CS 152—Notes on Optimization Algorithms (Rev. 2)

Neil Rhodes

September 30, 2019

1 Optimization Algorithms

The loss function, L, is a function of x, y, and, conceptually, Θ. We can think of L as taking note only x and y as parameters, but Θ as well, and can represent an evaluation of L as $L(\Theta; x, y)$ where I separate Θ from x and y with a semicolon because it seems like a different type of parameter than x and y.

The partial derivative $\frac{\partial L}{\partial \Theta_i}$ is also a function that takes Θ, x, and y as parameters. We can similarly represent an evaluation of that function as $\frac{\partial L}{\partial \Theta_i}(\Theta; x, y)$.

I'll use a notation of V_t to represent the value of V at time t. I'll use Θ_t to represent the value of Θ at time t. $(\Theta_t)_i$ will represent the i'th parameter from Θ_t. For partial derivatives, though, which don't need to represent time, we'll use $\frac{\partial L}{\partial \Theta_i}$ to represent the partial derivative of L with respect to the ith parameter Θ_i (that is, the subscript will refer to the parameter, not to the timestep).

1.1 Standard optimization

$$(\Theta_t)_i = (\Theta_{t-1})_i - \lambda \frac{\partial L}{\partial \Theta_i}(\Theta_{t-1}; x, y)$$

1.2 Momentum

$$(V_t)_i = \beta (V_{t-1})_i + (1-\beta) \frac{\partial L}{\partial \Theta_i}(\Theta_{t-1}; x, y)$$
$$(\Theta_t)_i = (\Theta_{t-1})_i - \lambda V_t$$

Equivalently, we can compute all the elements of the vectors Θ_t and V_t in parallel:

$$V_t = \beta V_{t-1} + (1-\beta) \frac{\partial L}{\partial \Theta}(\Theta_{t-1}; x, y)$$
$$\Theta_t = \Theta_{t-1} - \lambda V_t$$

1.3 Nesterov Momentum

The only difference with Nesterov Momentum from regular Momentum is the values we use when evaluating the partial derivative. We evaluate not at the location of the previous Θ, but instead at the value of the previous Θ adjusted by the previous V, since that's our best guess at this point as to where we'll end up. (We'll be moving by a previous V amount anyway, so we evaluate as if we had made that movement). Note that we must *subtract* the previous V from the current Θ since we always move the parameters in a direction that is negative to the gradient. We evaluate at that best guess since it should be a more accurate picture of the actual gradient value.

$$
\begin{aligned}
V_t &= \beta V_{t-1} + (1-\beta)\frac{\partial L}{\partial \Theta_i}(\Theta_{t-1} - V_{t-1}; x, y) \\
\Theta_t &= \Theta_{t-1} - \lambda V_t
\end{aligned}
$$

1.4 Adagrad

With Adaptive Gradient, we step away from momentum and look at adjusting learning rates on a parameter-by-parameter basis. We define an overall max learning rate λ, and then calculate a learning rate for each timestep t, and parameter i:

$$
\begin{aligned}
(\lambda_t)_i &= \frac{\lambda}{\sqrt{\epsilon + \sum_{k=1}^{t}(\frac{\partial L}{\partial \Theta_i}(\Theta_k; x, y))^2}} \\
(\Theta_t)_i &= (\Theta_{t-1})_i - (\lambda_t)_i \frac{\partial L}{\partial \Theta_i}(\Theta_{t-1}; x, y)
\end{aligned}
$$

In the above formula, ϵ of 1 may be a good choice (it'll limit the resulting parameter-specific learning rate to be between 0 and λ). β is a hyperparameter (often around 0.9).

The denominator increases as there are many and/or large gradients. Thus, many and/or large gradients for a parameter reduce the learning rate for that parameter.

One disadvantage of Adagrad is that for a given parameter i, the sequence of learning rates $(\lambda_1)_i, (\lambda_2)_i, ...$ is monotonic decreasing. Thus, a parameter i can be doomed with a low learning rate even once it has paid its debt to society:)

1.5 RMSProp

With RMSProp, we adjust Adagrad to forget about old learning rates by using an exponential moving average of squared gradients rather than a sum-of-squared gradients for all timesteps.

We define E_t to represent the exponential moving squared gradient with a decay factor of γ (between 0 and 1):

$$
\begin{aligned}
(E_t)_i &= \gamma(E_{t-1})_i + (1-\gamma)\left(\frac{\partial L}{\partial \Theta_i}(\Theta_{t-1}; x, y)\right)^2 \\
(\lambda_t)_i &= \frac{\lambda}{\sqrt{\epsilon + (E_t)_i}} \\
(\Theta_t)_i &= (\Theta_{t-1})_i - (\lambda_t)_i \frac{\partial L}{\partial \Theta_i}(\Theta_{t-1}; x, y)
\end{aligned}
$$

γ is a hyperparameter (often around 0.9).

1.6 Adam

Adam (adapative moment estimation) is a combination of RMSProp with Momentum (with a slight twist where V_t and E_t are scaled to $\hat{V}_t$ and $\hat{E}_t$):

$$
\begin{aligned}
V_t &= \beta V_{t-1} + (1-\beta)\frac{\partial L}{\partial \Theta}(\Theta_{t-1}; x, y) \\
\hat{V}_t &= \frac{V_t}{1-\beta^t} \\
E_t &= \gamma E_{t-1} + (1-\gamma)\left(\frac{\partial L}{\partial \Theta}(\Theta_{t-1}; x, y)\right)^2 \\
\hat{E}_t &= \frac{E_t}{1-\gamma^t} \\
(\lambda_t)_i &= \frac{\lambda}{\sqrt{\epsilon + (\hat{E}_t)_i}} \\
(\Theta_t)_i &= (\Theta_{t-1})_i - (\lambda_t)_i (\hat{V}_t)_i
\end{aligned}
$$