CS5339 Machine Learning

Optimization I

Lee Wee Sun
School of Computing
National University of Singapore
leews@comp.nus.edu.sg

Semester 2, 2019/20

Computation

We discuss the intrinsic computational difficulties in some learning problems and look at gradient methods for convex problems.

Outline

Computational Complexity of Learning

Consider a sequence of learning problems, $(Z_n, \mathcal{H}_n, \ell_n)_{n=1}^{\infty}$ where n denotes a complexity parameter we are interested in. We say that a learning algorithm A is an efficient algorithm if its runtime is $O(p(n, 1/\epsilon, 1/\delta))$ for some *polynomial* p.

- For example, for the class of axis aligned rectangles, we may be interested in knowing whether there is an efficient algorithm as the number of dimension d grows. In this case, we use $n = d$.

Axis Aligned Rectangles

Axis aligned rectangles are efficiently learnable in the realizable case:

- VC dimension is $2d$, so sample complexity is $O(\frac{d\log(1/\epsilon)+\log(1/\delta)}{\epsilon})$. (For simplicity, we only consider sample complexity of realizable cases in this section.)
- Only need to be able to do ERM efficiently to be efficiently learnable.

- In realizable case, can efficiently find smallest enclosing rectangle for the positive examples by removing negative examples, and taking the smallest bounding box.

On the other hand, the ERM problem is NP-hard for axis aligned rectangles in the non-realizable case.

Exercise O1-1 (Archipelago):
Consider learning an axis parallel rectangle in $\mathbb{R}^2$. The training data is
$((1,5),-),((3,4),+),((7,7),-),((2,5),+),((3,3),+),((5,1),-)$.
Output a rectangle that classifies all examples correctly, in the format
$(\mathit{LeftBoundary}, \mathit{RightBoundary}, \mathit{BottomBoundary}, \mathit{TopBoundary})$.

Linear Functions

Linear (affine) functions are efficiently learnable in the realizable case:

- VC dimension is $d + 1$ where d is the input dimension, so sample complexity is $O(\frac{d \log(1/\epsilon) + \log(1/\delta)}{\epsilon})$.

- Linear programming can be used to do ERM efficiently when the problem is separable.

Unfortunately, ERM is NP-hard for linear functions in the non-realizable case.

Boolean Conjunctions

A Boolean conjunction is a mapping $\mathcal{X} = \{0,1\}^d$ to $\mathcal{Y} = \{0,1\}$ expressed as a propositional formula of the form $x_{i_1} \wedge \ldots, x_{i_k} \wedge \neg x_{j_1} \wedge \ldots, \wedge \neg x_{j_r}$ for some indices $i_1, \ldots, i_k, j_1, \ldots, j_r \in \{1, \ldots, d\}$.

- There are at most $3^d + 1$ formulas: each variable either appears, appear with a negation, or does not appear, plus an additional all negative formula.

- For finite classes, sample complexity is polynomial in $O(\frac{1}{\epsilon}(\log|\mathcal{H}| + \log(1/\delta))$, hence in this case polynomial in d.

In the non-realizable case, it is NP-hard to find a conjunction that minimizes the empirical risk.

We describe an efficient algorithm for learning Boolean conjunctions in the realizable case.

- Remove all negative examples.
- Start with $h_0 = x_{i_1} \wedge \ldots, x_{i_k} \wedge \neg x_{j_1} \wedge \ldots, \wedge \neg x_{j_r}$.
  - Note that this classifies all instances as negative.
- At step t, add the t-th positive example. Remove all literals in h_{t-1} that does not agree with the new example to form h_t.
- This maintains the invariant that h_t is the most restrictive conjunction that classifies all the positive examples that has been added.
  - Only necessary literals are removed at each step.
  - The larger the conjunction size, the more restrictive it is (classifies the fewest instances as positive).
- Runtime of $O(md)$.

Exercise O1-2 (Archipelago):
Consider learning a conjunction in $\{0,1\}^2$. Each labeled instance is of the form $((x_1, x_2), \mathit{class})$. The training data is $((0,0),-),((0,1),+),((1,1),+)$. Give the conjunction produced by the learning algorithm.

