ECE 595: Machine Learning I
Lecture 05 Gradient Descent

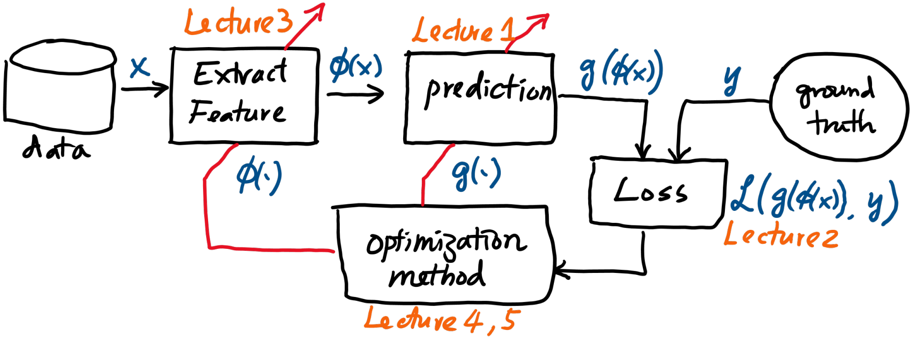
Spring 2020

Stanley Chan

School of Electrical and Computer Engineering
Purdue University

ⓒStanley Chan 2020. All Rights Reserved.

Outline



© Stanley Chan 2020. All Rights Reserved.

Outline

Mathematical Background

- Lecture 4: Intro to Optimization
- Lecture 5: Gradient Descent

Lecture 5: Gradient Descent

- Gradient Descent
  - Descent Direction
  - Step Size
  - Convergence
- Stochastic Gradient Descent
  - Difference between GD and SGD
  - Why does SGD work?

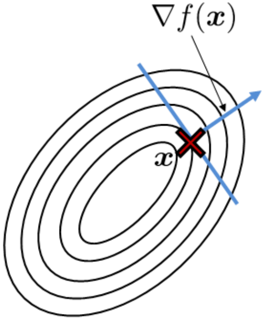
©Stanley Chan 2020. All Rights Reserved.

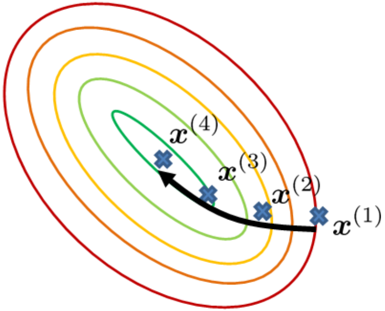
Gradient Descent

The algorithm:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha^{(t)} \nabla f(\mathbf{x}^{(t)}), \qquad t = 0, 1, 2, \ldots,$$

where $\alpha^{(t)}$ is called the **step size**.





©Stanley Chan 2020. All Rights Reserved.

Why is the direction $-\nabla f(x)$?

- Recall (Lecture 4): If $\boldsymbol{x}^*$ is optimal, then

$$\underbrace{\lim_{\epsilon \to 0} \frac{1}{\epsilon}\left[f(\boldsymbol{x}^* + \epsilon \boldsymbol{d}) - f(\boldsymbol{x}^*)\right]}_{\geq 0, \ \forall \boldsymbol{d}} = \nabla f(\boldsymbol{x}^*)^T \boldsymbol{d}$$

$$\Longrightarrow \qquad \nabla f(\boldsymbol{x}^*)^T \boldsymbol{d} \geq 0, \quad \forall \boldsymbol{d}$$

- But if $\boldsymbol{x}^{(t)}$ is not optimal, then we want

$$f(\boldsymbol{x}^{(t)} + \epsilon \boldsymbol{d}) \leq f(\boldsymbol{x}^{(t)})$$

- So,

$$\underbrace{\lim_{\epsilon \to 0} \frac{1}{\epsilon}\left[f(\boldsymbol{x}^{(t)} + \epsilon \boldsymbol{d}) - f(\boldsymbol{x}^{(t)})\right]}_{\leq 0, \ \text{for some } \boldsymbol{d}} = \nabla f(\boldsymbol{x}^{(t)})^T \boldsymbol{d}$$

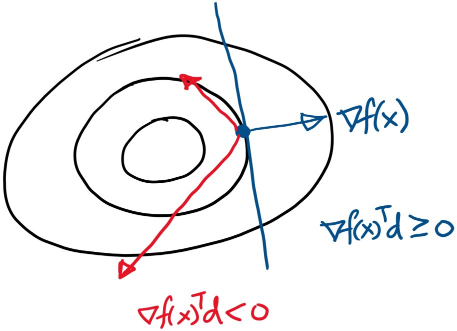
$$\Longrightarrow \qquad \nabla f(\boldsymbol{x}^{(t)})^T \boldsymbol{d} \leq 0$$

©Stanley Chan 2020. All Rights Reserved.

