10-425/625: Introduction to Convex Optimization **(Fall 2023)**

Lecture 22: Adaptive Methods / Regret Minimization

Instructor:[1] *Matt Gormley* *November 27, 2023*

22.1 More Adaptive Gradient Algorithms

22.1.1 The problem with AdaGrad

Last time, we considered the AdaGrad algorithm which adaptively adjusts the step size for each parameter x_j separately.

AdaGrad (Duchi, Hazan, and Singer 2010): Let $g^{(t)} = \nabla f_{i_t}(x^{(t-1)})$, and update for $t = 1, 2, 3, \ldots$:

$$x_j^{(t)} = x_j^{(t-1)} - \alpha \frac{g_j^{(t)}}{\sqrt{\sum_{s=1}^{t} (g_j^{(s)})^2 + \epsilon}}$$

With an abuse of notation in which division and multiplication are elementwise between vectors and squaring $(\cdot)^2$ is applied elementwise, we can write the AdaGrad update as follows:

$$x^{(t)} = x^{(t-1)} - \frac{\alpha}{\sqrt{\sum_{s=1}^{t} (g^{(s)})^2 + \epsilon}} g^{(t)}$$

The problem with AdaGrad is that its step sizes are always decreasing. That is, the squares of the gradients are always positive and as they accumulate in the denominator they eventually become infinitesimally small. This has the potential to slow down training in practice unless the initial step size α is carefully chosen.

[1]

22.1.2 RMSProp

RMSProp is an unpublished algorithm that appeared in Lecture 6[2] of Geoff Hinton's Coursera course in 2012 that became very popular for training neural networks. The basic idea is to replace AdaGrad's full sum over squared gradients $\sum_{s=1}^{t}(g^{(s)})^2$ with an exponentially decaying average $\sum_{s=1}^{t}\gamma^{t-s}(1-\gamma)(g^{(s)})^2$. The update rule can be efficiently computed as:

$$\begin{aligned} v^{(t)} &= \gamma v^{(t-1)} + (1-\gamma)(g^{(t)})^2 \\ x^{(t)} &= x^{(t-1)} - \frac{\alpha}{\sqrt{v^{(t)}+\epsilon}} g^{(t)} \end{aligned}$$

where again we assume the square $(\cdot)^2$ is applied elementwise, and operations involving $x^{(t-1)}, v^{(t)}, g^{(t)} \in \mathbb{R}^d$ are elementwise as well.

Intuitively, the $v^{(t)}$ term keeps a memory of the recent squared gradients, but to allow that memory to gradually fade. As such, the adaptive step size can gradually grow and shrink over time as the recent gradients change.

This is particularly helpful in training neural networks for several reasons:

- The objective function for a neural network is nonconvex. At the start of training, the slope along one dimension x_j may be very steep, but very flat for another x_i. However, after passing some saddle point the optimization may move into a very different topography in which the slope along x_i is stepp and x_j is shallow. RMSProp and related algorithms can account for this change over time.
- A well known issue when training neural networks is the vanishing gradient problem. That is, the gradient for parameters in lower layers of the network may be very small near the start of training, while the gradients are very large for the parameters in high layers of the network. Again, RMSProp handles this case elegantly by enabling the low-layer parameters to move faster earlier during training, and then slow down later when the size of their gradients catches up to those of the higher layers.

The name RMSProp derives from the exponentially decaying average of the root mean squared (RMS) of the squared gradients; and its close connection to the rprop algorithm.

[2] `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`

22.1.3 AdaDelta

In the same year, 2012, AdaDelta was introduced to combat the same issues with AdaGrad. The paper introduced two ideas independently: Idea 1 was identical to RMSProp. Idea 2 got rid of the learning rate α altogether by scaling by an exponentially decaying average of the root-mean-squared of the changes to the parameters $\delta x^{(t-1)} = (x^{(t-1)} - x^{(t-2)})$.

$$\begin{aligned} u^{(t)} &= \gamma u^{(t-1)} + (1-\gamma)(x^{(t-1)} - x^{(t-2)}) \\ v^{(t)} &= \gamma v^{(t-1)} + (1-\gamma)(g^{(t)})^2 \\ x^{(t)} &= x^{(t-1)} - \frac{\sqrt{u^{(t)} + \epsilon}}{\sqrt{v^{(t)} + \epsilon}} g^{(t)} \end{aligned}$$

This has an intuitive motivation if we consider a setting in which the Hessian matrix $H = \nabla^2 f(x)$ is a diagonal matrix. In this case, the inverse Hessian is simply $H^{-1} = \frac{1}{\nabla^2 f(x)}$. Then the Newton update is $x^{(t+1)} = x^{(t)} + \Delta x^{(t)}$ where $\Delta x^{(t)} = (H^{(t)})^{-1} g^{(t)} = \frac{g^{(t)}}{H^{(t)}}$. Rewriting, we obtain that $\frac{1}{H^{(t)}} = \frac{\Delta x^{(t)}}{g^{(t)}}$. Since AdaDelta is a first-order method, we can not obtain $\Delta x^{(t)}$. However, the numerator can be viewed as working in the units of some recent values of $\Delta x^{(t)}$ via an exponentially decaying average.

