

## Convex Optimization and Gradient Descent

Lecturer: Drew Bagnell

Scribe: Yi Zhang

### 1 Online Convex Programming

Consider a *convex* set  $G$  of experts, where each expert  $w \in G$  is, e.g., an assignment for the parameters of our prediction model. At each time point  $t$ , we have a *convex* loss function  $l_t(w)$  defined on any expert  $w \in G$ . The optimal expert in  $w^* \in G$  is defined as:

$$w^* = \operatorname{argmin}_{w \in G} \sum_{t=1}^T l_t(w) \quad (1)$$

Our goal is by online choosing a set of expert  $w_t$  at each time point  $t$ , to minimize the overall regret:

$$R(\{w_t\}_{t=1}^T) = \sum_{t=1}^T R_t(w_t) = \sum_{t=1}^T [l_t(w_t) - l_t(w^*)] \quad (2)$$

### 2 Subgradients

#### 2.1 Definition

Given a convex function  $f()$ , a subgradient of  $f()$  at point  $x$  is defined as any vector  $\nabla f_x$  that satisfies (for any  $y$ ):

$$f(y) \geq f(x) + \nabla f_x^T (y - x) \quad (3)$$

Briefly, if a convex function  $f()$  is differentiable at  $x$ , the subgradient of  $f()$  at  $x$  is unique and is the gradient at  $x$ . Otherwise, we can have more than one subgradients for  $f()$  at  $x$ .

#### 2.2 Subgradient for online learning

According to the definition of subgradient, we can derive an upper bound for the regret:

*Proof.*

$$l_t(w^*) \geq \nabla l_t(w_t)^\top (w^* - w_t) + l_t(w_t) \quad (4)$$

$$R_t(w_t) = l_t(w_t) - l_t(w^*) \leq \nabla l_t(w_t)^\top (w_t - w^*) \quad (5)$$

$$R(\{w_t\}_{t=1}^T) = \sum_{t=1}^T R_t(w_t) = \sum_{t=1}^T [l_t(w_t) - l_t(w^*)] \leq \sum_{t=1}^T [\nabla l_t(w_t)^\top (w_t - w^*)] \quad (6)$$

From this upper bound, we can see that the regret at time  $t$ , i.e.,  $R_t(w_t)$ , is highest if  $w_t - w^*$  is on the direction of the subgradient  $\nabla l_t(w_t)$ . Of course we won't let it happen ...

### 3 Projected Subgradient Descent

#### 3.1 Algorithm

Yeah! Let's minimize our overall regret by minimizing the upper bound. At each time  $t$ , we achieve this by pushing  $w_t$  towards the reverse direction of the latest subgradient. The algorithm, called Projected Subgradient Descent, is as follows:

---

**Algorithm 1** Projected Subgradient Descent():

---

- 1: Initialize  $w_1 \in G$
  - 2: **for**  $t = 1 \dots T$  **do**
  - 3:   Predict using  $w_t$ , incur loss  $l_t(w_t)$ , and get subgradient  $\nabla l_t(w_t)$
  - 4:    $\hat{w}_{t+1} \leftarrow w_t - \alpha \nabla l_t(w_t)$
  - 5:    $w_{t+1} \leftarrow Proj_G[\hat{w}_{t+1}]$
  - 6: **end for**
- 

The line 5 ( $w_{t+1} \leftarrow Proj_G[\hat{w}_{t+1}]$ ) projects the weights  $\hat{w}_{t+1}$  back into the convex set  $G$ . Thanks to the convexity,  $w_{t+1}$  is closer to any point in  $G$  than  $\hat{w}_{t+1}$ .

In line 4,  $\alpha$  represents the step size, or learning rate. Large values of  $\alpha$  will learn faster but are less likely to converge as it may step over the minimal. Smaller values pay a larger upfront cost but are more likely to converge and have a lower regret over time. There are various methods where  $\alpha_t$  decreases over time.

#### 3.2 Projected Gradient Descent Regret Bounds

Distance between optimal weight vector  $w^*$  and the weight vector at time  $t$  ( $w_t$ ).

$$D(w_t, w^*) = (w_t - w^*)^T (w_t - w^*) \tag{7}$$

then the distance after the update for the next time step is:

$$D(w_{t+1}, w^*) - D(w_t, w^*) = \|w_t - \alpha \nabla l_t - w^*\|^2 - \|w_t - w^*\|^2 \tag{8}$$

Now we substitute  $z_t = w_t - w^*$  and get

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

canceling and substituting in for  $z_t$  gives us:

$$D(w_{t+1}, w^*) - D(w_t, w^*) = -2\alpha \nabla l_t^T (w_t - w^*) + \alpha^2 \nabla l_t^2 \tag{10}$$

Now we can sum over time ( $t = 0, 1, \dots, T - 1$ ) to get the distance up until time  $T$ .

