1 Combining Adversarial Guarantees and Fast Rates in Online Learning (Wouter Koolen, CWI)

I will give a research talk about methods for online convex optimization that are robust and adapt to a spectrum of "Easy Data".

We consider online learning algorithms that guarantee worst-case regret rates in adversarial environments (so they can be deployed safely and will perform robustly), yet adapt optimally to favorable environments (so they will perform well in a variety of settings of practical importance). In this talk we introduce the MetaGrad algorithm for online convex optimization, its luckiness bound, and consequent unified adaptivity to two common scenarios:

- curvature : we show that MetaGrad exploits exp-concavity
- stochastic case: we show that MetaGrad adapts to the Bernstein exponent (generalization of Tsybakov margin condition). MetaGrad's computational efficiency is comparable to AdaGrad's. We expect its new adaptivity to be especially useful in practice.

We have nice algorithms designed with worst-case ideas. Very often people stop there, "practitioners should just do this." Practitioners say, "I played with the parameter, changed it from the value you said it should be, and it does better." The theoreticians say, "Don't do that!"

We show two problems where you should set your parameters differently from what theoreticians have recommended.

I'll tell you the MetaGrad algorithm for online optimization. It covers worst-case, stochastic data, and curved cases

We will optimize $\min_w \sum_{t=1}^T f_t(w)$ where f_t are convex. This could be for batch training (for classification), time series (for investment), or big data.

We have a learner and environment. The learner picks a parameter setting w_1. Receive loss and gradient $f_1(w_1)$, $\nabla f_1(w_1)$. Repeat.

We care about regret $\sum_{t=1}^T f_t(w_t) - \min_u \sum_{t=1}^T f_t(u)$. The typical vanilla algorithm is online gradient descent. Choose a starting parameter w_0 and let

$$w_{t+1} = w_t - \eta_t \nabla f_t(w_t).$$

For a single fixed function, it's reasonable that you approach the optimum. It's not immediately obvious that it extends to the case the f_t change, but this is true. In the worst case, $R_T = O(\sqrt{T})$. (Assumptions: Bounded domain, bounded gradient norm, measured in L_2.)

But there is a hierarchy of loss functions; if we assume more about the loss functions we can do better.

1. convex (ex. linear, hinge absolute) $\sqrt{T}$
2. exp-concave (ex. logistic squared) $d \ln T$: I can extrapolate in a quadratic way.
3. strongly convex (ex. squared distance) $\ln T$: I can extrapolate in every direction.

How to exploit those losses. For exp-concave, run online Newton step. For strongly convex, run online gradient descent but with the parameters tuned differently.

Can we make adaptive methods for online convex optimization that are

- worst-case safe,
- exploit curvature automatically, and
- is computationally efficient?

Can we adapt to other important regimes,

- mixed or in-between cases
- stochastic data
- absence of curvature

Sometime we can get the rates in the taxonomy without the assumptions.

Main idea: for every optimization the algorithm tuning is crucial: regularization, step size, learning rate, modal complexity. Let's learn optimal tuning from data. The key obstacle is to avoid learning η at a slow rate. Otherwise the learning of the learning rate is going to kill any benefit you get!

(Competing with constant rates is enough. We don't need to compete with other schedules. The decaying rate $\frac{1}{T}$ is a confusion; it can be thought of as a constant learning rate plus quadratic losses.)

It will maintain a collection of learning rates: multiple eta gradient algorithm.

Here is a description. There are several viewpoints.

Maintain a grid of learning rates. For each run a certain algorithm. A master algorithm takes the outputs of the "slave" algorithms and makes a prediction without too much overhead. Each algorithm is a variant of online online Newton step. Algorithm i maintains parameters Σ_i, w_i and has learning rate η_i. The master maintains weights η_i. We need $\ln(T)$ subalgorithms with different learning rates.

The master algorithm takes an average $w = \frac{\sum_i \pi_i \eta_i w_i}{\sum_i \pi_i \eta_i}$. Note we need to include the learning rate η_i in the weighting.

The master plays w. We only get the gradient at the point the master played, $g = \nabla f(w)$. Make updates (cf. multiplicative weights)

$$\pi_i \hookleftarrow \pi_e e^{-\eta_i r_i - \eta_i^2 r_i^2}$$

where

$$r_i = (w_i - w)^T g.$$

Note it's important that

1. we use the regret r_i rather than the loss (there isn't a difference for a single algorithm, but there is a difference here) (actually the lower bound on regret using the linear expansion)

2. we have a quadratic term.

We call this "tilted exponential weights." It's similar to Squint (COLT 2015: track one or several numbers for each expert—this is intractable when e.g. the experts are all the points in the plane), ABProp.

The slaves have a information problem; they don't get the gradient of the point they play. Update

$$\Sigma_i \hookleftarrow (\Sigma_i^{-1} + 2\eta_i^2 g g^T)^{-1} \tag{1}$$
$$w_i \hookleftarrow w_i - \eta_i \Sigma_i g(1 + \eta_i r_i. \tag{2}$$

π determines whose weights get used, but the algorithms don't see π.