3-Term DNF

Consider the class $\mathcal{H}_d$ of 3-term DNF, represented by Boolean formula of the form $h(\mathbf{x}) = A_1(\mathbf{x}) \vee A_2(\mathbf{x}) \vee A_3(\mathbf{x})$, where $A_i(\mathbf{x})$ is a Boolean conjunction.

- The size $|\mathcal{H}_d|$ is $O(3^{3d})$, so sample complexity is $O(\frac{1}{\epsilon}(d + \log(1/\delta))$.

- But unless RP=NP, no polynomial time algorithm for learning 3-term DNF using 3-term DNF formula (proper learning) even in the realizable case.

However, we can learn 3-term DNF functions using a larger class of 3-CNF.

- Can rewrite any 3-term DNF formula as

$$A_1 \vee A_2 \vee A_3 = \wedge_{u \in A_1, v \in A_2, w \in A_3} (u \vee v \vee w).$$

- This is a 3-CNF formula.
- Consider $(u \vee v \vee w)$. We can form a new variable x_{uvw} that takes the value 1 iff $(u \vee v \vee w)$ takes the value 1.
  - There are $(2d)^3$ such variables.
  - We can learn a conjunction over these variables to learn 3-term DNF using 3-CNF.
  - Sample complexity is $O(\frac{1}{\epsilon}(d^3 + \log(1/\delta))$.

Interesting phenomena: No efficient algorithm for proper learning, but by enlarging the class of functions, can be efficiently learned.

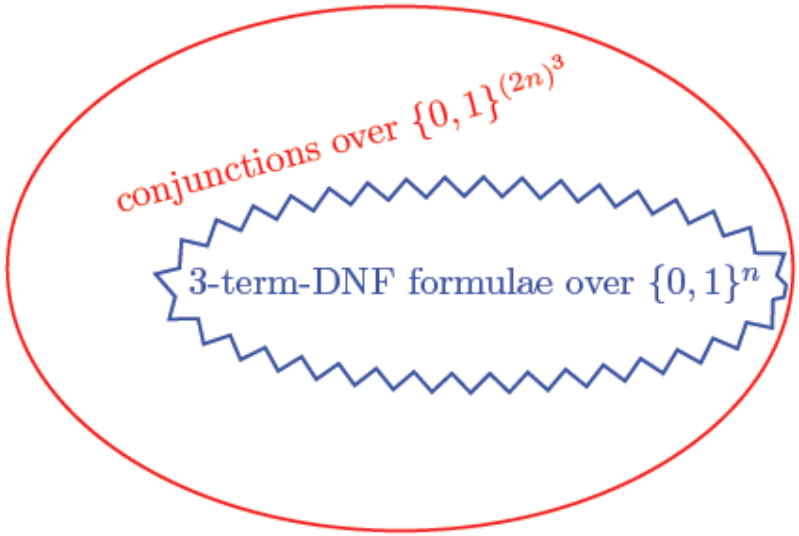


Figure from SSBD.

But there are also function classes that are not efficiently learnable regardless of representation, under cryptographic assumptions.

- For example, inverting trapdoor one-way functions.

Exercise O1-3 (Archipelago):
Compare the sample complexity of proper learning of 3-term DNF with the sample complexity of using 3-CNF to learn 3-term DNF. What is the trade-off that you are making?

Outline

Gradient

- Consider function $y = f(w)$.
- Assume differentiable with derivative denoted $f'(w)$ or $\frac{dy}{dw}$.
- Points where $f'(w) = 0$ are critical points or stationary points. A critical point may be
  - A local minimum: $f(w)$ is lower than all neighbouring points.
  - A local maximum: $f(w)$ is larger than all neighbouring points.
  - A saddle point: neither a local minimum or maximum.

