Kernel Methods and Gradient Descent

CSci 5525: Machine Learning

Instructor: Nicholas Johnson

October 1, 2020

Announcements

- HW1 due tonight at 11:59 PM CDT
- HW2 posted today (due Oct 15)

Feature Transformation

- Transform features $\mathbf{x}$ into $\Phi(\mathbf{x})$
- Examples:

$$x \in \mathbb{R}, \Phi(x) = \ln(1 + x)$$
$$\mathbf{x} \in \mathbb{R}^p, \Phi(\mathbf{x}) = (1, x(1), \ldots, x(p), x(1)^2, \ldots, x(p)^2, x(1)x(2), \ldots, x(p-1)x(p))$$
$$x \in \mathbb{R}, \Phi(x) = (1, \sin(x), \cos(x), \sin(2x), \cos(2x), \ldots)$$

- Choice of representations: fixed, implicit, learned
- We consider feature transformations that are easy to compute $\Phi(\mathbf{x})^\top \Phi(\mathbf{x})$ even though $\Phi(\mathbf{x})$ is hard to compute

Feature Transformation Examples: Quadratic

- Let's start small
- Let $\mathbf{x} \in \mathbb{R}^p$, then consider the quadratic expansion:

$$\Phi(\mathbf{x}) = (1, \sqrt{2}x_1, \ldots, \sqrt{2}x_p, x_1^2, \ldots, x_p^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_{p-1}x_p)$$

- The product $\Phi(\mathbf{x})^\top \Phi(\mathbf{x}') = (1 + \mathbf{x}^\top \mathbf{x}')^2$ can be computed in $O(p)$ time as opposed to $O(p^2)$

Feature Transformation Examples: Product of all Subsets

- Now let's increase the dimension more
- Consider the feature expansion

$$\Phi(\mathbf{x}) = \left(\prod_{i \in S} x_i \right)_{S \subseteq [p]}$$

- The product $\Phi(\mathbf{x})^\top \Phi(\mathbf{x}') = \prod_{i=1}^{p} (1 + x_i x_i')$ can be computed in $O(p)$ time as opposed to $O(2^p)$

Feature Transformation Examples: Gaussian Kernel

- Now let's increase the dimension to infinity
- For any $\sigma > 0$, consider the feature expansion

$$\Phi(\mathbf{x})^\top \Phi(\mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{2\sigma^2}\right)$$

- This product can be computed in $O(p)$
- What is Φ?

Feature Transformation Examples: Gaussian Kernel

- Consider simple case of $x \in \mathbb{R}$

$$\begin{aligned}
\Phi(x)\Phi(y) &= \exp(-(x-y)^2/(2\sigma^2)) \\
&= \exp(-x^2/(2\sigma^2))\exp(-y^2/(2\sigma^2))\exp(xy/\sigma^2) \\
&= \exp(-x^2/(2\sigma^2))\exp(-y^2/(2\sigma^2))\sum_{j=0}^{\infty}\frac{1}{j!}(xy/\sigma^2)^j
\end{aligned}$$

- This gives

$$\Phi(x) = \exp(-x^2/(2\sigma^2))\left(1, \frac{x}{\sigma}, \frac{1}{2!}\left(\frac{x}{\sigma}\right)^2, \frac{1}{3!}\left(\frac{x}{\sigma}\right)^3, \cdots\right)$$

which is in $\mathbb{R}^\infty$

- This feature expansion $k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x})^\top \Phi(\mathbf{x}')$ is the radial basis function (RBF) or Gaussian kernel

Kernel

- **Definition.** A kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric function such that for any $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathcal{X}$, the $n \times n$ Gram matrix $\mathbf{G}$ with each (i, j)-th entry $G_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ is positive semidefinite (psd)
  - Recall: a matrix is psd iff $\forall \mathbf{u}, \mathbf{u}^\top \mathbf{G}\mathbf{u} \geq 0$

