K}_N} \Big(\eta \sum_{j \leq t} m^{(j)} \cdot p + R(p) \Big), \tag{2.2}$$

where R is a "regularizer" and $\eta > 0$ a small weight of our choosing used to balance the two terms.

2.3 The Multiplicative Weights Update algorithm

The multiplicative weights update (MWU for short) algorithm is a special case of follow the regularized leader where the regularizer R is chosen to be the (negative of the) entropy function.

Regularizing via entropy. Let $R(p) = -H(p)$, where $H(p) = -\sum_i p_i \ln p_i$ is the *entropy* of p. (In this lecture we'll use the unusual convention that logarithms are measured in "nats". This is only for convenience, and in later lectures we'll revert to the binary logarithm log.) We'll use the following properties of entropy.

Lemma 2.1. *Let p, q be distributions in $\mathcal{K}_N$, $H(p) = \sum_i p_i \ln \frac{1}{p_i}$ the entropy of p and $D(p\|q) = \sum_i p_i \ln \frac{p_i}{q_i}$ the relative entropy (sometimes also called KL divergence) between p and q. Then the following hold:*

(1) For all p, $0 \leq H(p) \leq \ln N$.

(2) For all p and q, $D(p\|q) \geq 0$.

Exercise 2. Prove the lemma.

The only way to have zero entropy is to have $\ln p_i = 0$ for each i such that $p_i \neq 0$, i.e. $p_i = 1$ for some i and zero elsewhere: low-entropy distributions are highly concentrated. At the opposite extreme, the only distribution with maximal entropy is the uniform distribution (as being uniform is the only way to saturate Jensen's inequality used in the proof of the lemma). Thus the effect of using the negative entropy as a regularizer is to penalize distributions that are too highly concentrated.

The multiplicative weights update rule. With this choice of regularizer it is possible to solve for (2.2) explicitly. Specifically, note that the gradient at point q is simply $\nabla q_i = \eta \sum_{j\leq t} m_i^{(j)} + \ln(q_i) + 1$, and setting this to zero gives $q_i = e^{-1-\eta \sum_{j\leq t} m_i^{(j)}}$. This gives us the unique minimizer over $\mathbb{R}^n$. To take into account the normalization to a probability distribution, we project on $\mathcal{K}_N$ and find the update rule

$$\begin{aligned} p_i^{(t+1)} &= \frac{e^{-\eta \sum_{j\leq t} m_i^{(j)}}}{\sum_i e^{-\eta \sum_{j\leq t} m_i^{(j)}}} \\ &= \frac{e^{-\eta m_i^{(t)}} p_i^{(t)}}{\sum_i e^{-\eta m_i^{(t)}} p_i^{(t)}}. \end{aligned}$$

You can check using the KKT conditions that this solution indeed minimizes (2.2) over $\mathcal{K}_N$, when $R(p) = -H(p)$. This suggests the following *multiplicative weights update algorithm*:

```
Algorithm 1: Multiplicative Weights Algorithm (a.k.a MW or MWA)
Choose $\eta \in (0,1)$;
Initialize weights $w_i^{(1)}$ for each $i \in \{1,\dots,N\}$ ;
(example: $w_i^{(1)} = 1$, but they can be chosen arbitrarily)
for $t = 1,2,\dots,T$ do
    Choose decisions proportional to the weights $w_i^{(t)}$, i.e., use the distribution
        $p^{(t)} = \left(\frac{w_1^{(t)}}{\Phi^{(t)}},\dots,\frac{w_N^{(t)}}{\Phi^{(t)}}\right)^T$ ,
     where $\Phi^{(t)} = \sum_i w_i^{(t)}$;
    Observe the costs $m^{(t)}$ of the decisions;
    Penalize costly decisions by updating their weights: $w_i^{(t+1)} \leftarrow e^{-\eta m_i^{(t)}} w_i^{(t)}$;
end
```

At every step we update our weights $w^{(t)} \leftarrow w^{(t+1)}$ by re-weighting expert i by a multiplicative factor $e^{-\eta m_i^{(t)}}$ that is directly related to its performance in the previous step. However, we do so smoothly, a small step at a time; how small the step is governed by η, a parameter that we'll choose for best performance later.

Remark 2.2. Note that for small η, $e^{-\eta m_i^{(t)}} \approx 1 - \eta m_i^{(t)}$. So we could also use a modified update rule, $w_i^{(t+1)} = (1-\eta m_i^{(t)}) w_i^{(t)}$, directly in the algorithm; it sometimes lead to a slightly better bound (depending on the structure of the loss functions).