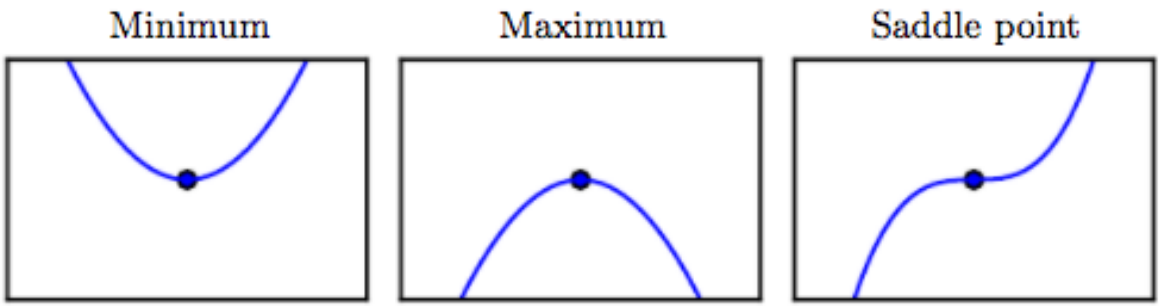


Figure from [1].

- A local minimum that obtains the lowest possible value of $f(w)$ is a global minimum.

- For functions with multiple inputs, we consider partial derivatives $\frac{\partial}{\partial w_i} f(\mathbf{w})$ for change with respect to w_i.
- The gradient is a vector containing all the partial derivatives denoted

$$\nabla_w f(\mathbf{w}) = \begin{bmatrix} \frac{\partial}{\partial w_1} f(\mathbf{w}) \\ \frac{\partial}{\partial w_2} f(\mathbf{w}) \\ \vdots \\ \frac{\partial}{\partial w_d} f(\mathbf{w}) \end{bmatrix}$$

- The gradient is the direction of steepest ascent, i.e. the value of the function increases the fastest in the direction of the gradient.

- Correspondingly the direction of steepest descent is the negative gradient.
- Gradient descent (GD) is an algorithm that iteratively takes a step in the negative gradient direction.

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}} f(\mathbf{w}^{(t)}),$$

where η is the learning rate that determines the step size.

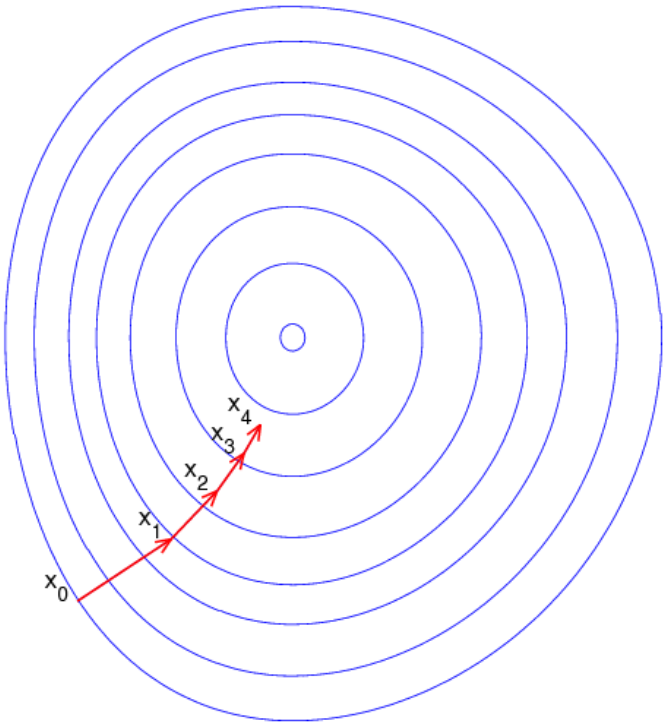


Figure from Wikipedia.

Hessian

- The matrix of second derivatives is called the Hessian matrix

$$\mathbf{H}(f)(\mathbf{w})_{i,j} = \frac{\partial^2}{\partial w_i \partial w_j} f(\mathbf{w}).$$

- The Hessian matrix is symmetric when the second partial derivatives are continuous.

- The second derivative in direction $\mathbf{d}$ is $\mathbf{d}^T\mathbf{H}\mathbf{d}$.
  - When $\mathbf{d}$ is an eigenvector of unit length, the second derivative in direction $\mathbf{d}$ is given by the corresponding eigenvalue.
  - The maximum eigenvalue determines the maximum second derivative and the minimum eigenvalue determines the minimum.

