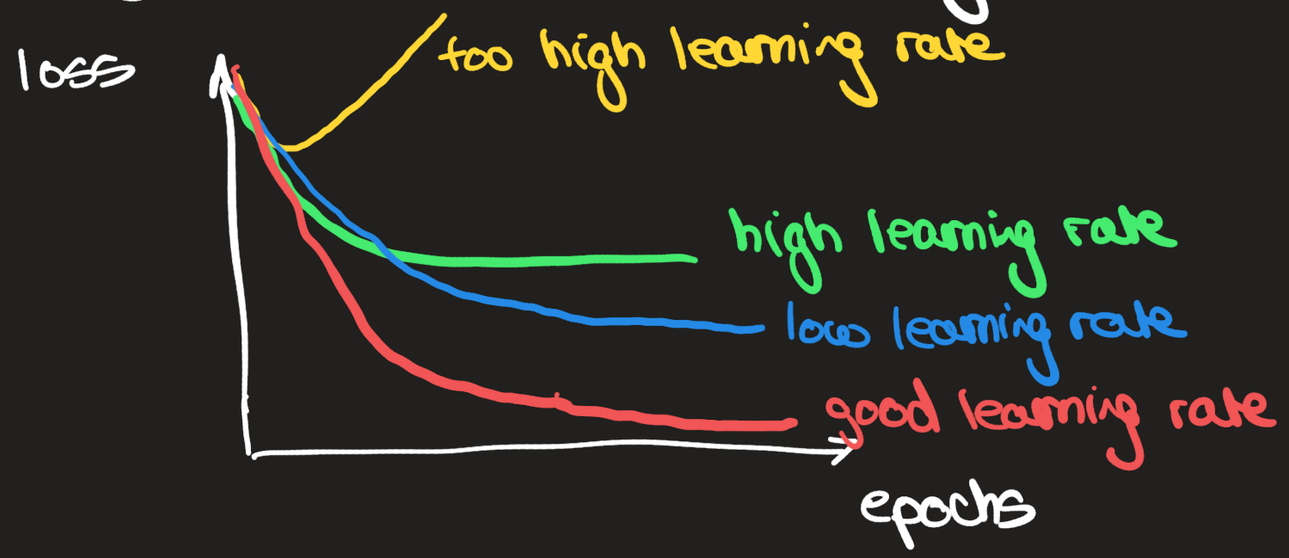
ML and Optimization Lecture 13

- Tips for tuning stepsizes for SGD
- Adaptive Gradient Descent:
  - momentum, exponential averaging, preconditioning
  - SGD w/ momentum
  - Nesterov's accelerated gradient descent
  - AdaGrad
  - RMSProp
  - ADAM

Tips & Tricks for SGD (Bottou 2012)

- How to choose stepsizes, in practice:
  - use a stepsize schedule of the form $\alpha_t = \frac{\beta_0}{1+\beta_1 t}$ where β_0 and β_1 should be optimized
  - use a small subset of your training data to test the convergence behavior (look at the model accuracy, not the training loss)



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
- **easily randomizable** very easy to replace gradient w/ a randomized estimate to further reduce the cost per iteration
- **generally applicable** does not require differentiability or strong convexity

Disadvantages:

- sensitive to hyperparameter choices (stepsize in particular)
- do not model curvature of the objective (slower convergence than 2nd order methods)

Question: can we design first-order methods which are more insensitive to the hyperparameter choices and better model curvature?

Tools:

- momentum : keep moving in directions that historically made progress
- exponential averaging: forget history judiciously to balance between the current local model and the global information from history
- preconditioning : cheaply modeling curvature using just first order information

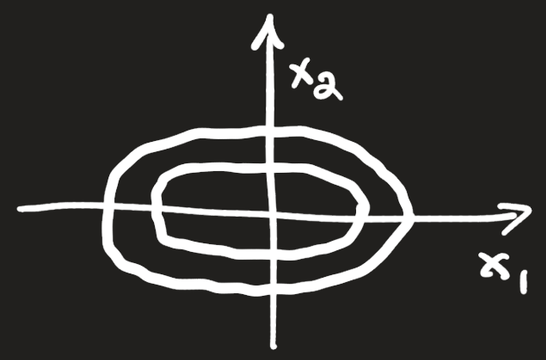
Adaptive GD methods

1964 – SGD w/ momentum (Polyak's heavy ball method)
1983 – Nesterov's accelerated gradient descent (NAG)
2014 – AdaGrad
2012 – RMSProp
2012 – AdaDelta
2015 – ADAM

Ex

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

$$= \frac{1}{2}x_1^2 + 5x_2^2$$



$$x_* = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad \nabla f(x) = D^2 x = \begin{bmatrix} x_1 \\ 100x_2 \end{bmatrix}$$

with GD:

$$x_{t+1} = x_t - \alpha D^2 x_t = (I - \alpha D^2)x_t = \cdots = (I - \alpha D^2)^{t+1} x_0$$

$$= \begin{bmatrix} (1-\alpha)^{t+1} & \\ & (1-\alpha 100)^{t+1} \end{bmatrix} \begin{bmatrix} x_{0,1} \\ x_{0,2} \end{bmatrix}$$

