Gradient Descent Methods

- basic ingredients of ML:

1) data: $\underline{X}$
2) model: $\theta \longmapsto g_\theta$ $\quad\theta$: model parameters e.g. weights, biases (trainable)
3) cost function:

$$\underline{X}, g, \theta \longmapsto E(\underline{X}, g_\theta)$$

evaluates how well model works

e.g. $\underline{X} = \{(x_i, y_i)\}_{j=1}^{n}$, n data points (e.g. generated from $x \mapsto f(x) = y$ unknown)

$g_\theta : x_j \longmapsto g_\theta(x_j)$ some parametrized fn.

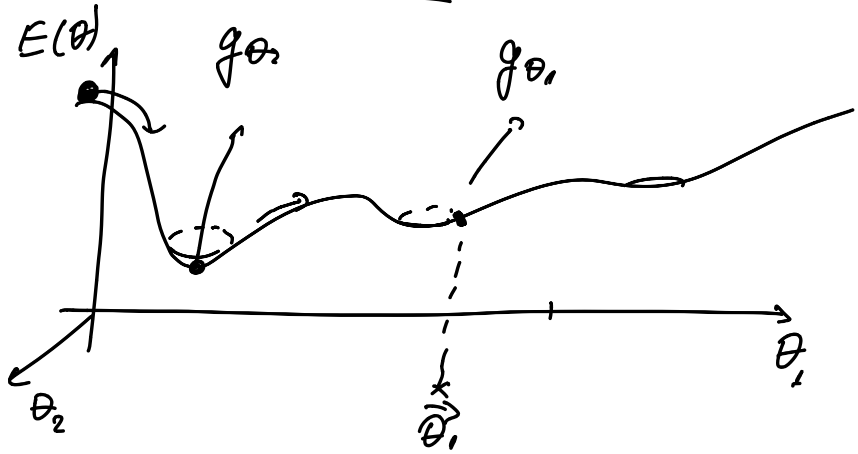
e.g. $g_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$

or $g_\theta(x) = NN_\theta(x)$

$$E : \underline{X}, g_\theta \longmapsto \frac{1}{n}\sum_{j=1}^{n}\left[y_j - g_\theta(x_j)\right]^2$$

- how do we train the model?, i.e. how do we find optimal values of params. θ?
  → by minimizing cost function E wrt. parameters θ.

→ use <u>Gradient</u> <u>descent</u> methods.

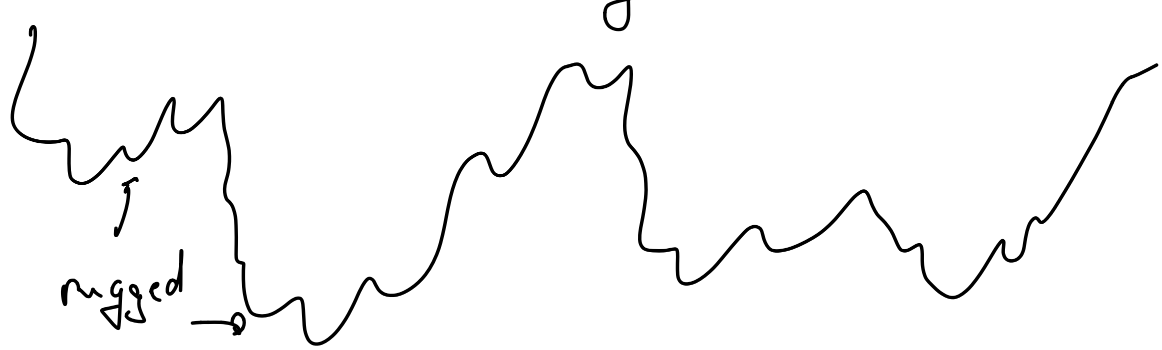


optimization landscape

<u>basic idea</u>: iteratively adjust params θ in directions where gradient of cost fn $\nabla_\theta E$ is large /negative