22.1.4 Recall: SGD with Classical Momentum

Recall SGD with classical momentum for a problem $f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$. At each timestep $t = 1, 2, 3, \ldots$, we compute the gradient of a minibatch $I_t \subseteq \{1, \ldots, n\}$ of size m:

$$g^{(t)} = \frac{1}{m} \sum_{i \in I_t} \nabla f_i(x^t)$$

Then we compute a direction to step $m^{(t)}$ and update our parameters as:

$$\begin{aligned} m^{(t)} &= \beta m^{(t-1)} + (1-\beta) g^{(t)} \\ x^{(t)} &= x^{(t-1)} - m^{(t)} \end{aligned}$$

Above we show the one hyperparameter version. The version with two replaces $(1-\beta)$ with a learning rate hyperparameter η.

22.1.5 Adam

Although Geoff Hinton's slides mention that there were various attempts at incorporating momentum into RMSProp around 2012, they were not met with great success until the Adam algorithm appeared in 2015.

Adam (Kingma & Ba, 2015) combines RMSProp with classical momentum. The basic idea is to scale standard classical momentum by the RMS of the inverse gradients. We initialize $m^{(0)} = 0$ and $v^{(0)} = 0$. Then for timesteps $t = 1, 2, 3, \ldots$:

$$m^{(t)} = \beta m^{(t-1)} + (1-\beta) m^{(t)} \tag{22.1}$$

$$v^{(t)} = \gamma v^{(t-1)} + (1-\gamma)(g^{(t)})^2 \tag{22.2}$$

$$\hat{m}^{(t)} = \frac{1}{1-\beta^t} m^{(t)} \tag{22.3}$$

$$\hat{v}^{(t)} = \frac{1}{1-\gamma^t} v^{(t)} \tag{22.4}$$

$$x^{(t)} = x^{(t-1)} - \frac{\alpha}{\sqrt{\hat{v}^{(t)}} + \epsilon} \hat{m}^{(t)} \tag{22.5}$$

This is a straightfoward combination of RMSProp and Classical Momentum, with a rescaling in the 3rd and 4th lines above.

Bias Correction We can interpret $m^{(t)}$ and $v^{(t)}$ as biased estimates of the mean and variance of the gradient. However, they are biased estimates because we initialized $m^{(0)} = 0$ and $v^{(0)} = 0$. We can correct for that bias by examining the expected value of, say, $v^{(t)}$:

$$\begin{aligned}
\mathbb{E}[v^{(t)}] &= \mathbb{E}\left[\gamma v^{(t-1)} + (1-\gamma)(g^{(t)})^2\right] \\
&= \mathbb{E}\left[\sum_{s=1}^{t} \gamma^{t-s}(1-\gamma)(g^{(s)})^2\right] \\
&= \sum_{s=1}^{t} \gamma^{t-s}(1-\gamma)\mathbb{E}[(g^{(s)})^2] \\
&= \left[\mathbb{E}[(g^{(t)})^2] \sum_{s=1}^{t} \gamma^{t-s}(1-\gamma)\right] + \xi \\
&= \mathbb{E}[(g^{(t)})^2](1-\gamma^t) + \xi
\end{aligned}$$

Above, $\xi = 0$ if $\mathbb{E}[(g^{(s)})^2]$ is stationary. Thus, we divide by $(1-\gamma^t)$ to obtain a corrected estimate of the variance.

Adam in Practice Adam works well very a variety of problem types, but particularly for deep neural networks. The typical hyperparameter settings are $\gamma = 0.999$ (pushing it closer in behavior towards AdaGrad), $\beta = 0.9$ (so that we can frequently renew our momentum direction), $\epsilon = 10^{-8}$, $\alpha = 0.001$.

For training on large datasets, it is common to use learning rate warmup in which the pre-initial learning rate α is set to an even smaller value and gradually increased up to its initial value over the course of some iterations. This may also help the values of $v^{(t)}$ and $m^{(t)}$ settle in more gradually away from their initial zero values.