Descent Direction

Pictorial illustration:

- $\nabla f(\boldsymbol{x})$ is **perpendicular** to the contour.
- A search direction $\boldsymbol{d}$ can either be on the positive side $\nabla f(\boldsymbol{x})^T \boldsymbol{d} \geq 0$ or negative side $\nabla f(\boldsymbol{x})^T \boldsymbol{d} < 0$.
- Only those on the negative side can reduce the cost.
- All such $\boldsymbol{d}$'s are called the **descent directions**.



©Stanley Chan 2020. All Rights Reserved.

The Steepest $\boldsymbol{d}$

Previous slide: If $\boldsymbol{x}^{(t)}$ is not optimal yet, then some $\boldsymbol{d}$ will give

$$\nabla f(\boldsymbol{x}^{(t)})^T \boldsymbol{d} \leq 0.$$

- So, let us make $\nabla f(\boldsymbol{x}^{(t)})^T$ as negative as possible.

$$\boldsymbol{d}^{(t)} = \underset{\|\boldsymbol{d}\|_2=\delta}{\operatorname{argmin}} \; \nabla f(\boldsymbol{x}^{(t)})^T \boldsymbol{d},$$

- We need δ to control the magnitude; Otherwise $\boldsymbol{d}$ is unbounded.
- The solution is

$$\boldsymbol{d}^{(t)} = -\nabla f(\boldsymbol{x}^{(t)})$$

- Why? By Cauchy Schwarz,

$$\nabla f(\boldsymbol{x}^{(t)})^T \boldsymbol{d} \geq -\|\nabla f(\boldsymbol{x}^{(t)})\|_2 \|\boldsymbol{d}\|_2.$$

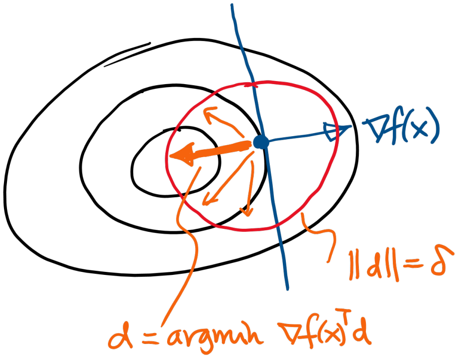
- Minimum attained when $\boldsymbol{d} = -\nabla f(\boldsymbol{x}^{(t)})$.
- Set $\delta = \|\nabla f(\boldsymbol{x}^{(t)})\|_2$.

©Stanley Chan 2020. All Rights Reserved.

Steepest Descent Direction

Pictorial illustration:

- Put a ball surrounding the current point.
- All $\boldsymbol{d}$'s inside the ball are feasible.
- Pick the one that minimizes $\nabla f(\boldsymbol{x})^T \boldsymbol{d}$.
- This direction must be parallel (but opposite sign) to $\nabla f(\boldsymbol{x})$.



© Stanley Chan 2020. All Rights Reserved.

Step Size

The algorithm:

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \alpha^{(t)}\nabla f(\boldsymbol{x}^{(t)}), \qquad t = 0, 1, 2, \ldots,$$

where $\alpha^{(t)}$ is called the step size.

- **1. Fixed step size**

$$\alpha^{(t)} = \alpha.$$

- **2. Exact line search**

$$\alpha^{(t)} = \underset{\alpha}{\text{argmin}} \;\; f\left(\boldsymbol{x}^{(t)} + \alpha \boldsymbol{d}^{(t)}\right),$$

  - E.g., if $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x}$, then

$$\alpha^{(t)} = -\frac{\nabla f(\boldsymbol{x}^{(t)})^T\boldsymbol{d}^{(t)}}{\boldsymbol{d}^{(t)T}\boldsymbol{H}\boldsymbol{d}^{(t)}}.$$

- **3. Inexact line search**:
  Amijo / Wolfe conditions. See Nocedal-Wright Chapter 3.1.