→ params θ "flow" towards a local minimum of cost fn / optimization landscape

– <u>in ML</u>: i) cost fn's are rugged, non-convex landscapes in high-dim. space with many local minima



ii) don't have access over true cost function we want to minimize
→ data is corrupt wise, uncertainty

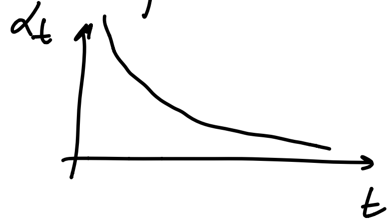
→ <u>gradient descent (GD)</u>

initialize: θ_0 at random

want: new parameters θ_1 from old θ_0

$$\underbrace{\theta_{t+1}}_{\substack{\text{new}\\ \text{params}}} = \underbrace{\theta_t}_{\substack{\text{old}\\ \text{params}}} - \underbrace{\alpha_t}_{\substack{\text{learning}\\ \text{rate}}} \underbrace{\nabla_\theta E(\theta_t)}_{\substack{\text{gradient of}\\ \text{cost fn.}}}$$

step size



→ re-write as

$$\rho_t := \alpha_t \nabla_\theta E(\theta_t)$$

$$\theta_{t+1} = \theta_t - \rho_t$$

<u>Newton's Method</u>

$$E(\theta - \rho) \approx E(\theta) - \rho \cdot \underbrace{\nabla_\theta E}_{\text{gradient}} + \frac{1}{2} \rho^t \underbrace{H(\theta)}_{\text{Hessian matrix}} \rho \quad (*)$$

$\approx$: expand for small ρ

$$H_{mn} = \partial_{\theta_m} \partial_{\theta_n} E(\theta)$$

assume $\theta - \rho_*$ is an optimum of $E(\theta)$

$$\nabla_\theta E(\theta - \rho_*) = 0$$

take ∇_ρ of $(*)$ at $\rho = \theta - \rho_*$

$\Rightarrow$

$$0 = \nabla_\rho E \Big|_{\theta - \rho_*} \approx - \nabla_\theta E + H(\theta)\, \rho_*$$

$$\Rightarrow \rho_* = H^{-1}(\theta)\, \nabla_\theta E(\theta)$$

$\Rightarrow$ <u>Newton method update rule</u> :

$$\rho_t = H^{-1}(\theta_t)\, \nabla_\theta E(\theta_t)$$

$$\theta_{t+1} = \theta_t - \rho_t$$

<u>problems</u>: 1) H may not be invertible

fix $\rightarrow$ regularize: $H^{-1} \rightarrow (H + \varepsilon \mathbb{1})^{-1}$

ε small regularizer

2) computing inverse H^{-1} extremely expensive when # params is large, e.g. $\theta \in \mathbb{R}^{10^7}$

- what do we learn from Newton's method about GD?

. in GD $\alpha_t = \text{const}$ $\nearrow$ in time t / $\rightarrow$ for all params θ

in Newton: $\alpha_t = H^{-1}$ is

$\rightarrow$ θ-dependent, couples different θ

recall: $x \mapsto f(x) = a x^2$ 

$\Rightarrow f'' = a$; a^{-1} defines curvature at the extremum

H is generalization of second derivative

$\rightarrow$ inverse eigenvalue define curvature in direction given by corresponding eigenvector

$\rightarrow$ Newton's method adjusts the learning rate α_t proportional to local curvature of $E(\theta)$:

- take larger steps in flat directions (small curvature)
- take small steps in steep directions (high curvature)

