Online optimization algorithms

David Andrzejewski
CS 880 - Final Project

May 16, 2007

1 Introduction

This project surveys some recent algorithms and results in the field of online learning theory [5][6][7]. The problem context is that of an online decision problem, where one must choose a point x_t from a fixed set K for each round t. After a choice has been made, a cost function $f_t(x)$ for that round is revealed and used to compute the cost of that decision. On the next round $t+1$, a new point x_{t+1} must be chosen, a new cost function $f_{t+1}(x)$ will be revealed, and so on. The goal then is to minimize the *regret* R, the difference between the cumulative cost of the single best point chosen in hindsight x^* and the cumulative cost of the sequence of points chosen by the online procedure $x_{1...T}$.

$$R = \sum_{t=1}^{T} f_t(x_t) - \min_{x^* \in K} \sum_{t=1}^{T} f_t(x^*) \tag{1}$$

This formulation is quite general, and can be used to model a wide variety of real-world problems and applications, such as industrial production. In particular, many papers mention, and several investigate in-depth [3], the connection between the online decision problem and the universal portfolio algorithm [4], which deals with setting stock proportions in a financial portfolio. Within the field of theoretical computer science, online learning also has strong connections to auction and game theory [2]. Finally, many practical applications of machine learning techniques would be most naturally handled in an online fashion. This has led to the recent research into online versions of well-studied machine learning techniques, such as kernel methods [9].

Two of these papers [5][7] deal with the case of *online convex optimization*, where the feasible set of points K is convex, and each cost function $f_t(x)$ is also convex. That is, for all $\{x, y\} \in K,\ \lambda \in [0, 1]$ and $f_t(x)$ we have:

$$\lambda x + (1-\lambda)y \quad \in \quad K \tag{2}$$

$$f(\lambda x + (1-\lambda)y) \quad \leq \quad \lambda f(x) + (1-\lambda) f(y) \tag{3}$$

The remaining paper [6] ([Kalai]) does not assume convexity, and also examines the interesting modification where there is a cost associated with changing

the previous round's decision. It also does not make use of any gradient information, unlike the techniques discussed in both [5] and [7].

The outline of this survey is as follows. First, we will briefly present the major algorithms along with their contextual assumptions and regret bounds. For each algorithm we will briefly describe the general strategy or motivation behind it, and then outline the analysis and regret bound proof, highlighting key steps.

Next, we examine the commonalities between the different approaches. In particular, many of the approaches can be interpreted as variants of a "follow the leader" stategy. The differences between these approaches reflect uses of additional information (gradient, curvature), or randomization to avoid "worst-case" cost function sequences. Studying these differences can help shed light on how exactly these algorithms work.

2 Gradient Descent [5]

This algorithm deals with the setting of online convex optimization, as described in the introduction. Each round t, the algorithm must select a point from a bounded feasible set K, after which a convex cost function f_t will be revealed. It should be emphasized that this sequence of cost functions can be totally arbitrary, with no relationship between one and the next.

There are some assumptions required. The diameter of K is be bounded by some $D \in \Re$, such that $||x - y||_w \leq D \ \forall x, y \in K$. Furthermore the algorithm requires the gradient $\nabla f_t(x)$ for each revealed cost function f_t, and it requires that these length of these gradients be bounded by some G such that $||\nabla f_t(x)||_w \leq G$. Finally, it requires a projection operator Π_K capable of projecting any given x into the nearest point in K.

The algorithm itself is quite simple. For the very first point x_1, select any arbitrary point in K. Then for every round afterwards, select the next point according to a simple gradient descent rule followed by a projection back into the convex set K.

$$x_{t+1} = \Pi_K(x_t - \eta_t \nabla f_t(x_t)) \tag{4}$$

The additive regret over T rounds of play for this algorithm is $O(\sqrt{T})$, with step sizes chosen as $\eta_t = 1/\sqrt{t}$.