©Stanley Chan 2020. All Rights Reserved.

Convergence

Let $\boldsymbol{x}^*$ be the global minimizer. Assume the followings:

- Assume f is twice differentiable so that $\nabla^2 f$ exist.
- Assume $0 \preceq \lambda_{\min} \boldsymbol{I} \preceq \nabla^2 f(\boldsymbol{x}) \preceq \lambda_{\max} \boldsymbol{I}$ for all $\boldsymbol{x} \in \mathbb{R}^n$
- Run gradient descent with exact line search.

Then, (Nocedal-Wright Chapter 3, Theorem 3.3)

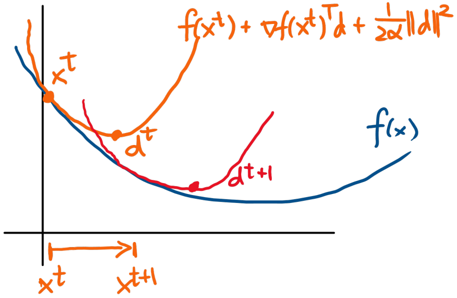
$$
\begin{aligned}
f(\boldsymbol{x}^{(t+1)}) - f(\boldsymbol{x}^*) &\leq \left(1 - \frac{\lambda_{\min}}{\lambda_{\max}}\right)^2 \left(f(\boldsymbol{x}^{(t)}) - f(\boldsymbol{x}^*)\right) \\
&\leq \left(1 - \frac{\lambda_{\min}}{\lambda_{\max}}\right)^4 \left(f(\boldsymbol{x}^{(t-1)}) - f(\boldsymbol{x}^*)\right) \\
&\leq \qquad \vdots \\
&\leq \left(1 - \frac{\lambda_{\min}}{\lambda_{\max}}\right)^{2t} \left(f(\boldsymbol{x}^{(1)}) - f(\boldsymbol{x}^*)\right).
\end{aligned}
$$

Thus, $f(\boldsymbol{x}^{(t)}) \to f(\boldsymbol{x}^*)$ as $t \to \infty$.

© Stanley Chan 2020. All Rights Reserved.

Understanding Convergence

- Gradient descent can be viewed as **successive approximation**.
- Approximate the function as
$$f(\boldsymbol{x}^t + \boldsymbol{d}) \approx f(\boldsymbol{x}^t) + \nabla f(\boldsymbol{x}^t)^T \boldsymbol{d} + \frac{1}{2\alpha}\|\boldsymbol{d}\|^2.$$
- We can show that the $\boldsymbol{d}$ that minimizes $f(\boldsymbol{x}^t + \boldsymbol{d})$ is $\boldsymbol{d} = -\alpha \nabla f(\boldsymbol{x}^t)$.
- This suggests: Use a **quadratic function** to locally approximate f.
- Converge when curvature α of the approximation is not too big.



©Stanley Chan 2020. All Rights Reserved.

Advice on Gradient Descent

- Gradient descent is useful because
    - Simple to implement (compared to ADMM, FISTA, etc)
    - Low computational cost per iteration (no matrix inversion)
    - Requires only first order derivative (no Hessian)
    - Gradient is available in deep networks (via back propagation)
- Most machine learning has built-in (stochastic) gradient descents
- Welcome to implement your own, but you need to be careful
    - Convex non-differentiable problems, e.g., ℓ_1-norm
    - Non-convex problem, e.g., ReLU in deep network
    - Trap by local minima
    - Inappropriate step size, a.k.a. learning rate
- Consider more "transparent" algorithms such as `CVX` when
    - Formulating problems. No need to worry about algorithm.
    - Trying to obtain insights.

©Stanley Chan 2020. All Rights Reserved.

Outline

©Stanley Chan 2020. All Rights Reserved.