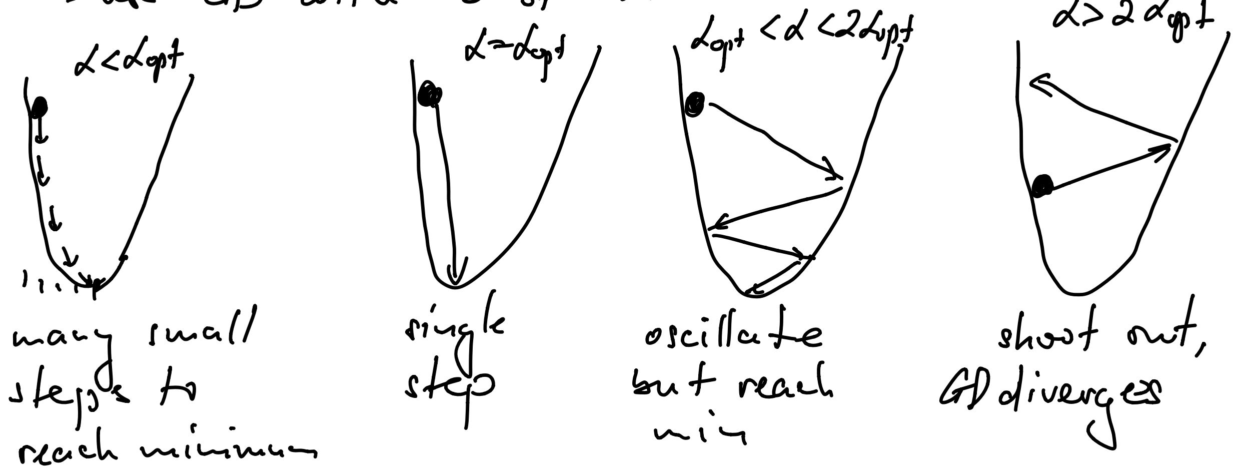
Example: optimal learning rate:

$E(\theta) = \theta^2$ is quadratic, $\theta \in \mathbb{R}$

$$E(\theta - \rho) = E(\theta) - \rho\, \partial_\theta E(\theta) + \tfrac{1}{2} \rho^2 \partial_\theta^2 E \quad \text{exact!}$$

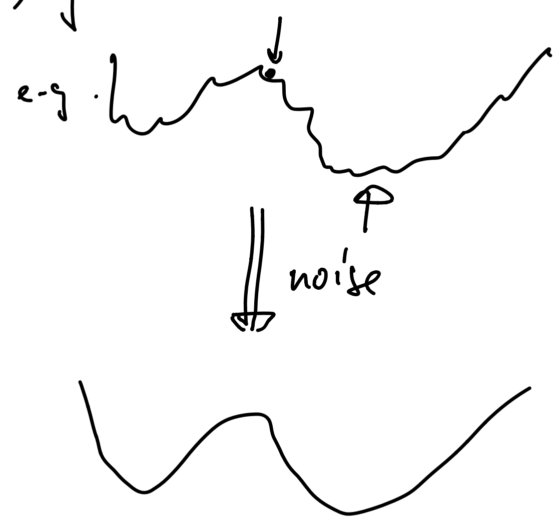
$\Rightarrow$ optimal learning rate $\alpha_{opt} = (\partial_\theta^2 E)^{-1}$

→ use GD with const α :



— Some <u>disadvantages of GD</u>

1) gets stuck in local minima :



suggestion: introduce small noise to smear out cost function barriers

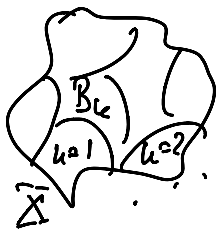
→ upgrade GD to stochastic GD (SGD)

2) gradients are computationally expensive in large data sets $\bar{X}$

→ do we need all data at every step t?

⇒ use subsets of the data, called <u>minibatches</u>

e.g. $E(\theta) \sim \sum_{j=1}^{n} \left[y_j - g_\theta(x_j)\right]^2$



$$= \sum_{k=1}^{M} \sum_{j \in B_k} \left[y_j - g_\theta(x_j)\right]^2$$

M minibatches of size $|B_k|$

→ drop sum over k

$$E(\theta) \approx \sum_{j \in B_k} \left[y_j - g_\theta(x_j)\right]^2$$

→ B_k are determined randomly every step t

⇒ <u>SGD</u>

3) GD is sensitive to choice learning rate α_t

→ consider an adaptive schedule for α_t (Runge-Kutta methods)