Convergence Analysis of Adam The convergence analysis of Adam, and related adaptive gradient algorithms, stems from the online gradient descent analysis of Zinkevich (2003). The key idea is to bound the regret of algorithm in hindsight. We develop these ideas in the next section.

The key result is that with mild assumptions (convexity, bounded gradients, bounded distance between iterates) we can show that Adam achieves $O(\sqrt{T})$ bound on the regret $R(T)$.

22.2 Regret

We consider the settings of stochastic learning and online optimization. Below we formalize the problem as online learning.

Online Learning In the online learning setting, we choose a sequence of parameters x^t for $t = 1, 2, 3, \ldots$. At each time step t, some adversary gives us another loss function f_t and we receive the loss $f_t(x^t)$.

Regret The goal is then to ensure that the total loss up to each time step T, $\sum_{t=1}^{T} f_t(x^t)$ is not much worse (larger) than $\min_x \sum_{t=1}^{T} f_t(x)$, which is the smallest total loss of any fixed set of parameters x chosen retrospectively.

$$R(T) = \sum_{t=1}^{T} f_t(x^t) - \min_x \sum_{t=1}^{T} f_t(x) \tag{22.6}$$

Our goal is to choose an algorithm which bounds this regret.

The offline learner If we denote the best single set of parameters in hindsight by

$$\hat{x} = \operatorname*{argmin}_x \sum_{t=1}^{T} f_t(x),$$

we can see that this is an *offline* algorithm for choosing a single set of parameters that minimize the cost $f(x) = \sum_{t=1}^{T} f_t(x)$. This offline algorithm has an advantage over the online learner in that it has full information about all the losses $f_1, \ldots, f_T$ ahead of time. However, it is at a disadvantage because it must choose only a single $\hat{x}$ to use at all timesteps. In this way, if an adversary chooses f_{t-1} and f_t so that they vary wildly, the oracle will have a difficult time satisfying both losses. As well, as T grows, the offline model will see less of an advantage if there is little volatility in the sequence of f_t.

Why do we bound regret? As we will see below, Zinkevich (2003) presents a framework for bounding the regret that has influenced much of the theory for stochastic optimization that followed. This turns out to be a compelling theoretical result for SGD and its variants. In this setting, we let the sequence of functions $f_1, f_2, f_3, \ldots$ be defined as follows: at time t, we select a mini-batch of training examples I_t and let f_t be the average loss on those examples.

If we had some offline algorithm (e.g. gradient descent, Newton's method) that works directly with the average of the f_t functions directly, it will find some $\hat{x} = \operatorname{argmin}_x \sum_{t=1}^{T} f_t(x)$.

A bound on our regret $R(T) = \sum_{t=1}^{T} f_t(x^t) - \sum_{t=1}^{T} f_t(\hat{x})$ should show that the average loss that we accumulate throughout the course of stochastic training is (hopefully) not too much worse than the *final* loss obtained by the non-stochastic algorithm upon convergence.

22.3 Online Gradient Descent

The *online gradient descent* algorithm of Zinkevich (2003) is the online variant of projected gradient descent. That is, suppose we have a constrained problem $\min_{x \in C} \sum_{t=1}^{T} f_t(x)$, but we only get to see one $f_t(\cdot)$ at a time, and must choose a corresponding $x^{(t)}$ in response before the next $f_{t+1}(\cdot)$ arrives.

The update rule uses the projection operator P_C at each iteration to project a gradient step back onto the feasible set:

$$\begin{aligned} & x^{(t+1)} = P_C(x^{(t)} - \eta_t \nabla f_t(x^{(t)}) \\ & \text{where } P_C(x) = \operatorname*{argmin}_{y \in C} \|x - y\|_2 \end{aligned}$$

If we choose $C = \mathbb{R}^d$, we obtain the *online gradient descent* algorithm.

Regret Analysis For the online gradient descent algorithm, we can obtain the following regret bound.

Theorem 22.1. *Assume our sequence of functions $f_1, \ldots, f_T$ has bounded gradients $\|\nabla f_t(x)\|_2 \leq G, \forall t, x \in C$ and the diameter of C is bounded $\max_{x,y \in C} \|x - y\|_2 \leq D$. If $\eta = \frac{D}{G\sqrt{T}}$, then*

$$R(T) = \sum_{t=1}^{T} f_t(x^t) - \min_{x \in C} \sum_{t=1}^{T} f_t(x) \leq DG\sqrt{T}$$

Segue... Next time, we'll prove the theorem.