ML and Optimization Lecture 15

- Momentum, Adaptive Learning Rates, and Preconditioning for First-Order Algorithm
- Kernel Machines & the Kernel Trick for nonlinear ML
Adaptive (Sub)gradient Methods

Recall advantages of first-order methods:

- scalable: much cheaper per-iteration than 2nd order methods, don't require solving linear systems, use less memory
- easy made stochastic to scale even more
- don't require strong convexity

Disadvantages:

- sensitive to hyperparameter choices (stepsize in particular)
- don't model curvature of the objective function (slower convergence than 2nd order methods)
Question: can we design first-order methods which are more insensitive to hyperparam choices & better model curvature?

Tools:
- momentum: keep moving in directions that I historically made progress in
- exponential averaging: forget history judiciously to balance between the current local model and the global information from history
- preconditioning: cheaply model curvature using just first order info
Today:
- SGD w/ momentum (heavy-ball method)
- Nesterov's Accelerated Gradient (NAG)
- AdaGrad (2011)
- RMSProp (2012)
- AdaDelta (2012)
- ADAM (2015)
\[ f(x) = \frac{1}{2} \|Dx\|_2^2 \]
\[ D = \begin{bmatrix} 1 \\ 10 \end{bmatrix} \]
\[ x^* = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \]
\[ \nabla f(x) = D^2 x = \begin{bmatrix} x_1 \\ 100 x_2 \end{bmatrix} \]

with gradient:
\[ x_{t+1} = x_t - \alpha D^2 x_t = \begin{bmatrix} x_{t,1} - \alpha x_{t,1} \\ x_{t,2} - 100 \alpha x_{t,2} \end{bmatrix} \]

The optimal learning rate is \( \alpha = \frac{1}{100} \) because regardless of where \( x_0, \Delta \) is, we have
\[ x_{1,2} = x_{0,2} - x_{0,2} = 0 = x^*_2 \]
Three lessons:

1) we see the usefulness of per-coordinate step-sizes

2) we see the potential of correcting for curvature as e.g. Newton's method would

3) we see the advantage of dampening the oscillations of the GrD procedure

Because GrD always moves II to the current level set, it oscillates.
**SGD w/ momentum (aka Polyak heavy-ball method)**

Idea is to dampen oscillations by continuing to move in historically useful directions.

Update direction: 
\[ \mu_t = \alpha_t \nabla f_t(w_t) + \gamma_t \mu_{t-1} \]

\[ w_{t+1} = w_t - \mu_t \]

where \( \nabla f_t \) is an estimate of \( \nabla f \) at iteration \( t \) (so this subsumes both SGD and SGGD).

Notice 
\[ w_{t+1} = w_t - \mu_t = w_t - \left[ \alpha_t \nabla f_t(w_t) + \gamma_t \mu_{t-1} \right] \]

\[ \mu_{t-1} = w_{t-1} - w_t \]

\[ \Rightarrow w_{t+1} = w_t - \alpha_t \nabla f_t(w_t) + \gamma_t (w_t - w_{t-1}) \]

— usual (S)GD

keep moving in the prev. dir
Notice that the update direction at time $t$ is an exponentially weighted average of the historical gradients.

E.g. assume $\gamma_0 = 0$ and $\gamma_t = \gamma = 0.9$ and $\alpha_t = \alpha$ then

$$w_{t+1} = w_0 - \alpha \sum_{k=0}^{t} \gamma^k \nabla f_{t-k}(w_{t-k})$$

Theoretical advantage of GD+momentum over GD is that its convergence rate is linear w/ rate determined by $\sqrt{k}$ vs $k$ for GD.
Nesterov’s Accelerated Gradient Descent (NAG)

\[ \mu_t = \delta_t \mu_{t-1} + \alpha_t \nabla f_t (w_t - \delta_t \mu_{t-1}) \]

\[ w_{t+1} = w_t - \mu_t \]