$$\sum_t (D(w_{t+1}, w^*) - D(w_t, w^*)) = -2\alpha \sum_t (\nabla l_t^T(w_t - w^*)) + \sum_t (\alpha^2 \nabla l_t^2) \quad (11)$$

$$\sum_t (D(w_{t+1}, w^*) - D(w_t, w^*)) = D(w_T, w^*) - D(w_0, w^*) \quad (12)$$

The overall regret  $R()$  is bounded by:

$$R(\{w_i\}_{i=0}^{T-1}) \leq \sum_t (\nabla l_t^T(w_t - w^*)) = -\frac{1}{2\alpha} \left[ D(w_T, w^*) - D(w_0, w^*) - \sum_t \alpha^2 \nabla l_t^2 \right] \quad (13)$$

Next, we need to assume an upper bound  $G$  for  $\|\nabla l_t\|$ :

$$\nabla l_t^2 \leq G^2 \quad (14)$$

The bound on the overall regret then becomes:

$$R(\{w_i\}_{i=0}^{T-1}) \leq \frac{\alpha}{2} G^2 \cdot T + \frac{1}{2\alpha} D(w_0, w^*) - \frac{1}{2\alpha} D(w_T, w^*) \quad (15)$$

Since we want an upper bound and the last term can only decrease this bound, we can eliminate it.

$$R(\{w_i\}_{i=0}^{T-1}) \leq \frac{\alpha}{2} G^2 \cdot T + \frac{1}{2\alpha} D(w_0, w^*) \quad (16)$$

Since we are dealing with a bounded convex set of  $w$  vectors, let us bound  $D(w_0, w^*)$  by the size of the set  $F^2$ . Now we need to pick  $\alpha$  to minimize regret. To do this, we take the derivative of the above equation w.r.t  $\alpha$  and set it equal to 0. Some algebra gives us

$$\alpha = \frac{F}{G\sqrt{T}} \quad (17)$$

and the regret is thus bounded by<sup>1</sup>

$$R(\{w_i\}_{i=0}^{T-1}) \leq FG\sqrt{T} \quad (18)$$

note that the regret grows sub-linearly with time which means that as  $t \rightarrow \infty$ :

$$\frac{R}{t} \rightarrow 0 \quad (19)$$

Thus this algorithm is a “no-regret” algorithm.

---

<sup>1</sup>Drew says the right bound is actually  $R(\{w_i\}_{i=0}^{T-1}) \leq 2FG\sqrt{T}$

### 3.3 Examples

- Tree Prediction:

Given good object detection up-close, trees in this example, we want to learn to recognize similar objects far away. An example of this would be to have good laser scan data for short distances and trying to extend that to far objects detected via image characteristics (shape, texture, color). The algorithm will try to detect objects far away and then check if it was right when closer; this is an example of self-supervised online learning. Because what we predict influences what we see in the future, regret might not be a good measure of performance as it might change depending on the examples given.

Now, we are given the following state features:

- $x_1 = Red, x_2 = Blue, x_3 = Green$
- $x_4 = \text{point spread}$
- $x_5 = \text{response of Gaber filter on image patch (texture)}$
  
- $y_t = \{1, -1\} \leftarrow \text{tree or no tree}$

Using the loss function:

$$l_t(w_t) = (w_t^\top x_t - y_t)^2 \quad (20)$$

We can use the following update rule:

$$w_{t+1} \leftarrow w_t - 2\alpha(w_t^\top x_t - y_t) \cdot x_t \quad (21)$$

where  $(w_t^\top x_t - y_t)$  is the residual.

- Portfolio Optimization:

We are given a set of investment weights,  $w_t$ , and daily market return ratios,  $r_t$  such that:

$$w_t \geq 0 \quad (22)$$

$$\sum w_t = 1 \quad (23)$$

The daily increase in wealth is:

$$w_t^\top r_t \quad (24)$$

and total wealth over time is:

$$\prod_{t=1}^T w_t^\top r_t \quad (25)$$

We want to optimize the log-gain function:

$$\sum_{t=1}^T \log(w_t^\top r_t) \quad (26)$$

We will compare the policy to a constantly rebalancing portfolio that adjusts the investments each day to have constant investment ratios.

We make the following adjustments to the Projected Subgradient Descent algorithm:

- $w_0^i = \frac{1}{N}$
- $w_{t+1}^i \leftarrow w_t^i + \alpha \frac{r_t^i}{\sum_j w_t^j \cdot r_t^j}$
- Project each  $w_{t+1}^i$  back to the convex function (simplex)

The following problems exist with this algorithm, so you won't get "fabulously wealthy":

- Constant balance may not necessarily be the best policy over time
- Fixed transaction costs add up
- Large actions will affect the stock's market price