Statistical Techniques in Robotics (16-831, F11) Lecture#11 (Wednesday October 5)

Online Convex Programming

Lecturer: Drew Bagnell *Scribe: Prateek Tandon*[1]

1 Online Convex Programming Problems

We shall continue to develop the framework from last lecture of casting Online Learning Problems as Optimization problems (and especially Convex Optimization Problems). Convex Optimization Problems are a key framing choice since these problems have a structure that makes them easy to solve.

1.1 Definition

Online Convex Programming was proposed by Martin Zinkevich in 2003. Problems are framed in the same context of time steps, loss function, experts as the standard weighted majority algorithm. Algorithms for Online Convex Programming preserve many of the same qualities of the weighted majority algorithm while being computationally tractable. An online convex programming problem consists of:

- A set of experts that are elements of some convex set.
- A loss function $l_t(x_t)$ that is a convex over the set of experts
- The goal at each time step of predicting an expert w_t to minimize the regret given by:

$$R(w) = \sum_{t=0}^{T} l_t(w_t) - l_t(w^*). \tag{1}$$

  where w_t is an expert and w^* is the best expert in retrospect.

The instantaneous regret for some expert w_t at time t is:

$$R_{inst}(w) = l_t(w_t) - l_t(w^*) \tag{2}$$

1.2 Convex Functions

Recall that a convex set is a set such that any linear combination of two points in the set is also in the set: if $A \in C, B \in C$, then

$$\theta A + (1-\theta)B \in C \tag{3}$$

[1]Some content adapted from previous scribes: Anca Dragan, Siyuan Feng

A function is convex if its epigraph (the set of points on or above the graph of the function) is a convex set.

A subgradient at x, $\nabla f(x)$, is a line below the entire function and is given by:

$$f(y) \geq f(x) + \nabla f(x)^T (y - x), \forall y \tag{4}$$

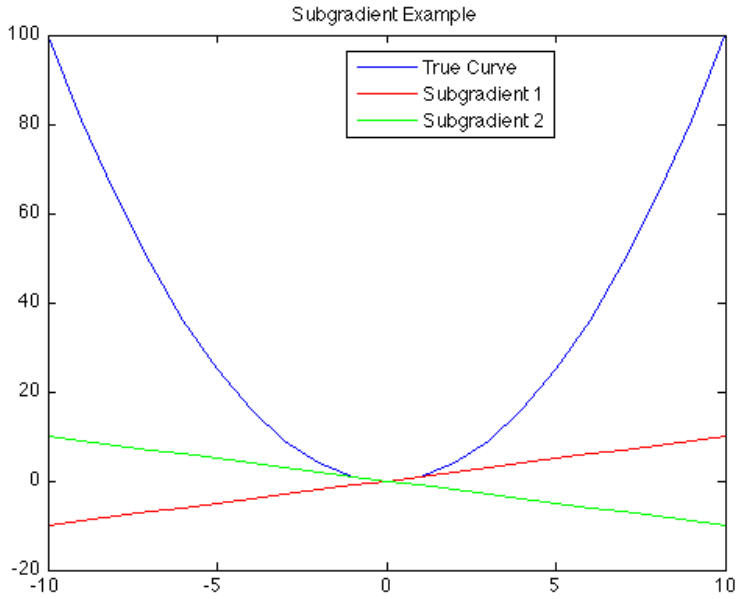

Figure 1: Examples of Subgradients

Convex functions have subgradients at every point in their domain. This property simply re-states that for any point in the domain of the function the value of the function at that point needs to be higher than the value of the subgradient line at the same point.

The subdifferential is the set of all subgradients at a point. Note that a function always has at least one subgradient. If a function is nondifferentiable at a point then it has infinite subgradients at that point. Convex functions are the max over all subgradients.