Creating New Kernels

- Suppose k_1, k_2 are valid kernels, $c \geq 0$, and g is a polynomial function with positive coefficients, f is any function, and $\mathbf{A}$ is a psd matrix. Then the following are valid kernels:
    - $k(\mathbf{x}, \mathbf{x}') = ck_1(\mathbf{x}, \mathbf{x}')$
    - $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$
    - $k(\mathbf{x}, \mathbf{x}') = g(k_1(\mathbf{x}, \mathbf{x}'))$
    - $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}')$
    - $k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}')$
    - $k(\mathbf{x}, \mathbf{x}') = \exp(k_1(\mathbf{x}, \mathbf{x}'))$
    - $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{A} \mathbf{x}'$

Non-linear SVMs

- All important equations have dot-products
  - Dual is expressed in terms of $\mathbf{x}_i^\top \mathbf{x}_j$
  - The predictions are in terms of $\mathbf{x}_i^\top \mathbf{x}$
- How to get a non-linear classifier:
  - Map $\mathbf{x}$ to some (higher dimensional) space $\Phi : \mathbb{R}^p \mapsto \mathcal{H}$
  - Derived feature vectors are $\Phi(\mathbf{x}_i), \forall i$
  - Dot products are $\Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}_j)$
- Kernel function allows *implicit calculation* of dot-products

$$k(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}_j)$$

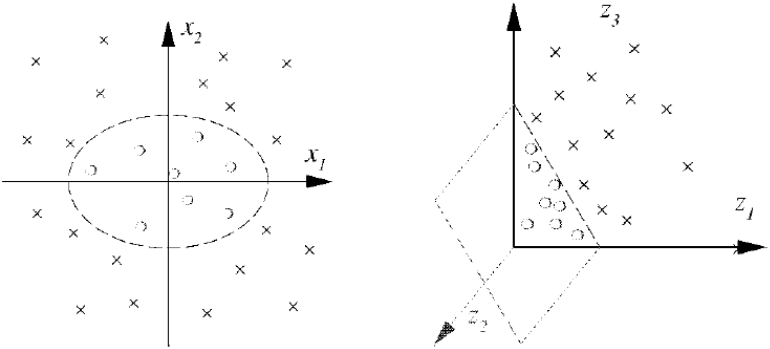
- Learn a linear max margin separator in $\mathcal{H}$
- The final prediction function

$f(x) = w^T x$

new point

$$f(\mathbf{x}) = \underbrace{\sum_{i:\lambda_i^* > 0} y_i \lambda_i^* \Phi(\mathbf{x}_i)^\top}_{w^*} \Phi(\mathbf{x}) = \sum_{i:\lambda_i > 0} y_i \lambda_i^* \underline{k(\mathbf{x}_i, \mathbf{x})}$$

Example



$$\Phi([x_1, x_2]) = [x_1^2, \sqrt{2}x_1x_2, x_2^2]$$

Example

$\phi(x) = x$

$\phi^T(x_i)\,\phi(x_j) = x_i^T x_j$

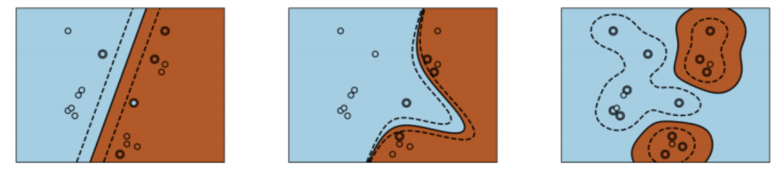


Figure : From left to right: decision boundaries of kernel SVM with linear, 3rd degree polynomial, and RBF kernels.

