Adaptive (Sub)gradient methods

"gradient-based"

Recall the advantage of first-order methods:

- scalable, easily made stochastic
- don't require a lot of memory or solving large linear systems

Disadvantages:

- sensitive to hyperparameter choices (stepsize)
- doesn't model the curvature of the objective function (slower convergence than 2nd order methods)

Question:

- can we have first order methods that are insensitive to hyperparam choices & better model curvature?

Today:

- Adagrad (2011)
- RMSprop (2012)
- AdaDelta (2012)
- ADAM (2015)

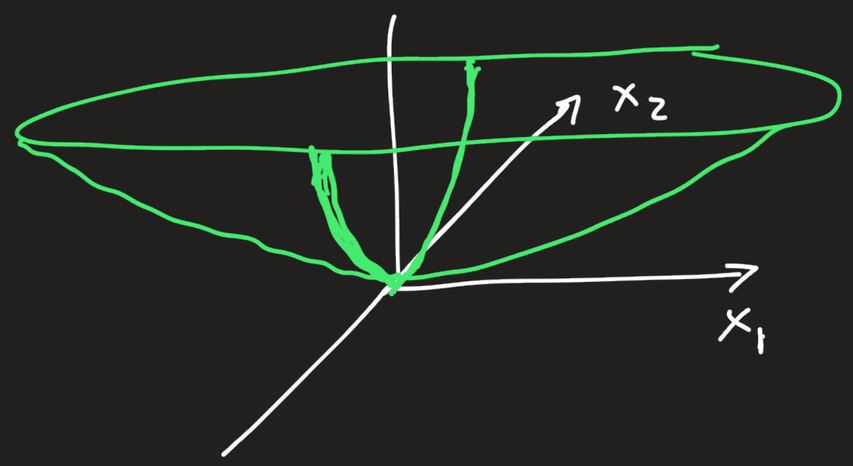
- SGD w/ momentum (heavy-ball method)
- Nesterov's Accelerated Gradient (NAG)

The ideas:

- use historical information to better model the function
- estimate curvature using first-order information
- handle different update rates for diff features

Ex: $f(x) = \frac{1}{2}\|Dx\|_2^2 \qquad D = \begin{bmatrix} 1 & \\ & 10 \end{bmatrix}$

$x^* = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$



$f(x) = \frac{1}{2}(x_1^2 + 10x_2^2)$

with gradient descent:

$$\nabla f(x) = Dx = \begin{bmatrix} x_1 \\ 100x_2 \end{bmatrix}$$

$$x_{t+1} = x_t - \alpha D x_t = \begin{bmatrix} x_{t,1} - \alpha x_{t,1} \\ x_{t,2} - 100\alpha x_{t,2} \end{bmatrix}$$

The optimal learning rate for $x_{t,2}$ is $\alpha = \frac{1}{100}$ > because regardless of what $x_{0,2}$ is, we have

$$x_{1,2} = x_{0,2} - x_{0,2} = 0 = x_2^*$$

OTOH, the optimal learning rate for $x_{t,1}$ is $\alpha = 1$ because then

$$x_{1,1} = x_{0,1} - x_{0,1} = 0 = x^*_1$$

These two features have <u>different</u> optimal learning rates, so no shared learning rate will be superior to assigning them different learning rates.

<u>Fact</u> Newton's exactly accounts for the curvature so it adapts the stepsizes for each feature individually:

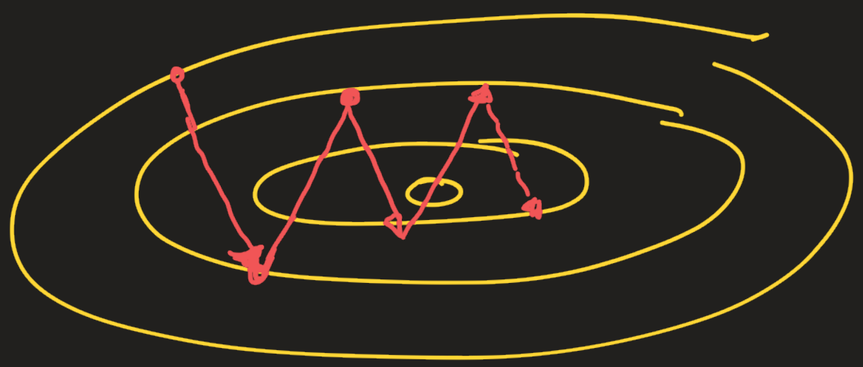
$$x_{t+1} = x_t - \alpha_t \left[\nabla^2 f(x_t)\right]^{-1} \nabla f(x_t)$$