The learning from theory that guarantees linear convergence is $\alpha = \frac{1}{100}$

Note this is controlled by the coordinate w/ the highest curvature

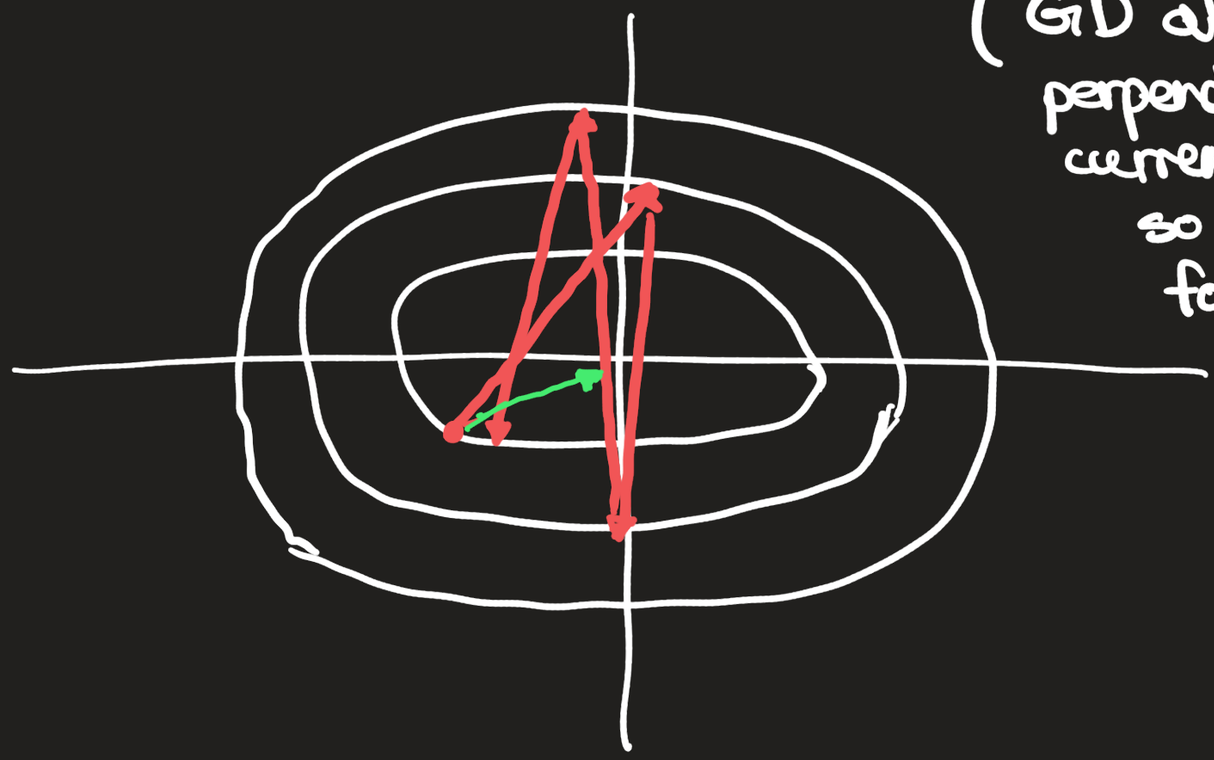
$$\Rightarrow x_t = \begin{bmatrix} (99/100)^t & \\ & 0 \end{bmatrix} \begin{bmatrix} x_{0,1} \\ x_{0,2} \end{bmatrix} \quad \text{so } x_t \to x^* \text{ very slowly}$$

Three lessons:

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

2) We see the importance of accounting for curvatures

3) We see the advantage of dampening the oscillations of GD



(GD always moves perpendicular to the current level set, so it oscillates for this problem)

SGD w/ momentum

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

update direction: $\mu_t = \alpha_t \nabla f_t(x_t) + \gamma_t \mu_{t-1}$

parameter update: $x_{t+1} = x_t - \mu_t$

historical update direction

momentum parameter (0.9)

where ∇f_t is an estimate of ∇f at iteration t
(so this subsumes both GD and SGD w/ momentum)

Notice that this is equivalent to

$$x_{t+1} = \underbrace{x_t - \alpha_t \nabla f_t(x_t)}_{\text{update suggested by local model (GD or SGD)}} + \underbrace{\gamma_t (x_t - x_{t-1})}_{\text{update suggested by history: keep moving in the previous direction}}$$

Notice that the update direction at time t is an exponentially weighted sum of the historical gradients

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

$$x_{t+1} = x_0 - \alpha \sum_{k=0}^{t} \gamma^k \nabla f_{t-k}(x_{t-k})$$

Theoretical advantage of GD + momentum over GD is that its local convergence rate is linear w/ rate determined by $\sqrt{\kappa}$ vs κ for GD (if we choose the optimal step size

$$\alpha = \frac{4}{(\sqrt{\beta} + \sqrt{\mu})^2}$$

(μ: strong convexity parameter)