- For one dimensional function, at a critical point $f'(w) = 0$
  - The second derivative $f''(w) > 0$ means that the function increases as we move away: local minimum.
  - $f''(w) < 0$: local maximum
  - $f''(0) = 0$: inconclusive, may be saddle point or flat region.

- In higher dimensions, the gradient $\nabla_{\mathbf{w}} f(\mathbf{w}^{(t)}) = 0$ at critical points.
  - Positive definite Hessian (all eigenvalues positive) implies a local minimum.
  - Negative definite Hessian (all eigenvalues negative) implies a local maximum.
  - At least one positive eigenvalue and at least one negative eigenvalue: saddle point.
  - Inconclusive if some eigenvalues are zero and the rest the same sign.

- Consider a second-order Taylor series expansion around $\mathbf{w}^{(0)}$ with gradient $\mathbf{g}$ and Hessian $\mathbf{H}$ at $\mathbf{w}^{(0)}$

$$f(\mathbf{w}) \approx f(\mathbf{w}^{(0)}) + (\mathbf{w} - \mathbf{w}^{(0)})^T \mathbf{g} + \frac{1}{2}(\mathbf{w} - \mathbf{w}^{(0)})^T \mathbf{H}(\mathbf{w} - \mathbf{w}^{(0)}).$$

- Assuming a learning rate η, after a gradient descent update

$$f(\mathbf{w}^{(0)} - \eta \mathbf{g}) \approx f(\mathbf{w}^{(0)}) - \eta \mathbf{g}^T \mathbf{g} + \frac{1}{2}\eta^2 \mathbf{g}^T \mathbf{H} \mathbf{g}.$$

- When $\mathbf{g}^T \mathbf{H} \mathbf{g}$ is too large the function can actually increase.
- Solving for η, the optimal value of η is

$$\eta^* = \frac{\mathbf{g}^T \mathbf{g}}{\mathbf{g}^T \mathbf{H} \mathbf{g}}.$$

- When $\mathbf{g}$ aligns with the eigenvector $\eta^* = 1/\lambda$, where λ is the eigenvalue. The optimal step size is smallest when $\eta^* = 1/\lambda_{\max}$.

- The condition number $|\lambda_{\max}/\lambda_{\min}|$ of $\mathbf{H}$ affects how quickly the function changes to small changes in the input.
- When condition number is large, gradient descent does poorly.
  - In one direction, derivative increases rapidly, while in another direction, slowly.
  - Choosing step size also difficult, overshoot in one direction, too slow in another.
  - Issue can be solved using second order methods.

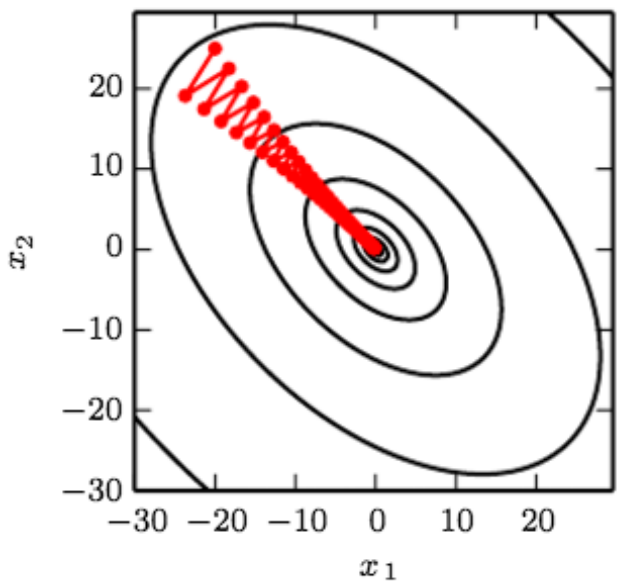


Figure from [1].