Analysis. The following theorem quantifies the performance of the Multiplicative Weights Algorithm.

Theorem 2.3. *For all $N \in \mathbb{N}$, $0 < \eta < 1$, $T \in \mathbb{N}$, losses $m_i^{(t)} \in [-1,1]$ for $i \in \{1,\dots,N\}$ and $t \in \{1,\dots,T\}$, the above procedure starting at some $p^{(1)} \in \mathcal{K}_N$ suffers a total loss such that*

$$\sum_{t=1}^{T} m^{(t)} \cdot p^{(t)} \leq \min_{p \in \Delta_N} \left(\sum_{t=1}^{T} m^{(t)} \cdot p + \frac{1}{\eta} D\big(p \| p^{(1)}\big) \right) + \eta \sum_{t=1}^{T} \sum_{i=1}^{N} \left(m_i^{(t)}\right)^2 p_i^{(t)}$$

where $D(p\|q) = \sum_i p_i \ln(p_i/q_i)$ is the relative entropy.

Before proving the theorem we can make a number of important comments. First let's make some observations that will give us a simpler formulation. By choosing $p^{(1)}$ to be the uniform distribution we can ensure that $D(p\|p^{(1)}) = \ln N - H(p)$ is always at most $\ln N$, whatever the optimal p is. Moreover, using $|m_i^{(t)}| \leq 1$ the last term is at most ηT. Using the definition of the regret (2.1), we get

$$R_T \leq \frac{1}{\eta} \ln N + \eta T.$$

The best choice of η in this equation is $\eta = \sqrt{\ln N/T}$, in which case we get the bound

$$\frac{1}{T} R_T \leq 2\sqrt{\frac{\ln N}{T}}.$$

What this means is that, if we want to have an average regret, over the T rounds, that is at most ε, then it will suffice to run the procedure for a number of iterations $T = 4 \ln N/\varepsilon^2$. The most remarkable feature of this bound is the logarithmic dependence on N: in only a logarithmic number of iterations we are able to narrow down on a way to select experts that ensures our average loss is small. This matches the guarantee of our earlier naive procedure of choosing the best expert so far, except that now it works in *all* cases, whether the experts are consistent or not! The dependence on ε, however, is not so good. This is an important drawback of the MWA; in many cases it is not a serious limitation but it is important to have it in mind, and we'll return to this issue when we look at some examples. First let's prove the theorem.

Proof of Theorem 2.3. The proof is based on the use of the relative entropy as a potential function: for any q we measure the decrease

$$\begin{aligned} D(q\|p^{(t+1)}) - D(q\|p^{(t)}) &= \sum_i q_i \ln \frac{p_i^{(t)}}{p_i^{(t+1)}} \\ &= \eta \sum_i q_i m_i^{(t)} + \Big(\sum_i q_i\Big) \ln \Big(\sum_i e^{-\eta m_i^{(t)}} p_i^{(t)}\Big) \\ &\leq \eta m^{(t)} \cdot q - \eta m^{(t)} \cdot p^{(t)} + \eta^2 \sum_i \big(m_i^{(t)}\big)^2 p_i^{(t)}, \end{aligned}$$

where for the last line we used $e^{-x} \leq 1 - x + x^2$ and $\ln(1-x) \leq -x$ for all $|x| \leq 1$. Summing over t and using that the relative entropy is always positive proves the theorem. $\square$

2.4 Applications

We now give three applications of the multiplicative weights algorithm: to finding linear classifiers, solving zero-sum games, and solving linear programs. You'll treat other applications in your homework.

2.4.1 Finding linear classifiers

Let's get back to our first example, learning a linear classifier. For simplicity, we are going to assume that there exists a strict classifier, in the sense that there exists $x^* \in \mathbb{R}^N$ such that

$$\|x^*\|_1 = 1 \qquad \text{and} \qquad \forall j \in \{1, \ldots, k\}, \quad l_j(a_j \cdot x^*) \geq \varepsilon \tag{2.3}$$

for some $\varepsilon > 0$ (here $\|a\|_1 = \sum_i |a_i|$).