Kernelized Ridge Regression

- Ridge regression solution: $\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y}$
- Linear algebra trick: let $\mathbf{P}$ be an $n \times m$ matrix and $\mathbf{Q}$ be an $m \times n$ matrix then

$$(\mathbf{PQ} + \mathbf{I}_n)^{-1} \mathbf{P} = \mathbf{P}(\mathbf{QP} + \mathbf{I}_m)^{-1}$$

- Set $\mathbf{P} = (1/\lambda)\mathbf{X}^\top$ and $\mathbf{Q} = \mathbf{X}$ then

$$(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_p)^{-1} \mathbf{X}^\top = \mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_n)^{-1} = \mathbf{X}^\top (\mathbf{G} + \lambda \mathbf{I}_n)^{-1}$$

where $\mathbf{G}$ is the $n \times n$ Gram matrix with $G_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$

Kernelized Ridge Regression

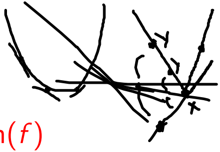
- Ridge regression solution:

$$\mathbf{w}^* = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_p)^{-1}\mathbf{X}^\top\mathbf{y} = \mathbf{X}^\top\underbrace{(\mathbf{G} + \lambda\mathbf{I}_n)^{-1}\mathbf{y}}_{\mathbf{v}}$$
$$= \mathbf{X}^\top\mathbf{v}$$
$$= \sum_{i=1}^{n} v_i\mathbf{x}_i \qquad f(x) = w^\top x$$

- For any new sample $\mathbf{x}$ the prediction will be $\mathbf{x}^\top\mathbf{w}^* = \sum_{i=1}^n v_i\mathbf{x}^\top\mathbf{x}_i$
- If we replace $\mathbf{x}_i$ with $\Phi(\mathbf{x}_i)$ then $\mathbf{G}$ is $G_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$
- Then prediction time we have $\Phi(\mathbf{x})^\top\mathbf{w}^* = \sum_{i=1}^n v_i\Phi(\mathbf{x})^\top\Phi(\mathbf{x}_i) = \sum_{i=1}^n v_i k(\mathbf{x}, \mathbf{x}_i)$

(Stochastic) Gradient Descent

Convex Optimization: Types of Functions



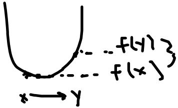
- Convex function f, $\mathrm{dom}(f) \subseteq \mathbb{R}^p$
- Subgradient of f at $\mathbf{x}$: any $g \in \mathbb{R}^p$ satisfying

$$f(\mathbf{y}) \geq f(\mathbf{x}) + (\mathbf{y}-\mathbf{x})^\top g \;, \quad \forall \mathbf{y} \in \mathrm{dom}(f)$$

Set of all subgradients: $\partial f(\mathbf{x})$, sub-differential set

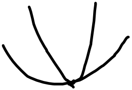
hinge

- f is L-Lipschitz: $\forall g \in \partial f(\mathbf{x}), ||g|| \leq L$



$$\longrightarrow \quad |f(\mathbf{x}) - f(\mathbf{y})| \leq L||\mathbf{x}-\mathbf{y}||$$

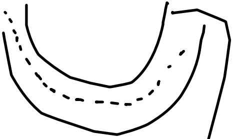
- f is β-smooth, $\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq \beta \|\mathbf{x}-\mathbf{y}\|$



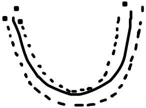
$$f(\mathbf{x}) \leq f(\mathbf{y}) + (\mathbf{x}-\mathbf{y})^\top \nabla f(\mathbf{y}) + \frac{\beta}{2}\|\mathbf{x}-\mathbf{y}\|^2$$

- f is non-smooth, if f is not smooth
  - Example: $f(x) = \max(0, 1-x)$ ← hinge

Convex Optimization: Types of Functions



- f is α-strongly convex



$$f(\mathbf{x}) \geq f(\mathbf{y}) + (\mathbf{x}-\mathbf{y})^\top \nabla f(\mathbf{y}) + \frac{\alpha}{2}\|\mathbf{x}-\mathbf{y}\|^2$$

- f is β-smooth and α-strongly convex

$$f(\mathbf{x}) \geq f(\mathbf{y}) + (\mathbf{x}-\mathbf{y})^\top \nabla f(\mathbf{y}) + \frac{\alpha}{2}\|\mathbf{x}-\mathbf{y}\|^2$$

$$f(\mathbf{x}) \leq f(\mathbf{y}) + (\mathbf{x}-\mathbf{y})^\top \nabla f(\mathbf{y}) + \frac{\beta}{2}\|\mathbf{x}-\mathbf{y}\|^2$$