Newton's Method

- Based on second order Taylor approximation

$$f(\mathbf{w}) \approx f(\mathbf{w}^{(0)}) + (\mathbf{w} - \mathbf{w}^{(0)})^T \mathbf{g} + \frac{1}{2}(\mathbf{w} - \mathbf{w}^{(0)})^T \mathbf{H}(\mathbf{w} - \mathbf{w}^{(0)}).$$

- Differentiating with respect to $\mathbf{w}$ and solving, we get

$$\mathbf{w}^* = \mathbf{w}^{(0)} - \mathbf{H}^{-1}\mathbf{g}.$$

- Gradient methods such as gradient descent called first-order optimization algorithm. Methods that use Hessian, e.g. Newton's method, called second-order optimization algorithms.

- If f is positive definite quadratic function, opt in one step.

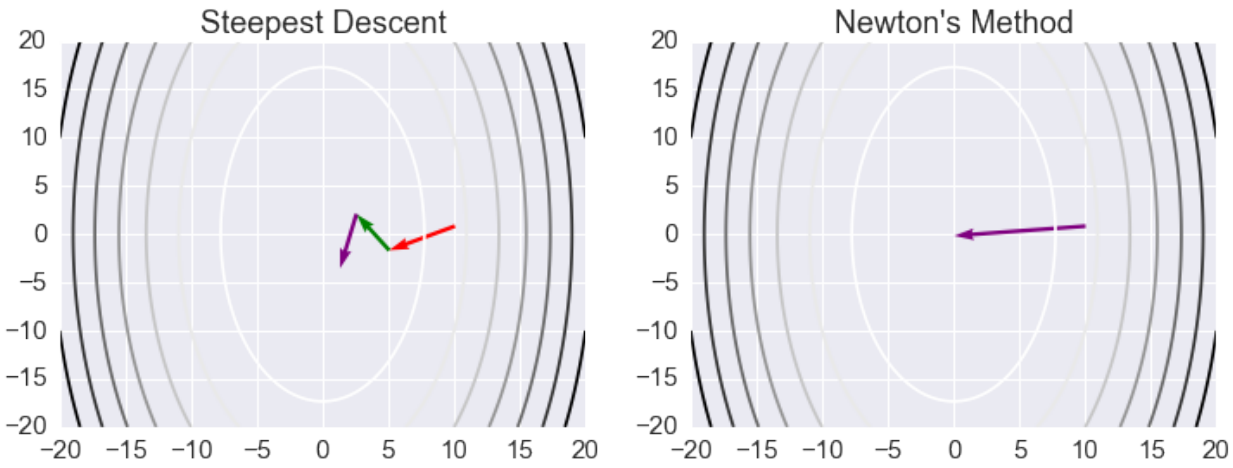


Figure from `http://people.duke.edu/~ccc14/sta-663-2016/12_MultivariateOptimizationAlgorithms.html#Newton's-Method`.

- Useful near local minimum, but harmful near saddle point.
- Good for convex optimization, but less clear for neural networks.

Stochastic Gradient

- Instead of requiring the gradient, in stochastic gradient, we allow the descent direction to be a random vector whose expected value is equal to the gradient.
  - Gradient of a single randomly selected example has this property.
  - Under some conditions (discussed later), faster than gradient descent with the entire data set (batch).
  - In practice, common to use minibatch of size larger than 1. Allows parallelism, better GPU processing, etc.
  - If data is not repeated, stochastic gradient follows the gradient of the generalization error instead of training error. But in practice, usually multiple passes of training error is done.

Outline

Surrogate Loss Function

- Minimizing the $0-1$ loss for linear threshold function is NP-hard in the non-realizable case.
  - Non-convex optimization problem.
- One common trick is to upper bound the non-convex loss with a convex loss.
  - Known as a surrogate loss
  - Should upper bound the loss so that minimizing it also helps to minimize the loss.
  - Convex so that it can be solved efficiently.