First we observe that we may also assume that $x^* \in \mathbb{R}_+^N$. For this, create a new input by setting $b_j = \begin{pmatrix} a_j \\ -a_j \end{pmatrix} \in \mathbb{R}^{2N}$. Suppose there exists $x^* \in \mathbb{R}^N$ such that (2.3) holds. Let $x_+^* = \max(x^*, 0)$ and $x_-^* = \min(x^*, 0)$ (component-wise), so $x^* = x_+^* + x_-^*$. Then $y^* = \begin{pmatrix} x_+^* \\ -x_-^* \end{pmatrix}$ has non-negative entries summing to 1, and satisfies (2.3) with the b_j replacing the a_j. Conversely, for any $y^* = \begin{pmatrix} y_1^* \\ y_2^* \end{pmatrix} \in \mathbb{R}_+^{2N}$ with entries summing to 1 such that (2.3) holds (for the b_j), we can define $x^* = \frac{1}{2}(y_1^* - y_2^*)$, so $\|x^*\|_1 \leq 1$ and x^* satisfies (2.3), with the a_j and ε replaced by $\frac{1}{2}\varepsilon$.

Thus the two problems are equivalent, and for the remainder of this section we assume that (2.3) is satisfied for some $x^* \in \mathbb{R}_+^N$ such that $\sum_i x_i^* = 1$ (we keep using the notation a_j). We also define $\rho = \max_j \|a_j\|_\infty$, where $\|a\|_\infty = \max_{i=1,\ldots,N} |a_i|$.

The idea behind applying MWA here is to consider each of the N coordinates of vectors a_j as an expert, and choose the loss function $m^{(t)}$ as a function of one of the a_j that is not well classified by the current vector $p^{(t)}$. If $p^{(t)}$ is such that all of the a_j's satisfy $l_j(a_j \cdot p^{(t)}) \geq 0$, then all points are well classified by $x = p^{(t)}$ and we can stop the algorithm. Otherwise, we update $p^{(t)}$ through the MWA update rule and go to the next step. The question here is to determine why the algorithm terminates, and in how many steps.

Let us analyze the MW approach in a bit more details. We initialize $w_i^{(1)} = 1$ for all i, and $p^{(t)}$ accordingly. At each round t, we look for a value j such that $l_j(a_j \cdot p^{(t)}) < 0$, i.e. a_j is not classified correctly. If there is none, the algorithm stops. If there is one, pick this j and define $m^{(t)} = -\frac{l_j}{\rho} a_j$; note that by definition of ρ, $m_i^{(t)} \in [-1, 1]$ for all i.

By assumption, there exists x^* such that

$$m^{(t)} \cdot x^* = -l_j(a_j \cdot x^*)/\rho \leq -\varepsilon/\rho.$$

By the main MW theorem, since x^* is a distribution, we have

$$\sum_{t=1}^{T} m^{(t)} \cdot p^{(t)} \leq \sum_{t=1}^{T} m^{(t)} \cdot x^* + \frac{\ln N}{\eta} + \eta T.$$

Using $m^{(t)} \cdot x^* \leq -\varepsilon/\rho$, we get

$$\sum_{t=1}^{T} m^{(t)} \cdot p^{(t)} \leq -\frac{\varepsilon}{\rho} T + \frac{\ln N}{\eta} + \eta T.$$

Now, one should remark that until the last time step T, i.e. until the algorithm stops, we chose a_j such that $l_j(a_j \cdot p^{(t)}) < 0$, which implies that $\forall t = 1, \ldots, T$, $m^{(t)} \cdot p^{(t)} = -l_j(a_j \cdot p^{(t)})/\rho > 0$. It immediately follows that

$$0 < -\frac{\varepsilon}{\rho} T + \frac{\ln N}{\eta} + \eta T.$$

Finally, choosing $\eta = \frac{\varepsilon}{2\rho}$, we obtain $T < \frac{4\rho^2 \ln N}{\varepsilon^2}$. This proves that the algorithm terminates, and that it finds a classifier in less than $\frac{4\rho^2 \ln N}{\varepsilon^2}$ timesteps.

2.4.2 Zero-sum games

Next we consider games in the game theoretic sense: informally, a set of k players each have a set of a actions they can choose from; every player chooses an action, after which he gets a payoff or utility that is a function of his own action and the actions of other players. Here will limit ourselves to 2-player zero-sum games, in which each of the two players has a finite number N of available actions. The payoffs of players 1 and 2 (let us call them the row and column players from now on) can be represented by a single payoff matrix $M \in \mathbb{R}^{N \times N}$ such if the row player plays action i, and the column player plays j, then the column player receives payoff $M(i,j)$ and the row player receives $-M(i,j)$.

A player can decide to randomize and choose a distribution over actions which she will sample from. In particular, if the row player plays according to a distribution p over row actions (instead of playing one specific action) and the column player plays according to a distribution q over column actions, then the expected payoff of the column player is $p^T M q$ and the expected payoff of the row player is $-p^T M q$.