The particularly interesting aspect of the analysis is that the regret is bounded by replacing each cost function f_t with the hyperplane tangent to f_t at x_t defined by the gradient $\nabla f_t(x_t)$. That is, for each t define $g_t = \nabla f_t(x_t)$. Then replace the cost function f_t with $\widetilde{f_t}(x) = g_t \bullet x$. This linearization preserves the gradient, meaning that the algorithm will still select the same points. By the convexity of f_t, we have that

$$f_t(x) \geq \nabla f_t(x_t)(x - x_t) + f_t(x_t) \tag{5}$$

where x^* is the optimal offline point. Using this expression for both x_t and x^* allows us to bound to upper bound the true regret with the regret of our linearized cost functions.

$$\begin{aligned} f_t(x_t) - f_t(x^*) &\leq f_t(x_t) - (g_t(x^* - x_t) + f_t(x_t)) \quad (6) \\ &\leq g_t \bullet x_t - g_t \bullet x^* \quad (7) \end{aligned}$$

This key trick is then combined with the fact that $||\Pi_K(x) - x^*||_w \leq ||x - x^*||_w$ in order to derive the bound.

This algorithm can also be interpreted as an instance of "follow the leader" for a suitably defined sequence of cost functions [7], which will be discussed later.

2.1 Experts problem, online linear programming [5]

This paper then describes an alternative formulation of the original problem as a series of repeated games. Here the utilities (costs) depend on the actions of another player, which takes the place of the sequence of cost functions. The property of universal consistency is proven for the gradient ascent (descent) algorithm in this setting.

But what is more interesting to us is the definition of several new problems, including the *experts problem*. Here the algorithm must select a probability distribution over n experts, which is equivalent to a vector on the boundary of an n dimensional simplex. Each round the algorithm then gets a cost vector, which is equivalent to a linear cost function f_t.

A related problem is *online linear programming*, where the feasible set is some convex polytope K and a series of cost vectors are presented to the algorithm, which must choose points $x_t \in K$ each round.

The interesting connection is that any experts problem can be formulated as an online linear program, by associating an expert with each vertex of a polytope. Note that the offline solution to the linear program uses the sum of all cost vectors, which is itself a cost vector. Therefore the optimal solution to the offline linear program must then be one of the vertices, which is equivalent to saying the optimal offline experts solution must be to choose a single expert, which is an interesting connection.

Furthermore, the paper mentions the fact that some expert algorithms begin with an uneven weighing over the experts. In the online linear program, this may actually be helpful, as some vertices of the polytope may be nearer to one another, spatially clustered. An uneven weighing over the experts (each associated with a vertex) may then actually allow the online linear program to start from a distribution which is more spatially uniform, allowing it to approach the true offline optimal vertex more quickly.

3 Follow Perturbed Leader [6]

This problem assumes a framework similar to the experts advice problem described earlier. Here the algorithm must choose from among n experts each round, after which a cost function will be revealed. For purposes of analysis, the authors use a linear generalization, where each choice corresponds to a decision vector d, and each cost corresponds to a cost vector c. The true experts problem can easily be mapped into this formulation by saying that choosing expert 1 means setting $d_1 = 1$ and $d_i = 0$ for all $i \neq 1$, etc.

For our linear generalization, we again bound the diameter of our decision space by D. We also bound the maximum cost $|d \bullet c|$ above by C, and the length of our cost vector by A. All bounds here are in terms of the ℓ_1 norm. This setting is not exactly the same as the previous one (no convexity assumptions) but there are still some interesting connections to be observed.

The Follow Perturbed Leader (FPL) algorithm is quite simple. A natural choice for the online experts problem is to simply choose the expert who had done best so far for the cost functions you have observed (the leader). However, this approach is vulnerable to worst-case sequences of cost functions chosen such that the leader changes constantly, and at each step the current leader does very badly. To avoid this pitfall, FPL randomly perturbs each experts' running total score. Specifically, it selects a vector uniformly from the cube $[0, \frac{1}{\epsilon}]^n$, adding each component of the vector to the score of the corresponding expert.