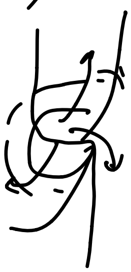
4) GD weighs all directions in parameter space uniformly:

→ make α_t θ-dependent (curvature of landscape)

(⇒ RMSProp, ADAM)

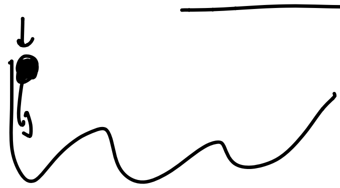
5) GD is sensitive to initial condition θ_0
(true for alternative methods)

6) GD can get stuck in saddle points



saddles are more common than minima/maxima in higher-dim space

Gradient Descent with Momentum:



- add memory of the direction (in param space) we are coming from:

$$\rho_t = \gamma \rho_{t-1} + \alpha_t \nabla_\theta E(\theta_t)$$

$$\theta_{t+1} = \theta_t - \rho_t$$

$\gamma \in [0,1]$: momentum parameter

- interpret ρ_t as a running average of recently encountered gradients

- analogy with a viscous particle in potential energy $E(z)$ described by $z(t)$:

$$m \frac{d^2 z}{dt^2} + \mu \frac{dz}{dt} = - \nabla_z E(z)$$

→ use finite differences

$$\frac{dz}{dt} \approx \frac{z_{t+\Delta t} - z_t}{\Delta t}$$

$$m \frac{\overbrace{z_{t+\Delta t} - z_t}^{= \Delta z_{t+\Delta t}} - z_t + z_{t-\Delta t}}{(\Delta t)^2} + \mu \frac{z_{t+\Delta t} - z_t}{\Delta t} = - \nabla_z E$$

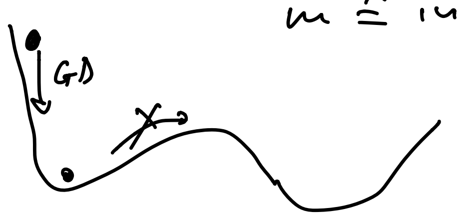
$$m \frac{\Delta z_{t+\Delta t}}{(\Delta t)^2} - m \frac{\Delta z_t}{(\Delta t)^2} + \mu \frac{\Delta z_{t+\Delta t}}{\Delta t} = - \nabla_z E$$

$$\Rightarrow \Delta z_{t+\Delta t} = \frac{m}{m + \mu \Delta t} \Delta z_t - \frac{(\Delta t)^2}{m + \mu \Delta t} \nabla_z E$$

identify: $\underline{\gamma = \frac{m}{m + \mu \Delta t}}$; $\alpha = \frac{(\Delta t)^2}{m + \mu \Delta t}$

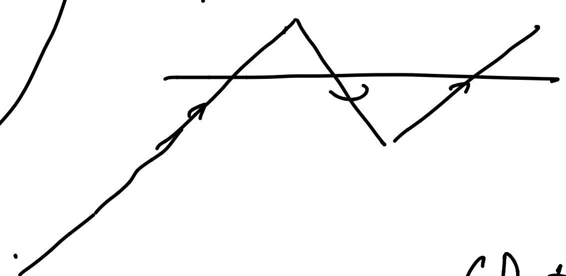
⇒ $\gamma \propto m$

$m \mathrel{\hat{=}}$ inertia → momentum

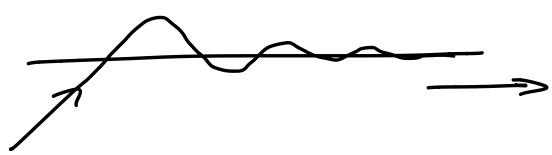


- why is momentum useful?
  - gain speed in directions with persistent small gradients
  - suppress oscillations in highly curved directions ($\gamma \to$ average over last few gradients)