Given that each player is trying to maximize her utility, we can see that the row and column players in a zero-sum games have conflicting objectives: the column player wants to maximize $p^T M q$, while the row player want to minimize the same quantity. One nice property of zero-sum games is that it does not matter for any of the players whether they act first: Von Neumann's minimax theorem states that

$$\max_q \min_p p^T M q = \min_p \max_q p^T M q = \lambda^*.$$

This means in particular that it does not matter whether the row player tries to minimize $p^T M q$ first and lets the column player act second and maximize $\min_p p^T M q$ or the column player tries to maximize $p^T M q$ first and the row player then minimizes $\max_q p^T M q$: the utility both players end up getting does not change. This "optimal" utility λ^* is called the value of the game.

We want to show that the MWA allows us to find strategies that are near-optimal for a zero-sum game, in the sense that for any $\varepsilon > 0$ and for a sufficient number of time steps T, the MWA finds a set of strategies $(\frac{1}{T}\sum_{t=1}^T p^{(t)}, \frac{1}{T}\sum_{t=1}^T q^{(t)})$ for the row and column players that satisfy

$$\lambda^* \leq \left(\frac{1}{T}\sum_{t=1}^T p^{(t)}\right)^T M \left(\frac{1}{T}\sum_{t=1}^T q^{(t)}\right) \leq \lambda^* + \varepsilon.$$

The idea behind the MW here is to take $p^{(t)}$ to be the strategy of the row player. At each time step, $q^{(t)}$ is chosen as the best response to strategy $p^{(t)}$ (i.e. the strategy which maximize the utility of the column player given that the row player plays $p^{(t)}$); then, the row player uses the multiplicative weight update based on a loss vector $m^{(t)}$ chosen as the vector of expected payoffs for the row player when the column player plays $q^{(t)}$ to decide what $p^{(t+1)}$ will be in the next time step. Intuitively, the row player tries to penalize the strategies that lead to a low payoff and to put more weight on the strategy that leads to a higher payoff. Formally, the algorithm proceeds as follows:

Algorithm 2: MWA for zero-sum games

```
Choose $\eta \in (0,1)$ ;
Initialize weights $w_i^{(1)} := 1$ for each $i \in \{1, \dots, N\}$;
for $t = 1, 2, \dots, T$ do
    Define the distribution $p^{(t)} = \{\frac{w_1^{(t)}}{\Phi^{(t)}}, \dots, \frac{w_N^{(t)}}{\Phi^{(t)}}\}$, where $\Phi^{(t)} = \sum_i w_i^{(t)}$;
    Define $q^{(t)} = \text{argmax}_q (p^{(t)})^T Mq$ and $m^{(t)} = Mq^{(t)}$;
    Penalize costly decisions by updating their weights: $w_i^{(t+1)} \leftarrow e^{-\eta m_i^{(t)}} w_i^{(t)}$;
end
```

Let's analyze this algorithm. Assume that the payoffs $M(i,j) \in [-1,1]$. Choosing $\eta = \sqrt{\ln N/T}$ and $T = 4 \ln N/\varepsilon^2$, according to the MW theorem, for any distribution p^*,

$$\frac{1}{T}\sum_{t=1}^{T} m^{(t)} \cdot p^{(t)} = \frac{1}{T}\sum_{t=1}^{T}(p^{(t)})^T Mq^{(t)} \leq \frac{1}{T}\sum_{t=1}^{T}(p^*)^T Mq^{(t)} + \varepsilon \ . \tag{2.4}$$

Take $p^* = \text{argmin}_p\, p^T Mq^{(t)}$. Then for any t,

$$(p^*)^T Mq^{(t)} \leq \min_p p^T Mq^{(t)} \leq \max_q \min_p p^T Mq \ = \ \lambda^* \ ,$$

and therefore (2.4) implies

$$\frac{1}{T}\sum_{t=1}^{T}(p^{(t)})^T Mq^{(t)} \leq \lambda^* + \varepsilon \ .$$

To lower bound $\frac{1}{T}\sum_{t=1}^{T}(p^{(t)})^T Mq^{(t)}$, note that for every distribution q,

$$(p^{(t)})^T Mq^{(t)} \geq (p^{(t)})^T Mq$$

given the choice of $q^{(t)}$ made in the algorithm; therefore,

$$(p^{(t)})^T Mq^{(t)} \geq \max_q (p^{(t)})^T Mq$$

and

$$(p^{(t)})^T Mq^{(t)} \geq \min_p \max_q p^T Mq = \lambda^*.$$

Using the fact that the above bounds are valid for any t, by linearity we get that the distributions produced by the algorithm satisfy

$$\lambda^* \leq \left(\frac{1}{T}\sum_{t=1}^{T} p^{(t)}\right)^T M \left(\frac{1}{T}\sum_{t=1}^{T} q^{(t)}\right) \leq \lambda^* + \varepsilon.$$