Non-Smooth, Lipschitz, Convex Functions

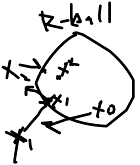
- Often work with "non-smooth" functions f
  - Hinge loss, L_1 norm, etc.
- Consider a non-smooth function f on domain S
- Sub-differential set $\partial f(\mathbf{x})$: $g \in \partial f(x)$ if

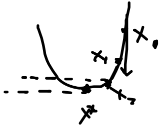
$$f(\mathbf{y}) \geq f(\mathbf{x}) + (\mathbf{y} - \mathbf{x})^\top g, \quad \forall \mathbf{y} \in S$$

- A non-smooth function is convex if $\partial f(\mathbf{x}) \neq \emptyset, \forall \mathbf{x} \in S$
  - Sub-differential set $\partial f(\mathbf{x})$ is convex, compact
  - Each $g \in \partial f(\mathbf{x})$ is a sub-gardient
- Lipschitz convex functions f on domain S:
  - f is convex on S, f has a minimizer $\mathbf{x}^*$ in S
  - f is L-Lipschitz, $\forall g \in \partial f(\mathbf{x})$, $\|g\| \leq L$

Non-Smooth Lipschitz: Projected Subgradient Descent

- Assume $\|x^*\| \leq R$
- Assume f is L-Lipschitz in the R-ball, i.e., $\|g\| \leq G$ for $g \in \partial f(x)$ for $\|x\| \leq R$
- Projected sub-gradient descent





$$y_{t+1} = x_t - \eta g_t \;, \quad \text{where } g_t \in \partial f(x_t)$$

$$x_{t+1} = \begin{cases} y_{t+1} \;, & \text{if } \|y_{t+1}\| \leq R \\ \frac{R}{\|y_{t+1}\|} y_{t+1} \;, & \text{if } \|y_{t+1}\| > R \end{cases}$$

$x_1 = x_0 - \eta \nabla f(x_0)$

$x_2 = x_1 - \eta \nabla f(x_1)$

- With $\eta = \frac{R}{G\sqrt{T}}$, $\bar{x}_T = \frac{1}{T}\sum_{t=1}^{T} x_t$ satisfies

$$f(\bar{x}_T) - f(x^*) \leq \frac{RG}{\sqrt{T}}$$

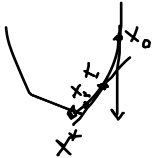
$O\left(\frac{1}{\sqrt{T}}\right)$

  - Step-size is more conservative, compared to smooth functions
  - Rate cannot be improved by "acceleration"
  - Bound holds for $\tilde{x}_T = \text{argmin}_{1 \leq t \leq T} f(x_t)$

Smooth: Gradient Descent

- Smooth convex function f on domain S
  - f has a minimizer x^* in S
  - f is convex and continuously differentiable on S
  - f is β-smooth, gradient ∇f is β-Lipschitz: $\forall x, y \in S$

$$\|\nabla f(x) - \nabla f(y)\| \leq \beta \|x - y\|$$



- Initial point x_0, iterative updates:

$$x_{t+1} = x_t - \eta \nabla f(x_t)$$

- With $\eta = \frac{1}{\beta}$, we have

$$f(x_T) - f(x^*) \leq \frac{2\beta \|x_0 - x^*\|^2}{T - 1}$$

$\beta\left(\frac{1}{T}\right)$

Smooth Functions: Accelerated Gradient Descent

- Assume f is β-smooth
- Initial point x_0, $\lambda_0 = 0$, iterative updates:

$$y_{t+1} = x_t - \frac{1}{\beta}\nabla f(x_t)$$

$$\lambda_{t+1} = \frac{1 + \sqrt{1 + 4\lambda_t^2}}{2}$$

$$x_{t+1} = y_{t+1} + \frac{\lambda_t - 1}{\lambda_{t+1}}(y_{t+1} - y_t)$$

- Nesterov's Accelerated Gradient Descent satisfies

$$f(x_T) - f(x^*) \leq \frac{2\beta\|x_0 - x^*\|^2}{T^2}$$

Iteration Complexity

- Algorithm needs access to an oracle
  - 0^{th} order: Given x, what is $f(x)$
  - 1^{st} order: Given x, what is $\nabla f(x)$ (or sub-gradient)
- An algorithm with a 1^{st} order oracle is a mapping:

$$x_t = \phi_t(\{x_\tau, f(x_\tau), \nabla f(x_\tau)\}, \tau = 0, \ldots, t-1)$$

- Iteration complexity: T to get $f(x_T) - f(x^*) \leq \epsilon$
  - GD for smooth functions: $T = O(\frac{1}{\epsilon})$
  - AGD for smooth functions: $T = O(\frac{1}{\sqrt{\epsilon}})$
  - 'GD' for non-smooth functions: $T = O(\frac{1}{\epsilon^2})$

Iteration Complexities

f	Algorithm	Rate	# Iterations
non-smooth, Lipschitz	PGD	$\frac{RL}{\sqrt{t}}$	$\frac{R^2L^2}{\epsilon^2}$
smooth	PGD	$\frac{\beta R^2}{t}$	$\frac{\beta R^2}{\epsilon}$
smooth	AGD	$\frac{\beta R^2}{t^2}$	$\frac{\sqrt{\beta}R}{\sqrt{\epsilon}}$
smooth, strongly convex	PGD	$R^2 \exp\left(-\frac{t}{Q}\right)$	$Q \log\left(\frac{R^2}{\epsilon}\right)$
smooth, strongly convex	AGD	$R^2 \exp\left(-\frac{t}{\sqrt{Q}}\right)$	$\sqrt{Q} \log\left(\frac{R^2}{\epsilon}\right)$

$Q = \frac{\beta}{\alpha}$

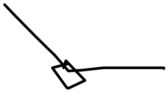
Gradient Descent for Machine Learning

- Machine learning problem (supervised learning):

$$\min_{\mathbf{w}} f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell((\mathbf{x}_i, y_i), \mathbf{w}) + \lambda R(\mathbf{w})$$

  - $\mathbf{x}_i$ is a data point, y_i is the label
- Examples:
  - Smooth: Linear classification with logistic regression

$$\frac{1}{n}\sum_{i=1}^{n} \left\{-y_i \mathbf{w}^\top \mathbf{x}_i + \log(1 + \exp(-\mathbf{w}^\top \mathbf{x}_i))\right\}$$



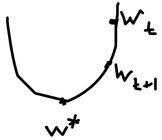
  - Non-smooth: Linear classification with SVMs

$$\frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - y_i \mathbf{w}^\top \mathbf{x}_i) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

Gradient Descent for Machine Learning

- Machine learning problem (supervised learning):

$$\min_{\mathbf{w}} f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell((\mathbf{x}_i, y_i), \mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell_i(\mathbf{w})$$



- At each iteration t, compute $\nabla f(\mathbf{w}_t)$
- Let $g_i = \nabla \ell_i(\mathbf{w})$,

$$\nabla f(\mathbf{w}) = \frac{1}{n}\cancel{\sum_{i=1}^{n}} g_i$$

- Per iteration computational complexity: $O(n)$
- Overall computation complexity: $O(\frac{n}{\epsilon})$, scales *linearly* with n
- Bad news for "big data": millions or more data points
  - For $n = 10^6, \epsilon = 10^{-2}$, 10^8 gradient computations

Stochastic Gradient Descent (SGD)

- Can we do better than gradient descent?
- Gradient descent for smooth functions: $O(\frac{n}{\epsilon})$
  - Number of iterations $O(\frac{1}{\epsilon})$
  - Runtime in each iteration n
- Sub-gradient descent for non-smooth functions: $O(\frac{n}{\epsilon^2})$
  - Number of iterations $O(\frac{1}{\epsilon^2})$
  - Runtime in each iteration n
- Main idea:
  - Decease the runtime in each iteration
  - Possibly increase the number of iterations
- Simplest case: Compute only 1 gradient per iteration
- Questions: What is the algorithm? Will this converge?