1.3 Bounding Instantaneous Regret

For our instantaneous regret, we have

$$\begin{aligned} l_t(w^*) &\geq l_t(w_t) + \nabla l_t(w_t)^T (w^* - w_t) \\ l_t(w_t) - l_t(w^*) &\leq \nabla l_t(w_t)^T (w_t - w^*). \end{aligned} \tag{5}$$

The left hand side is the instantaneous regret, and the right hand side is some linear function times $(w_t - w^*)$. Thus our total regret will be bounded by $\sum_{t=0}^{T} \nabla l_t(w_t)^T (w_t - w^*)$.

It is convenient that we have a linear bound on the instantaneous regret because linear functions are generally hard to bound. This means we can easily bound loss functions that are much easier

to bound than linear (i.e. convex functions). Finally, the bound tells us that we don't need to know the whole loss function to bound it - just the slopes!

1.4 Follow the Leader

The follow the leader algorithm suggested in the previous class might a decent starting point for selecting experts at each time step.

Consider a game against nature on the real line between $[0, 1]$. In this game, you select a point between 0 and 1 and nature hands you a loss function from which you can compute your loss. Imagine choosing the boundary point 0. Then nature might assign you the linear loss function that has minimum at 0 and maximum at 1. The "follow the leader" algorithm would encourage you to pick 1 as your next move. However on that move, nature decides to assign you the linear loss function that has minimum at 1 and maximum at 0. So you decide to move 0 again to follow the leader but nature again assigns a terrible loss function for you at 0. The net effect is that nature makes you go back and forth from move to move and your regret gets really bad.

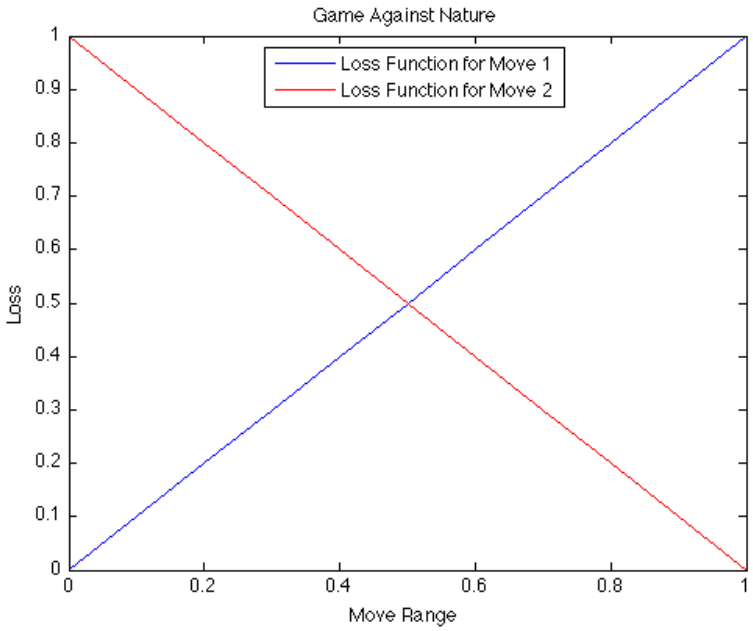

Figure 2: Follow the Leader is too aggressive and oscillates between 0 and 1 in this example.

This is an example of how the follow the leader can break down because it is too aggressive. In the next section we see a more powerful algorithm to minimize regret.

1.5 Algorithm for projected online subgradient descent

This algorithm is a method to minimize the regret for a online convex optimization problem.

Line 5 projects $\hat{w}_{t+1}$ back into the convex set C. α in line 4 is the learning rate. Smaller α pays a

Algorithm 1 Projected Subgradient Descent():

choose w_0
for $t = 1...T$ **do**
  Incur loss $l(w_t)$ and receive any $\nabla l_t(w_t)$
  $\hat{w}_{t+1} \leftarrow w_t - \alpha \nabla l_t(w_t)$
  $w_{t+1} \leftarrow Proj_c[\hat{w}_{t+1}]$
end for

larger upfront cost but is more likely to converge and has a lower regret over time. α can also be dependent on t. Note that the projection will not cause the loss to grow because it will bring $\hat{w}_{t+1}$ closer to any member of C and thus closer to the optimal expert w^* too.