$$\nabla^2 f(x_t) = D^2 = \begin{bmatrix} 1 & \\ & 100 \end{bmatrix} \Rightarrow x_{t+1} = x_t - \begin{bmatrix} \alpha_t & \\ & \alpha_t/100 \end{bmatrix} \nabla f(x_t)$$

$$= x_t - \begin{bmatrix} \alpha_t & \\ & \alpha_t/100 \end{bmatrix} \begin{bmatrix} 1 & \\ & 100 \end{bmatrix} x_t = 0$$

so Newton's method converges in one step when $\alpha_t = 1$

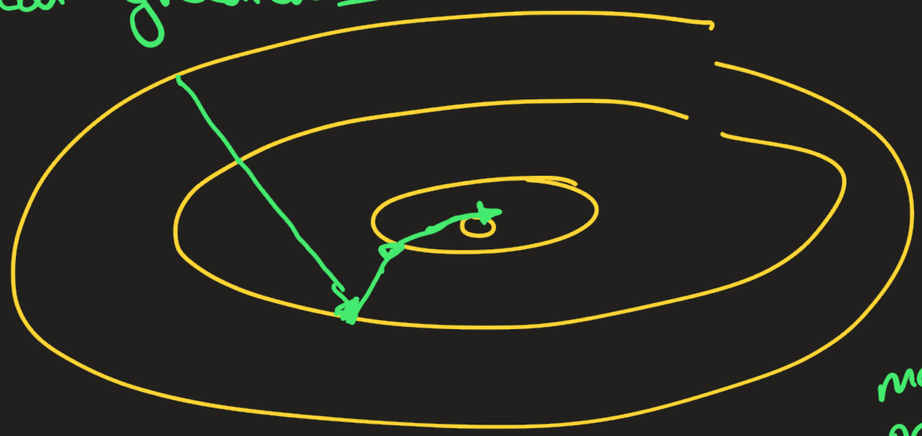
SGD w/ momentum



gradient descent oscillates in this case because it always moves perpendicular to the level set at the current point.

level set $\mathcal{L}_{f,\alpha} = \{x : f(x) = \alpha\}$

momentum reduces oscillatory behavior in recurring directions by averaging out historical gradients



SGD w/ momentum

momentum param

update direction

$$\mu_t = \alpha_t \nabla f_t(\omega_t) + \gamma_t \mu_{t-1}$$

$$\omega_{t+1} = \omega_t - \mu_t$$

typically take $\gamma_t = \gamma = 0.9$

here ∇f_t is an estimate of ∇f at iteration t.