Stochastic Gradient Descent

Most loss functions in machine learning problems are **separable**:

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}(g_{\boldsymbol{\theta}}(\boldsymbol{x}^n), y^n) = \frac{1}{N}\sum_{n=1}^{N} J_n(\boldsymbol{\theta}). \tag{1}$$

For example,

- Square-loss:

$$J(\boldsymbol{\theta}) = \sum_{n=1}^{N} (g_{\boldsymbol{\theta}}(\boldsymbol{x}^n) - y^n)^2$$

- Cross-entropy loss:

$$J(\boldsymbol{\theta}) = -\sum_{n=1}^{N} \left\{ y^n \log g_{\boldsymbol{\theta}}(\boldsymbol{x}^n) + (1-y^n)\log(1 - g_{\boldsymbol{\theta}}(\boldsymbol{x}^n)) \right\}$$

- Logistic loss:

$$J(\boldsymbol{\theta}) = \sum_{n=1}^{N} \log(1 + e^{-y^n \boldsymbol{\theta}^T \boldsymbol{x}^n})$$

©Stanley Chan 2020. All Rights Reserved.

Full Gradient VS Partial Gradient

Vanilla **gradient descent**:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} - \underbrace{\eta^t \nabla J(\boldsymbol{\theta}^t)}_{\text{main computation}}. \tag{2}$$

The full gradient of the loss is

$$\nabla J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N} \nabla J_n(\boldsymbol{\theta}) \tag{3}$$

Stochastic gradient descent:

$$\nabla J(\boldsymbol{\theta}) \approx \frac{1}{|\mathcal{B}|}\sum_{n\in\mathcal{B}} \nabla J_n(\boldsymbol{\theta}) \tag{4}$$

where $\mathcal{B} \subseteq \{1, \ldots, N\}$ is a random subset. $|\mathcal{B}|$ = batch size.

©Stanley Chan 2020. All Rights Reserved.

SGD Algorithm

Algorithm (Stochastic Gradient Descent)

1. *Given* $\{(\boldsymbol{x}^n, y^n) \mid n = 1, \ldots, N\}$.
2. *Initialize* $\boldsymbol{\theta}$ *(zero or random)*
3. *For* $t = 1, 2, 3, \ldots$
    - *Draw a random subset* $\mathcal{B} \subseteq \{1, \ldots, N\}$.
    - *Update*

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \eta^t \frac{1}{|\mathcal{B}|} \sum_{n \in \mathcal{B}} \nabla J_n(\boldsymbol{\theta}) \tag{5}$$

- If $|\mathcal{B}| = 1$, then use only one sample at a time.
- The approximate gradient is **unbiased**: (See Appendix for Proof)

$$\mathbb{E}\left[\frac{1}{|\mathcal{B}|} \sum_{n \in \mathcal{B}} \nabla J_n(\boldsymbol{\theta})\right] = \nabla J(\boldsymbol{\theta}).$$

©Stanley Chan 2020. All Rights Reserved.

Interpreting SGD

- Just showed that the SGD step is unbiased:

$$\mathbb{E}\left[\frac{1}{|\mathcal{B}|}\sum_{n\in\mathcal{B}} \nabla J_n(\boldsymbol{\theta})\right] = \nabla J(\boldsymbol{\theta}).$$

- Unbiased gradient implies that each update is

$$\text{gradient} \quad + \text{zero-mean noise}$$

- Step size: SGD with constant step size **does not converge**.
- If $\boldsymbol{\theta}^*$ is a minimizer, then $J(\boldsymbol{\theta}^*) = \frac{1}{N}\sum_{n=1}^{N} J_n(\boldsymbol{\theta}^*) = 0$. But

$$\frac{1}{|\mathcal{B}|}\sum_{n\in\mathcal{B}} J_n(\boldsymbol{\theta}^*) \neq 0, \qquad \text{since } \mathcal{B} \text{ is a subset.}$$

- Typical strategy: Start with large step size and gradually decrease: $\eta^t \to 0$, e.g., $\eta^t = t^{-a}$ for some constant a.

©Stanley Chan 2020. All Rights Reserved.

Perspectives of SGD

Classical optimization literature have the following observations.