top view: GD

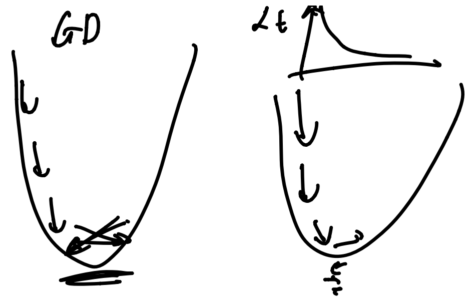


GD + momentum



- useful modification:
  Nesterov Accelerated Gradient (NAG)

$$\begin{cases} \rho_t = \gamma \rho_{t-1} + \alpha_t \nabla_\theta E(\theta_t + \gamma \rho_{t-1}) \\ \theta_{t+1} = \theta_t - \rho_t \end{cases}$$



$\gamma = 0 \to$ no mom.

→ kill effect of momentum at times meaningful?

GD methods with second-order moments

- recall: learning rate limited by curvature of optimization landscape / cost fn.

idea: adjust learning rate using Hessian

issue: H^{-1} infeasible to compute if our model g_θ has $\sim 10^4$ params (i.e. $\theta \in \mathbb{R}^{10^4}$)

can we achieve a similar goal w/o H?

Yes! → keep a running estimate of second moment of the gradient

RMSprop:

$$\begin{cases} g_t = \nabla_\theta E(\theta_t) \\ v_t = \beta v_{t-1} + (1-\beta) g_t^2 , \quad v_0 = 0 \\ \theta_{t+1} = \theta_t - \alpha_t \dfrac{g_t}{\sqrt{v_t + \varepsilon}} \end{cases}$$

- $\beta \in [0,1]$: controls averaging time of second moment of g_t
- ε : small regularizer ($\varepsilon \sim 10^{-7}$)
- α_t : learning rate schedule
- multiplication/division of two vectors is understood element-wise : $\frac{\vec{a}}{\vec{b}} = \overrightarrow{\left(\frac{a_i}{a_i}\right)}$

$\to$ why V_t is a running average?

$$V_t = \beta V_{t-1} + (1-\beta) g_t^2$$

$$= \beta \left(\beta V_{t-2} + (1-\beta) g_{t-1}^2 \right) + (1-\beta) g_t^2$$

$$= \dots$$

$$= (1-\beta) \sum_{j=1}^{t} \underbrace{\beta^{t-j}}_{\text{decay rate / (inv.) averaging time}} g_j^2$$

– suppose $E(\theta)$ is noisy (e.g. data X is noisy)

$\to$ can take an expectation over the noise $\mathbb{E}$

$$V_t = (1-\beta) \sum_{j=1}^{t} \beta^{j} g_j^2 \quad / \mathbb{E}[\cdot]$$

$$\mathbb{E}[V_t] = (1-\beta) \sum_{j=1}^{t} \beta^{j} \underbrace{\mathbb{E}[g_j^2]}_{\approx \mathbb{E}[g^2] \text{ indep. of } j}$$

$$\approx \mathbb{E}[g^2]\cancel{(1-\beta)} \underbrace{\sum_{j=1}^{t} \beta^j}_{\overset{\beta<1}{=} \frac{(1-\beta^t)}{\cancel{(1-\beta)}}} = \mathbb{E}[g^2]\underbrace{(1-\beta^t)}$$

creates bias at initial steps t

$$\beta^t \xrightarrow[t\to\infty]{} 0$$

<u>ADAM</u>: (bias-corrected RMSProp)

$$g_t = \nabla_\theta E(\theta)$$

$$\left.\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned}\right] \text{biased} \qquad \begin{aligned} &, \; m_0 = 0 \\ &, \; v_0 = 0 \end{aligned}$$

$$\left.\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \end{aligned}\right] \text{correct bias}$$

$$\theta_t = \theta_{t-1} - \alpha_t \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}}$$

β_1, β_2 : decay rates of first & second moment of the gradient estimate

$$\beta_j^t = \underbrace{\beta_j \cdots \beta_j}_{t\text{-times}}$$

ε : small regularizer

α_t : learning rate schedule

mult. /division : element-wise operations