called Polyak heavy-ball method and can write these updates as

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

and note

$$w_t = w_{t-1} - \mu_{t-1} \Rightarrow \mu_{t-1} = w_{t-1} - w_t$$

plug in to get

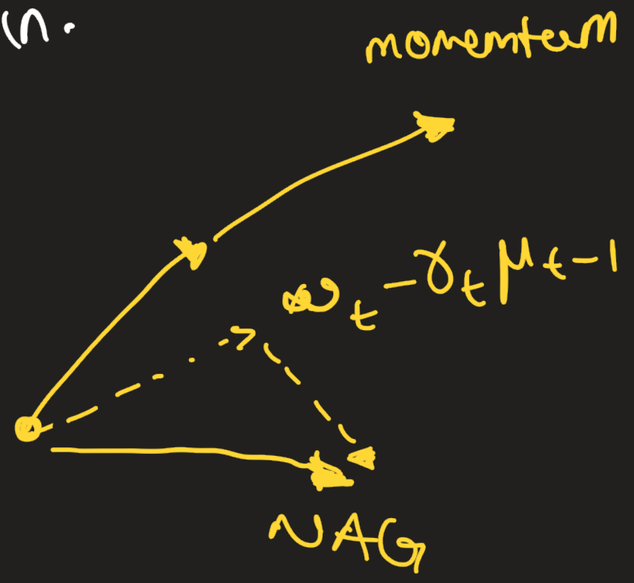
$$\underline{w_{t+1} = w_t - \alpha_t \nabla f_t(w_t) - \gamma_t (w_{t-1} - w_t)}$$

Nesterov's Accelerated Gradient Descent (NAG)

$$\mu_t = \gamma_t \mu_{t-1} + \alpha_t \nabla f_t(w_t - \gamma_t \mu_{t-1})$$

$$w_{t+1} = w_t - \mu_t$$

instead of trying to use just past information, look ahead along the current trend (moving in the μ_{t-1} direction) to choose an even better direction to move in.

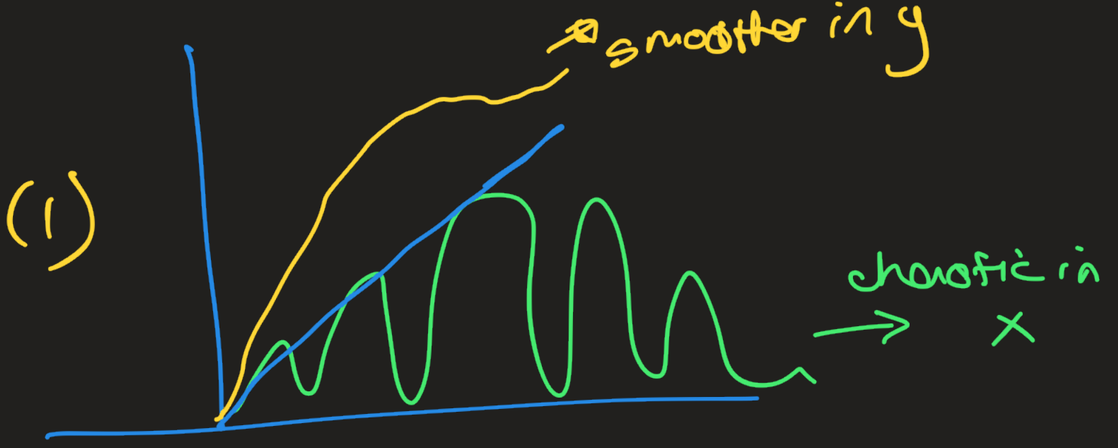


AdaGrad ("Adaptive Gradient Descent") diagonal version

- learn different stepsizes for each feature

(1) - useful when loss surface has different scales with respect to each feature

(2) - or, when dealing with sparse data where some features appear infrequently.



(2) Ex: consider a NLP application: fit classifier on bag of word features (BOW) to determine whether a review is positive

'good' 'bad' 'enjoy' 'sublime'

$x = [\,0 \;\; 2 \;\; 0 \cdots 0 \cdots 0 \;\; 1\,]$

$|V|$ dimension vector where V is our vocab.

← say x_{10K}

then, e.g. 'sublime' is very discriminative but seen infrequently, so want to update w_{10K} more aggressively when it is seen than say w_1 (corresp to 'good')

Idea: use per-feature learning rates that are inversely proportional to how much that feature has changed before.

$\varepsilon \approx 10^{-8}$

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

where

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

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

where $i \in [d]$ is a feature index

Advantages:

- relatively insensitive to α compared to SGD
- works impressively for sparse data

Drawbacks:

- the learning rate goes to zero with time because

$$\sum_{j=1}^{t} \nabla f(\omega_j)_i^2 \to \infty \quad \text{as} \quad t \to \infty$$

regardless of whether you're close to convergence or not.

- you need to store a learning rate for every parameter.