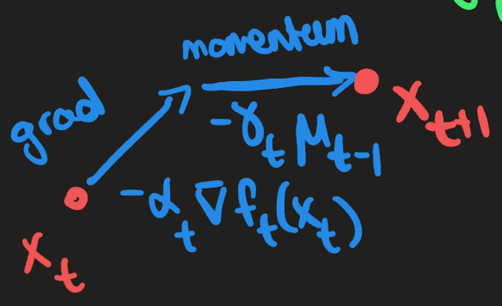
and momentum $\gamma = \left(\frac{\sqrt{\beta} - \sqrt{\mu}}{\sqrt{\beta} + \sqrt{\mu}} \right)^2$)

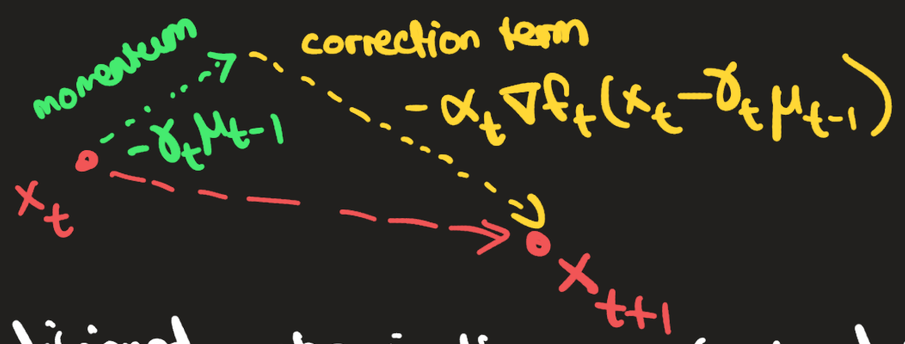
Nesterov's Accelerated Gradient Descent (NAG)

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

$$x_{t+1} = x_t - \mu_t$$

$$= \underbrace{(x_t - \gamma_t \mu_{t-1})}_{\text{predict where to go using historical information}} - \underbrace{\alpha_t \nabla f_t(x_t - \gamma_t \mu_{t-1})}_{\text{correct yourself w/ the local model at that point}}$$





conv rate for well-conditioned probs is linear w/ rate determined by $\sqrt{K}$

conv rate for β-smooth $O\left(\frac{1}{T^2}\right)$

Preconditioning

Imagine we want to solve $x^* = \operatorname{argmin} f(x)$

Introduce a change of variables $x = Ru$ where R is invertible

$$u^* = \operatorname{argmin} f(Ru) \iff x^* = Ru^*$$

We can solve in the u-space using GD via

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

It follows that

$$x_{t+1} = Ru_{t+1} = R(u_t - \alpha_t R^T \nabla f_t(Ru_t))$$

$$= x_t - \alpha_t RR^T \nabla f_t(x_t)$$

so if $R = [\nabla^2 f(x_t)]^{-1/2}$ then we get Newton's method.

In practice, use <u>diagonal</u> preconditioners.

AdaGrad "Adaptive Gradient Descent" (diagonal version)

- learns different stepsizes for each coordinate
- useful when the objective has different scales w.r.t. each feature

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

where

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

$$= \begin{bmatrix} \sum_{i=1}^{t} \left(\frac{\partial f_i(x_i)}{\partial x_1} \right)^2 & & \\ & \ddots & \\ & & \sum_{i=1}^{t} \left(\frac{\partial f_i}{\partial x_d}(x_i) \right)^2 \end{bmatrix}$$

← measures how large, historically, the components of the gradient corresponding to each coordinate has been

Per coordinate update:

$$(x_{t+1})_i = (x_t)_i - \frac{\alpha}{\left(\sum_{j=1}^{t} [\nabla f_j(x_j)]_i^2 + \varepsilon\right)^{1/2}} [\nabla f_t(x_t)]_i$$

Two phenomena occur:

- all coordinates' learning rates decrease with time
- coordinates that changed more (their gradients were large) have learning rates that go to zero faster

RMSProp (2012)

attempted to fix the problem of AdaGrad's inevitable learning rate decay

$$D_t = \beta D_{t-1} + (1-\beta)\,\mathrm{diag}\left(\nabla f_t(x_t)\nabla f_t(x_t)^T\right)$$

vs

$$D_t = D_{t-1} + \mathrm{diag}\left(\nabla f_t(x_t)\nabla f_t(x_t)^T\right)$$

The update remains the same as for AdaGrad

$$x_{t+1} = x_t - \alpha\,(D_t + \epsilon I)^{-1/2}\,\nabla f_t(x_t)$$

ADAM (2014)

very popular in the ML community

- tries to fix the learning rate decay issue of AdaGrad
- also tries to estimate a better searchdirection using momentum

$$\mu_t = \beta_1 \mu_{t-1} + (1-\beta_1)\nabla f_t(x_t)$$

$$D_t = \beta_2 D_{t-1} + (1-\beta_2)\,\mathrm{diag}\left(\nabla f_t(x_t)\nabla f_t(x_t)^T\right)$$

$$x_{t+1} = x_t - \alpha\,(D_t + \epsilon I)^{-1/2}\,\mu_t$$