NAG = predictor-corrector method

conv. rate depends on \( \sqrt{K} \)
Preconditioning

\[ w^* = \operatorname{argmin}_w f(w) \]

introduce invertible matrix \( R \) and consider

\[ u^* = \operatorname{argmin}_u f(Ru) \]

Then \( w^* = Ru^* \)

Let \( g(u) = f(Ru) \), then \( \nabla g(u) = R^T \nabla f(Ru) \)

and \( \nabla^2 g(u) = R^T \nabla^2 f(Ru) R \)

so we can take \( R = \left[ \nabla^2 f(Ru) \right]^{-\frac{1}{2}} \), then

\[ \nabla^2 g(u) = I \]

and we have a very well-conditioned optim prob!
So we can solve

$$u^* = \arg\min_u f(R_0)$$

very efficiently via

$$u_{t+1} = u_t - \alpha_t R^T \nabla f(R_0)$$

Let $$w_{t+1} = R u_{t+1}$$

$$\Rightarrow w_{t+1} = R u_{t+1} = R u_t - \alpha_t R R^T \nabla f(R_0)$$

$$= w_t - \alpha_t R R^T \nabla f(R_0)$$

so if $$R = \left[ \nabla^2 f(w_t) \right]^{-1/2}$$ then this is Newton's method.

In practice, use diagonal preconditioners.
AdaGrad "Adaptive Gradient Descent" (diagonal version of preconditioner)

- learn different step sizes for each feature
- useful when loss surface has different scales with respect to each feature
- or useful when dealing with sparse data where some features appear infrequently

\[ \omega_{t+1} = \omega_t - \alpha \left( D_t + \varepsilon I \right)^{-1/2} \nabla f(\omega_t) \]

where
\[ D_t = \text{diag} \left( \sum_{i=1}^{t} \nabla f_i(\omega_i) \nabla f_i(\omega_i)^T \right) \]

\[ \varepsilon \approx 10^{-8} \]
Notice this corresponds to

\[
(\omega_{t+1})_i = (\omega_t)_i - \alpha \frac{\left( \sum_{j=1}^{t} \left[ \nabla f_i(\omega_j) \right]_i \right)^2}{\left( \sum_{j=1}^{t} \left[ \nabla f_i(\omega_j) \right]_i + \varepsilon \right)^{1/2}}
\]

Two phenomena occur with adagrad:

- all coordinates’ learning rates decrease with time because \( \sum_{j=1}^{t} \left[ \nabla f_i(\omega_j) \right]_i \) increases with \( t \)

- the coordinates that are seen more or changed more have learning rates that go to zero faster

This is relatively insensitive to \( \alpha \), but has the drawback that the learning rates go to zero regardless of how far we are from convergence.
RMSProp (2012) attempted to fix “prob” of Adagrad’s inevitable learning rate decay. 

\[ D_t = \beta D_{t-1} + (1-\beta) \text{diag} \left( \nabla f_t(\omega_t) \nabla f_t(\omega_t)^T \right) \]

vs (for Adagrad)

\[ D_t = D_{t+1} + \text{diag} \left( \nabla f_t(\omega_t) \nabla f_t(\omega_t)^T \right) \]

The update is the same:

\[ \omega_{t+1} = \omega_t - \alpha_t (D_t + \epsilon I)^{-1/2} \nabla f_t(\omega_t) \]
ADAM (2014) very popular in the DL community - tries to fix decreasing step size prob of Adagrad - also tries to estimate a better search direction using momentum (exponential averaging of gradients)

\[ \mu_t = \beta_1 \mu_{t-1} + (1 - \beta_1) \nabla f_t(\omega_t) \]
\[ D_t = \beta_2 D_{t-1} + (1 - \beta_2) \text{diag}(\nabla f_t(\omega_t) \nabla f_t(\omega_t)^T) \]
\[ \omega_{t+1} = \omega_t - \alpha_t (D_t + \epsilon I)^{-\frac{1}{2}} \mu_t \]