The interesting aspect of this modification is the tradeoff it induces. The perturbation can be seen as making FPL less "data-driven", and thus less susceptible to misleading sequences of cost functions which will lead to large regret. However, the fact that the algorithm is made less data-driven will also hamper its ability to react to the observed sequence of cost functions. The expected additive regret bound for this algorithm with perturbation parameter ϵ then is given by

$$E[FPL(\epsilon)] \leq MIN + \epsilon CTA + \frac{D}{\epsilon} \tag{8}$$

If all of these aspects of the problem are known ahead of time, the perturbation parameter ϵ can be chosen to minimize expected regret by simpling solving for the minimum of the right-hand side. In this case, knowing these things about the problem allows us to set ϵ to optimize this tradeoff. If these parameters are not known, ϵ can be gradually decreased as the rounds go on (known as ϵ-havling tricks). The ϵT term in the expected regret bound shows why we would wish to shrink ϵ as the rounds go on, if T is not known beforehand.

4 Follow Expected Leader [6]

An interesting modification to FPL is for the case of non-discrete feasible sets. While FPL had to do with choosing among n experts every round, Follow Expected Leader (FEL) transfers the same ideas into the continuous domain of

convex sets. This makes the setting extremely similar to [5] (although here no gradient information is used, nor are any bounds on the gradient assumed).

The modification is, on any given step, to try a series of perturbations independently, and take the average of the best points chosen from each combination of the cost function history and a perturbation. This requires a convex set of feasible points in order to guarantee that these averaged points are themselves feasible. Linearity of expectation means that the expected regret bound for this algorithm is the same as for FPL. The FPL bound is linear in T, which is not as good as the gradient descent algorithm, but makes fewer assumptions.

5 Follow Lazy Leader [6]

Another proposed variant of FPL is Follow Lazy Leader (FLL). This algorithm achieves the same expected regret as FPL, but without changing its decision as often. This is accomplished by coupling the perturbations, or rather only making a single perturbation at the beginning.

Recall that our FPL perturbations were uniformly distributed over a cube with side lengths $1/\epsilon$. For FLL, we use this initial perturbation to define the offset from the origin of an entire grid of $1/\epsilon$ cubes. For each step then, the perturbed cost is taken to be the intersection of this grid with a fake perturbation box centered at the true running costs total. Because of the initial perturbation, these grid points give us the same expectation as the original FPL algorithm. However, as long as the true costs are not moving us too far compared to the grid size ($1/\epsilon$), we should be using the same grid point relatively often. Using the same grid point means that the leader is guaranteed to stay the same, avoiding both the costs of computing the leader and possibly the cost of changing the decision this round.

One commonality among these "Follow the Leader" style algorithms is that they all assume the existence of an efficient offline algorithm for calculating the current leader based on the current history of cost functions. Also, these algorithms require remembering the previous cost histories for each expert, while the gradient descent algorithm only relies on the current gradient and the previous point.

6 Online Newton Step [7]

The Online Newton method requires only that the cost functions be log-concave ($\exp(-\alpha f_t(x))$ is concave for some α), and that the gradients be bounded by G. It achieves regret $O(\log T)$ The exact formula can be interpreted as a projection with respect to a sum of gradient outer products, but can more easily be seen as another variant of follow the leader.

7 Gradient Descent as Follow the Leader

As mentioned earlier, gradient descent can be cast as a follow the leader algorithm for a suitably defined sequence of loss functions, which is an interesting result. The function definitions are

$$\tilde{f_0}(x) = \frac{(x - x_1)^2}{\eta} \tag{9}$$

$$\tilde{f_t}(x) = f_t(x_t) + \nabla f_t(x_t)^T (x - x_t) \tag{10}$$

This interpretation of gradient descent is quite interesting, because it connects the seemingly different gradient descent and follow the leader styles. In particular, the cost of a point with respect to previous cost functions is not taken for the functions themselves, but for their gradients.

This connection allows us to see more clearly what gradient descent is doing, which is not what one would first think. Gradient descent here is not trying to find the minima of each cost function, because the sequence of cost functions are not guaranteed to be related. Rather, it is trying to move slightly towards a point which would have been relatively low cost for that previous function, because the optimal offline decision must be at a point which is, overall, relatively low-cost for the sequence of cost functions.