This equation shows that p^* and q^* are approximate equilibria of the game, in the sense that neither the row or the column player can improve her utility by more than ε by changing her strategy, when the strategy of the other player remains fixed. Indeed, imagine that the column player decides to change her strategy to another q. Then, she is going to get payoff

$$\Big(\frac{1}{T}\sum_{t=1}^{T} p^{(t)}\Big)^T Mq \leq \frac{1}{T}\sum_{t=1}^{T} (p^{(t)})^T Mq^{(t)} \leq \lambda^* + \varepsilon.$$

However, she is guaranteed to earn payoff at least λ^* when playing q^*, and can therefore not improve her payoff by more than ε by changing her strategy. A similar reasoning applies to the row player.

2.4.3 Solving linear programs

Our goal is to check to feasibility of a set of linear inequalities,

$$Ax \geq b, \quad x \geq 0,$$

where $A = [a_1 ... a_m]^T$ is an $m \times n$ matrix and x an n-dimensional vector, or more precisely to find an approximately feasible solution $x^* \geq 0$ such that for some $\varepsilon > 0$,

$$a_i^T x^* \geq b_i - \varepsilon, \qquad \forall i.$$

The analysis will be based on an oracle that answers the following question: Given a vector c and a scalar d, does there exist an $x \geq 0$ such that $c^T x \geq d$? With this oracle, we will be able to repeatedly check whether a convex combination of the initial linear inequalities, $a_i^T x \geq b_i$, is infeasible; a condition that is sufficient for the infeasibility of our original problem. Note that the oracle is straightforward to construct, as it involves a single inequality. In particular, it returns a negative answer if $d > 0$ and $c < 0$.

The algorithm is as follows. Experts correspond to each of the m constraints, and loss functions are associated with points $x \geq 0$. The loss suffered by the i-th expert will be $m_i = \frac{1}{\rho}(a_i^T x - b_i)$, where $\rho > 0$ is a parameter used to ensure that the losses $m_i \in [-1, 1]$. (Although one might expect the penalty to be the violation of the constraint, it is exactly the opposite; the reason is that the algorithm is trying to actually prove infeasibility of the problem.) In the t-th round, we use our distribution $p^{(t)}$ over experts to generate an inequality that would be valid, if the problem were feasible: the inequality is

$$\sum_i p_i^{(t)} a_i^T x \geq \sum_i p_i^{(t)} b_i. \tag{2.5}$$

We then use the oracle to detect whether this constraint is infeasible, in which case the original problem is infeasible, or return a point $x^{(t)} \geq 0$ that satisfies the inequality. The loss we suffer is equal to $\frac{1}{\rho}\sum_i p_i^{(t)}(a_i^T x^{(t)} - b_i)$, and we use this loss to update our weights. Note that in case infeasibility is not detected, the penalty we pay is always nonnegative, since $x^{(t)}$ satisfies the inequality (2.5).

The simplified form of the multiplicative weights theorem gives us the following guarantee:

$$\sum_{t=1}^{T} m^{(t)} \cdot p^{(t)} \leq \sum_{t=1}^{T} m_i^{(t)} \cdot p_i^{(t)} + 2\rho\sqrt{\frac{\ln m}{T}},$$

for any i. Choose $T = 4\rho^2 \ln(m)/\varepsilon^2$. Using that the left-hand side is non-negative, if we set

$$x^* = \frac{1}{T}\sum_{t=1}^{T} x^{(t)}$$

we see that for every i, $a_i^T x^* \geq b_i - \varepsilon$, as desired.

The number of iterations required to decide approximate feasibility, T, scales logarithmically with the number of constraints m. This is much better than "standard" algorithms such as the ellipsoid algorithm, which are polynomial in n. However, the dependence on the accuracy ε is quadratic, which is much worse than these algorithms, for which the dependence is in $\log(1/\varepsilon)$. Finally the dependence of T on ρ^{-1} is also quadratic. The value of ρ depends on the oracle (it is called the "width" of the oracle), and depending on the specific LP we are trying to solve it may or may not be possible to design an oracle that guarantees a small value of ρ.