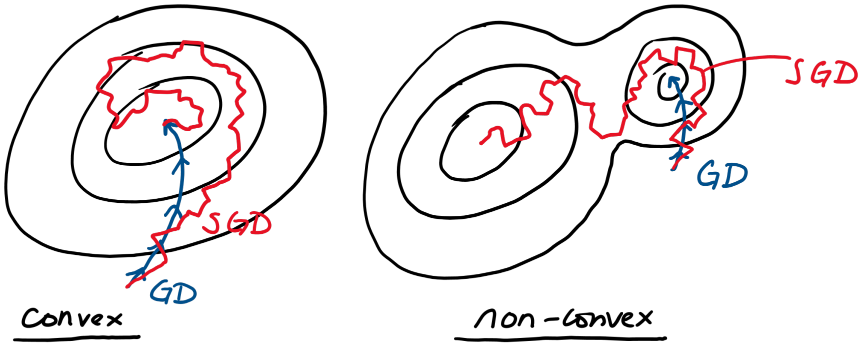
- Compared to GD in **convex** problems:
- SGD offers a **trade-off** between accuracy and efficiency
- More iterations
- Less gradient evaluation per iteration
- Noise is a by-product

Recent studies of SGD for **non-convex** problems found that

- SGD for training deep neural networks works
- SGD finds solution faster
- SGD find a better local minima
- Noise matters

©Stanley Chan 2020. All Rights Reserved.

GD compared to SGD



© Stanley Chan 2020. All Rights Reserved.

Smoothing the Landscape

Analyzing SGD is an active research topic. Here is one by Kleinberg et al. (`https://arxiv.org/pdf/1802.06175.pdf` ICML 2018)

- The SGD step can be written as GD + noise:

$$\begin{aligned} \boldsymbol{x}^{t+1} &= \boldsymbol{x}^t - \eta(\nabla f(\boldsymbol{x}^t) + \boldsymbol{w}^t) \\ &= \underbrace{\boldsymbol{x}^t - \eta\nabla f(\boldsymbol{x}^t)}_{\overset{\text{def}}{=}\boldsymbol{y}^t} - \eta\boldsymbol{w}^t. \end{aligned}$$

- $\boldsymbol{y}^t$ is the "ideal" location returned by GD.
- Let us analyze $\boldsymbol{y}^{t+1}$:

$$\begin{aligned} \boldsymbol{y}^{t+1} &\overset{\text{def}}{=} \boldsymbol{x}^{t+1} - \eta\nabla f(\boldsymbol{x}^{t+1}) \\ &= (\boldsymbol{y}^t - \eta\boldsymbol{w}^t) - \eta\nabla f(\boldsymbol{y}^t - \eta\boldsymbol{w}^t) \end{aligned}$$

- Assume $\mathbb{E}[\boldsymbol{w}] = 0$, then

$$\mathbb{E}[\boldsymbol{y}^{t+1}] = \boldsymbol{y}^t - \eta\nabla\mathbb{E}[f(\boldsymbol{y}^t - \eta\boldsymbol{w}^t)]$$

©Stanley Chan 2020. All Rights Reserved.

Smoothing the Landscape

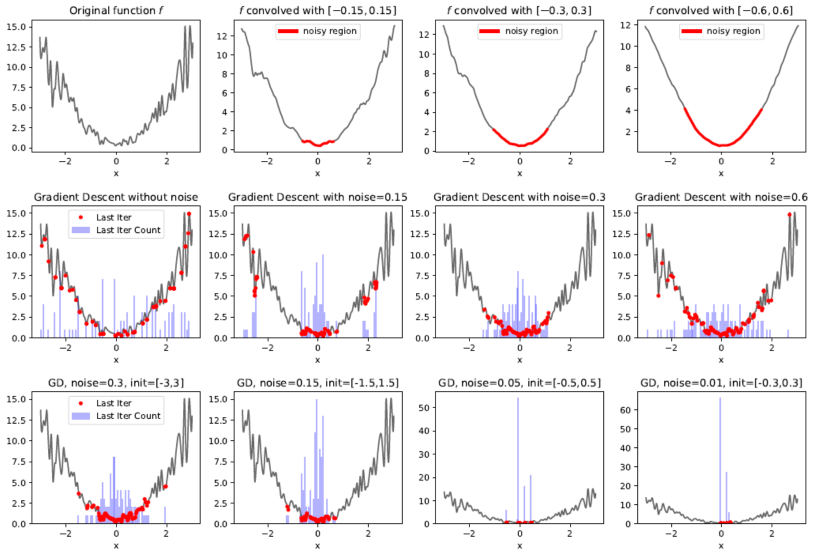
- Let us look at $\mathbb{E}[f(\boldsymbol{y}^t - \eta \boldsymbol{w}^t)]$:

$$\mathbb{E}[f(\boldsymbol{y} - \eta \boldsymbol{w})] = \int f(\boldsymbol{y} - \eta \boldsymbol{w}) p(\boldsymbol{w}) \, d\boldsymbol{w},$$

  where $p(\boldsymbol{w})$ is the distribution of $\boldsymbol{w}$.
- $\int f(\boldsymbol{y} - \eta \boldsymbol{w}) p(\boldsymbol{w}) \, d\boldsymbol{w}$ is the **convolution** between f and p.
- $p(\boldsymbol{w}) \geq 0$ for all $\boldsymbol{w}$, so the convolution always **smoothes** the function.
- Learning rate controls the smoothness
- Too small: Under-smooth. You have not yet escaped from bad local minimum.
- Too large: Over-smooth. You may miss a local minimum.

©Stanley Chan 2020. All Rights Reserved.

Smoothing the Landscape



©Stanley Chan 2020. All Rights Reserved.

Reading List

Gradient Descent

- S. Boyd and L. Vandenberghe, "Convex Optimization", Chapter 9.2-9.4.
- J. Nocedal and S. Wright, "Numerical Optimization", Chapter 3.1-3.3.
- Y. Nesterov, "Introductory lectures on convex optimization", Chapter 2.
- CMU 10.725 Lecture https://www.stat.cmu.edu/~ryantibs/convexopt/lectures/grad-descent.pdf

Stochastic Gradient Descent

- CMU 10.725 Lecture https://www.stat.cmu.edu/~ryantibs/convexopt/lectures/stochastic-gd.pdf
- Kleinberg et al. (2018) "When Does SGD Escape Local Minima", https://arxiv.org/pdf/1802.06175.pdf

ⓒStanley Chan 2020. All Rights Reserved.

Appendix

© Stanley Chan 2020. All Rights Reserved.

Proof of Unbiasedness of SGD gradient

Lemma

If n is a random variable with uniform distribution over $\{1, \ldots, N\}$, then

$$\mathbb{E}\left[\frac{1}{|\mathcal{B}|}\sum_{n\in\mathcal{B}} \nabla J_n(\boldsymbol{\theta})\right] = \nabla J(\boldsymbol{\theta}).$$

Denote the density function of n as $p(n) = 1/N$. Then,

$$\begin{aligned}\mathbb{E}\left[\frac{1}{|\mathcal{B}|}\sum_{n\in\mathcal{B}} \nabla J_n(\boldsymbol{\theta})\right] &= \frac{1}{|\mathcal{B}|}\sum_{n\in\mathcal{B}} \mathbb{E}\left[\nabla J_n(\boldsymbol{\theta})\right] = \frac{1}{|\mathcal{B}|}\sum_{n\in\mathcal{B}}\left\{\sum_{n=1}^{N} J_n(\boldsymbol{\theta})p(n)\right\} \\ &= \frac{1}{|\mathcal{B}|}\sum_{n\in\mathcal{B}}\left\{\frac{1}{N}\sum_{n=1}^{N} J_n(\boldsymbol{\theta})\right\} = \frac{1}{|\mathcal{B}|}\sum_{n\in\mathcal{B}} \nabla J(\boldsymbol{\theta}) = \nabla J(\boldsymbol{\theta}).\end{aligned}$$

©Stanley Chan 2020. All Rights Reserved.

Q&A 1: What is momentum method?

- The momentum method was originally proposed by Polyak (1964).
- Momentum method says:

$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^{t} - \alpha\left[\beta g^{t-1} + (1-\beta)g^{t}\right],$$

  where $g^t = \nabla f(\boldsymbol{x}^t)$, and $0 < \beta < 1$ is the damping constant.
- Momentum method can be applied to both gradient descent and stochastic gradient descent.
- A variant is the Nesterov accelerated gradient (NAG) method (1983).
- Importance of NAG is elaborated by Sutskever et al. (2013).
- The key idea of NAG is to write $\boldsymbol{x}^{t+1}$ as a **linear combination** of $\boldsymbol{x}^{t}$ and the span of the **past gradients**.
- Yurii Nesterov proved that such combination is the best one can do with first order methods.

©Stanley Chan 2020. All Rights Reserved.