8 Online Newton as Follow the Leader

The Online Newton method can also be defined in terms of a follow the leader strategy, known as Follow Approximate Leader. This algorithm follows the strategy outlined in [5], where a series of lower-bound functions are used to calculate the leader, as opposed to the true cost functions. Here, the lower bound functions are parabolas incident to the true cost function at the point the cost function was defined on. These lower bound functions are defined as

$$\tilde{f_t}(x) = f_t(x) + \nabla_t^T (x - x_t) + \frac{\beta}{2} (x - x_t)^T (\nabla_t \nabla_t^T)(x - x_t) \tag{11}$$

where $\nabla_t = \nabla_t f_t(x)$.

9 Discussion and Extensions

It is surprising that all algorithms discussed in this survey are expressable in the Follow the Leader framework. A notable exception, which was covered in [7] but omitted from this survey, is an exponential weighting scheme. Howver, this approach takes integrals over the feasible set, weighing points by exponentials of the cost functions, which is similar to the notion of following the leader.

This commonality reflects the "under the hood" reality of these online optimization algorithms. Aside from some assumptions of convexity or gradient

bounds, these algorithms assume virtually nothing about the series of cost functions which will be presented to them. Therefore, minimizing regret with respect to an optimal static strategy basically involves trying to play like the optimal static strategy. On any given round, the "best guess" as to the optimal static strategy is the current leader.

What is interesting about these various algorithms is the way the use additional information to augment the follow the leader technique. On the surface, the perturbation techniques of [6] could seem like a regularization technique from machine learning to avoid overfitting the observed data. While this analogy is appealing, the fundamental usage is somewhat different. In this online learning setting, no assumptions are made about cost functions being from a certain distribution, or even having anything to do with one anything whatsoever. The notion of prior knowledge or preferences with respect to online decisions is not clear. Rather, the usage of randomness in [6] seems more similar to randomized algorithms, such as those for Euclidean TSP [10], in that the chief aim is to provide good expected behavior by avoiding worst-case problem instances.

It is also important to realize that these fully general online algorithms may not be the best choice for a given problem. Online approaches for specific problems can often be designed to exploit structure known to be inherent in the problem or the cost functions. [7] mentions such algorithms for linear predictions with convex cost functions, and [6] makes a similar remark with respect to online shortest paths problems.

Another interesting setting is the *bandit* versions of these problems, where the the value of each cost function is revealed for the chosen point only. This is significantly less information than we are given for the problems discussed in this survey, and would seem to require explicit "information seeking" behavior on the part of the algorithm. In the same base setting as the $O(\sqrt{T})$ regret algorithm from [5], [1] achieves expected regret $O(n^{3/4})$ in the bandit version.

References

[1] Abraham Flaxman, Adam Kalai, H. Brendan McMahan. Online convex optimization in the bandit setting: gradient descent without a gradient. SODA, 2005.

[2] Avrim Blum, Vijay Kumar, Atri Rudra, Felix Wu. Online Learning in Online Auctions. SODA, 2003.

[3] Adam Kalai and Santosh Vempala. Efficient algorithms for universal portfolios. JMLR, 2003.

[4] Thomas Cover. Universal portfolios. Math. Finance, 1991.

[5] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. ICML, 2003.

[6] Adam Kalai, Santosh Vempala. Efficient Algorithms for Online Decision Problems. Journal of Computer and System Sciences 71(3): 291-307, 2005.

[7] Elad Hazan, Adam Kalai, Satyen Kale, Amit Agarwal. Logarithmic Regret Algorithms for Online Convex Optimization. COLT, 2006.

[8] D. MacKay. Information Theory, Inference, and Learning Algorithms. Cambridge University Press, 2003.

[9] Jyrki Kivinen, Alexander J. Smola, Robert C. Williamson. Online Learning with Kernels, NIPS 2002.

[10] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. In *Journal of the ACM*, 1996.