1.5.1 Carrot/Tree Example

We can cast a linear regression problem as an instance of using this algorithm. Recall from last lecture our carrot/tree world where a robot must drive on 1D line to classify an obstacle that is either a carrot or tree. The robot's perception system logs at each time step a list of features such as distance to obstacle and RGB values. As the robot gets closer its predictions approach the ground truth. The robot will know the true label once it sees the obstacle up close. For this problem we have a feature set at every time step [d,r,g,b] and the robot assigns a label to this feature set: 0 if the obstacle is a carrot and 1 if the obstacle is a tree.

We can solve this problem by assigning the loss function:

$$l_t = 1/2(w^T f - y_t)^2 \tag{6}$$

where w_t are the weights we predict, f is the feature set and y_t is the world handing us the true label of the object. We pay cost l_t when we find the true label of the obstacle y_t from the world.

We use the update following update equation for our weights:

$$w_{t+1} = w_t - \alpha(w^T f - y_t) \tag{7}$$

which is basically a step in gradient descent.

1.6 Regret bounds for projected subgradient descent

The distance between w_t and w^* at time t is defined as

$$D(w_t, w^*) = (w_t - w^*)^T (w_t - w^*) \tag{8}$$

Now we look at

$$\begin{aligned} & D(w_{t+1}, w^*) - D(w_t, w^*) \\ & = (w_t - \alpha \nabla l_t(w_t) - w^*)^2 - (w_t - w^*)^2 \\ & = (z_t - \alpha \nabla l_t(w_t))^2 - z_t^2 \\ & = \alpha^2 (\nabla l_t(w_t))^2 - 2\alpha \nabla l_t^T(w_t) z_t, \end{aligned} \tag{9}$$

where $z_t = w_t - w^*$. If we sum all the terms over time, the intermediate terms will all cancel out and leave us just $D(w_T, w^*) - D(w_0, w^*)$.

$$\begin{aligned} &= \sum_t D(w_{t+1,w^*}) - D(w_t, w^*) \\ &= -2\alpha \sum_t (w_t - w^*)\nabla l_t + \alpha^2 \sum_t |\nabla l_t|^2 \\ &= D(w_T, w^*) - D(w_0, w^*) \\ &\leq -2\alpha \sum_t (w_t - w^*)\nabla l_t + \alpha^2 GT, \end{aligned} \tag{10}$$

where $|\nabla l_t|^2 \leq G$. Thus we have

$$2\alpha R_T \leq 2\alpha \sum_t (w_t - w^*)\nabla l_t \leq D(w_0, w^*) - D(w_T, w^*) + \alpha^2 GT \tag{11}$$

Sine the distance between w_T and w^* is always non negative, we can throw away the $D(w_T, w^*)$ term and still keep the inequality valid.

$$R_T \leq \sum_t (w_t - w^*)\nabla l_t \leq \frac{D(w_0, w^*)}{2\alpha} + \frac{\alpha GT}{2} \leq \frac{\alpha GT}{2} + \frac{F}{2\alpha}, \tag{12}$$

where F is the largest distance between any two experts in the set.
Suppose we set $alpha = \sqrt{\frac{F}{GT}}$, then the upper bound for total regret is bounded by $\sqrt{GTF}$, growing sub linearly of T.

Note: For those from CS, we just did an amortized analysis.

2 Portfolio optimization

We want to invest in n different stocks given a set of investment weights w_i s.t. $w_i \geq 0$ and $\sum w_i = 1$. We also know the market returns ratios $r_i = \frac{value^i_{t+1}}{value^i_t}$. So the daily increase in wealth is $w_t^T r_t$, and the total wealth over time is $m\Pi w_t^T r_t$, where m is the total value of initial investment. We want to maximize $\log \Pi w_t^T r_t = \sum \log w_t^T r_t$. We can use use the following algorithm:

- $w_0^i = \frac{1}{n}$
- $w_{t+1}^i = Proj[w_t^i + \alpha \frac{r_t^i}{\sum w_t^j r_t^j}]$

We shall compare the policy to a constantly rebalancing portfolio that maintains a set constant investment ratios. A key question for next time is: if we get off a simplex, how can we get back on?