Q&A 1: What is momentum method?

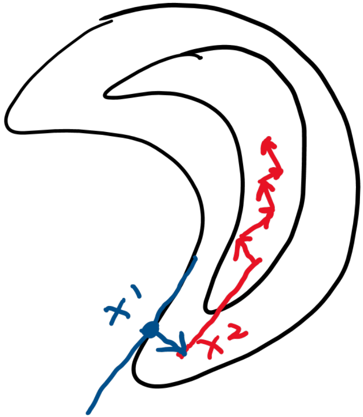
Here are some references on momentum method.

- Sutskever et al. (2013), "On the importance of initialization and momentum in deep learning", http://proceedings.mlr.press/v28/sutskever13.pdf
- UIC Lecture Note https://www2.cs.uic.edu/~zhangx/teaching/agm.pdf
- Cornell Lecture Note http://www.cs.cornell.edu/courses/cs6787/2017fa/Lecture3.pdf
- Yurii Nesterov, "Introductory Lectures on Convex Optimization", 2003. (See Assumption 2.1.4 and discussions thereafter)
- G. Goh, "Why Momentum Really Works", https://distill.pub/2017/momentum/

©Stanley Chan 2020. All Rights Reserved.

Q&A 2: With exact line search, will we get to a minimum in one step?

- No. Exact line search only allows you to converge faster. It does not guarantee convergence in one step.
- Here is an example. The function is called the **rosenbrock function**.



© Stanley Chan 2020. All Rights Reserved.

Q&A 3: Any example of gradient descent?

- Consider the loss function

$$J(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{A\theta} - \boldsymbol{y}\|^2$$

- Then the gradient is

$$\nabla J(\boldsymbol{\theta}) = \boldsymbol{A}^T(\boldsymbol{A\theta} - \boldsymbol{y})$$

- So the gradient descent step is

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t + \eta \underbrace{\boldsymbol{A}^T(\boldsymbol{A\theta}^t - \boldsymbol{y})}_{\nabla J(\boldsymbol{\theta}^t)}.$$

- Since this is a quadratic equation, you can find the exact line search step size (assume $\boldsymbol{d} = -\nabla f(\boldsymbol{\theta})$):

$$\eta = -\|\boldsymbol{d}\|^2/(\boldsymbol{d}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{d}).$$

©Stanley Chan 2020. All Rights Reserved.

Q&A 4: In finding the steepest direction, why is δ unimportant?

- The constraint $\|\boldsymbol{d}\| = \delta$ is necessary for minimizing $\nabla f(\boldsymbol{x})^T \boldsymbol{d}$. Without the constraint, this problem is unbounded below and the solution is $-\infty$ times whatever direction $\boldsymbol{d}$ that lives on the negative half plane of ∇f.
- For any δ, the solution (according to Cauchy Schwarz inequality), is

$$\boldsymbol{d} = -\delta \frac{\nabla f(\boldsymbol{x})}{\|\nabla f(\boldsymbol{x})\|}.$$

  You can show that this $\boldsymbol{d}$ minimizes $\nabla f(\boldsymbol{x})^T \boldsymbol{d}$ and satisfies $\|\boldsymbol{d}\| = \delta$.
- Now, if we use this $\boldsymbol{d}$ in the gradient descent step, the step size α will compensate for the δ.
- So we can just choose $\delta = 1$ and the above derivation will still work.

ⓒStanley Chan 2020. All Rights Reserved.

Q&A 5: What is a good batch size for SGD?

There is no definite answer. Generally you need to look at the validation curve to determine if you need to increase/decrease the mini-batch size. Here are some suggestions in the literature.

- Bengio (2012) https://arxiv.org/pdf/1206.5533.pdf

  [batch size] is typically chosen between 1 and a few hundreds, e.g. [batch size] = 32 is a good default value, with values above 10 taking advantage of the speedup of matrix-matrix products over matrix-vector products.

- Masters and Luschi (2018) https://arxiv.org/abs/1804.07612

  The presented results confirm that using small batch sizes achieves the best training stability and generalization performance, for a given computational cost, across a wide range of experiments. In all cases the best results have been obtained with batch sizes $m = 32$ or smaller, often as small as $m = 2$ or $m = 4$.

©Stanley Chan 2020. All Rights Reserved.