Q: if you don't know where the gradient is evaluated, how do you use it? Even if you get a lower bound in the wrong spot, it's still a lower bound on the function everywhere. All the algorithms can use the lower bound to reason on the quality of the w's.

Slaves assume curvature even if it isn't there. We show it doesn't hurt if they are wrong. (We only need to compare with the slave that got it right.) Even if there is no curvature, he best slave might be one that assumes there is curvature.

Now we need to maintain the matrices; this is the bottleneck. The compexity is d^2; you can use Sherman-Morrison to update the inverse, but this breaks the relationship between the different Σ_i.

Theorem 1.1. *The regret of MetaGrad is bounded by*

$$R_T = O\left(\min\left\{\sqrt{T}, \sqrt{V_T d \ln T}\right\}\right)$$

where

$$V_T = \sum_{t=1}^{T}((w_t - u^*)^T \nabla f_t(w_t))^2.$$

The V's are sums over rounds of the difference between what the learner played the the best in hindsight, times the gradient, squared. V measures variance compared to the offline optimum $u^* = \text{argmin}_u \sum_{t=1}^T f_t(u)$.

Here the V_T is unknown and could change. This makes the tuning tricky.

Corollary 1.2. *For α-exp-concave or α-strongly convex losses, MetaGrad ensures $R_T = O(d \ln T)$ without knowing α.*

This is because the curvature implies $\Omega(V_T)$ cumulative slack between loss and its tangent lower bound.

One consequence of this argument is that it applies for a fixed function. We get the same result for fixed $f_t = f$ (classical optimization) even without curvature via derivative condition.

(If you knew exp-concave, you could do grad descent with learning rate instead. Add a second family of slaves that do gradient descent with different learning rate. Some voices in ML community tell me strongly convex losses don't exist.)

One example: if you're obtimizing $|x - a|$, our algorithm gets close to the bottom and makes smaller error than typical gradient descent.

Now consider the stochastic case. When you design algorithms that are supposed to work in practice, you should test on the stochastic case! Many adaptive algorithms don't think this is special. This is closest model to what is encountered in practice.

Consider iid losses $f_t \sim \mathbb{P}$ with stochastic optimum $u^* = \operatorname{argmin}_u \mathbb{E} f(u)$. The goal is small pseudo-regret (compared to u^*)

$$R_T^* = \sum_{t=1}^T f_t(w_t) = \sum_{t=1}^T f_t(u^*).$$

The assumption is that the loss of a point close to the optimum is not too different from the loss at the optimum. The higher the parameter, the more severe the condition, the easier the data is.

Corollary 1.3. *For any β-Bernstein $\mathbb{P}$, MetaGrad keeps the expected regret below*

$$\mathbb{E}R_T \leq O((d \ln T)^{\frac{1}{2-\beta}} T^{\frac{1-\beta}{2-\beta}}).$$

This is because Bernstein bounds $\mathbb{E}[V_T^*]$ above by $\mathbb{E}[R_T^*]$.

There are cases where AdaGrad gets worst case $O(\sqrt{T})$ rate while MetaGrad gives $O(\ln T)$, ex. $f_t(u) = |u - \frac{1}{4}|$ for the offline case or $f_t(u) = |u - x_t|$ where $x_t = \pm\frac{1}{2}$ iid with probabilities $0.4, 0.6$ for the online case.

The gradients are not going to 0; the point jumps from left to right.

Summary: we get $\sqrt{T}$ in worst case $T^{\frac{1-\beta}{2-\beta}}$ for stochastic, $d \ln T$ for curved.

Code: `http://bitbucket.org/wmkoolen/metagrad`

1.1 Proofs

1. illegible bound
2. illegible bound $\implies$ exp-concave
3. illegible bound $\implies$ Bernstein stochastic

$$R_T^u = \sum_t f_t(w_t) - f_t(u) \tag{3}$$
$$\leq R_T^u = \sum_t \langle w_t - u, \nabla f_t(w_t) \rangle \tag{4}$$
$$\leq \sqrt{V_T^u d \ln T} \tag{5}$$
$$V_T^u = \sum_t \langle w_t - u, \nabla f_t(w_t) \rangle^2 \tag{6}$$

The first thing you do with exp-concavity is convert it in a quadratic lower bound. For all u,

$$f(u) \geq f(w) + \langle u - w, \nabla f(w_t) \rangle + \beta \langle u - w, \nabla f(w_t) \rangle^2 \tag{7}$$

Take linear approximation and add something that curves in one direction (gradient direction). It looks liek a gutter. This is different from strong convexity where you have a vase.

$$\begin{aligned} R_T &= \sum_t f_t(v_t) - f_t(u) && (8) \\ &\leq \widetilde{R}_T - \beta \sum_t \langle u - w, \nabla f(w_t) \rangle^2 && (9) \\ &\leq V_\beta - V_\beta + \frac{d \ln T}{\beta} && (10) \end{aligned}$$

using $2\sqrt{V d \ln T} \leq V_\beta + \frac{d \ln T}{\beta}$. Similar arguments work in the stochastic case.