Stochastic Gradient Descent (SGD)

- Assume: Samples $(\mathbf{x}_i, y_i)$ are i.i.d., consider

$$\min_{\mathbf{w}} f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell((\mathbf{x}_i, y_i), \mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \ell_i(\mathbf{w})$$

- Stochastic gradient descent:
  - For $t = 1, \ldots, T$
    - Randomly draw $i_t \in \{1, \ldots, n\}$
    - Compute (sub)gradient $g_{i_t} = \nabla \ell_i(\mathbf{w}_t)$
    - $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t g_{i_t}$
  - Output $\bar{\mathbf{w}}_T = \frac{1}{T}\sum_{t=1}^{T} \mathbf{w}_t$

SGD: Step-size, Convergence

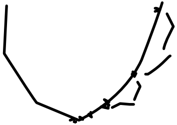
- Assume $E[\|g\|^2] \leq G^2$

$O\left(\frac{1}{\sqrt{T}}\right)$

- Fixed step-size: $\eta_t = \frac{\|\mathbf{w}^*\|_2}{G\sqrt{T}}$

$$E[f(\bar{\mathbf{w}}_T)] - f(\mathbf{w}^*) \leq \frac{G\|\mathbf{w}^*\|_2}{\sqrt{T}}$$

$O\left(\frac{1}{\sqrt{T}}\right)$



- Decaying step-size: $\eta_t = \frac{\|\mathbf{w}^*\|_2}{G\sqrt{t}}$

$$E[f(\bar{\mathbf{w}}_T)] - f(\mathbf{w}^*) \leq \frac{4G\|\mathbf{w}^*\|_2}{\sqrt{T}}$$

$O\left(\frac{1}{\sqrt{T}}\right)$

- Unknown $G, \|\mathbf{w}^*\|$: $\eta_t = \frac{\beta\|\mathbf{w}^*\|_2}{G\sqrt{t}}$, for some $\beta > 0$

$$E[f(\bar{\mathbf{w}}_T)] - f(\mathbf{w}^*) \leq \frac{4G\|\mathbf{w}^*\|_2}{\sqrt{T}} \max\left(\beta, \frac{1}{\beta}\right)$$

$O\left(\frac{1}{\sqrt{T}}\right)$

- Step size is small, slow convergence
  - Needed to balance variance of (noisy) gradient

Smooth Functions: SGD vs GD

- SGD convergence rate:

$$\mathbb{E}[f(\bar{\mathbf{w}}_T)] - f(\mathbf{w}^*) \leq O\left(\frac{1}{\sqrt{T}}\right)$$

$$\text{Iteration complexity } T = O\left(\frac{1}{\epsilon^2}\right)$$

Smooth functions	GD	SGD
Number of iterations	$O(\frac{1}{\epsilon})$	$O(\frac{1}{\epsilon^2})$
Each iteration	n	1
Total runtime	$O(\frac{n}{\epsilon})$	$O(\frac{1}{\epsilon^2})$
$n = 10^6$, $\epsilon = 10^{-2}$	10^8	10^4

- GD vs SGD: full gradient vs random gradient
- SGD is memory efficient, extends to mini-batches

Non-smooth Functions: SGD vs GD

- SGD convergence rate:

$$\mathbb{E}[f(\bar{\mathbf{w}}_T)] - f(\mathbf{w}^*) \leq O\left(\frac{1}{\sqrt{T}}\right)$$

$$\text{Iteration complexity } T = O\left(\frac{1}{\epsilon^2}\right)$$

Non-smooth functions	GD	SGD
Number of iterations	$O(\frac{1}{\epsilon^2})$	$O(\frac{1}{\epsilon^2})$
Each iteration	n	1
Total runtime	$O(\frac{n}{\epsilon^2})$	$O(\frac{1}{\epsilon^2})$
$n = 10^6$, $\epsilon = 10^{-2}$	10^{10}	10^4

- GD is $O(n)$ slower than SGD
- Example: Hinge loss (SVMs)