- Example: the hinge loss can be used as a convex surrogate for the $0-1$ loss.

$$\ell^{hinge}(\mathbf{w},(\mathbf{x},y)) = \max\{0, 1 - y\mathbf{w}^T\mathbf{x}\}.$$

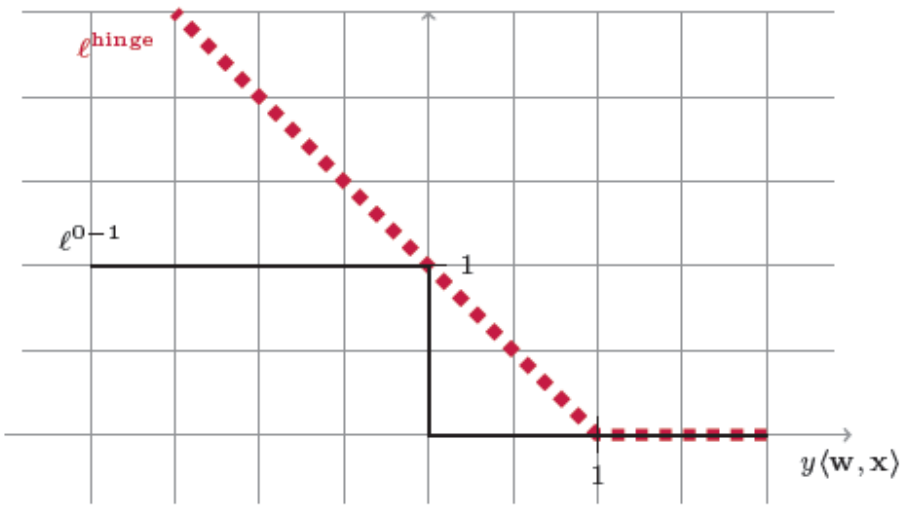


Figure from SSBD.

- With the use of a surrogate loss, we have additional error caused by using the approximate loss, instead of the original loss itself.

Convexity

Definition (Convex Set): (SSBD Defn 12.1) A set C in a vector space is convex if for any two vectors $\mathbf{u}, \mathbf{v} \in C$, the line segment between $\mathbf{u}$ and $\mathbf{v}$ is contained in C. That is, for any $\alpha \in [0, 1]$, we have that $\alpha\mathbf{u} + (1 - \alpha)\mathbf{v} \in C$.

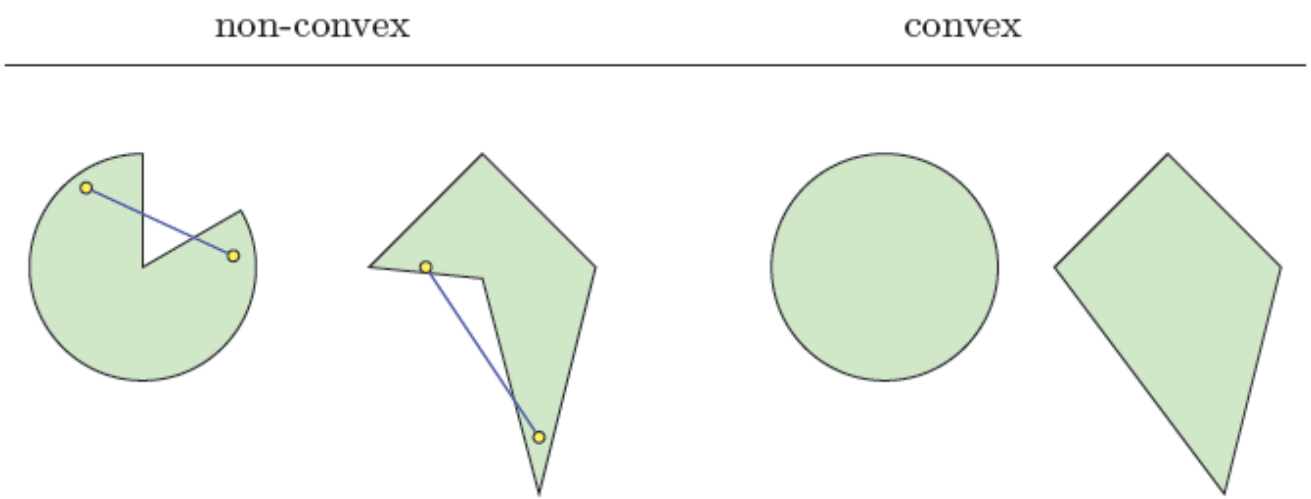


Figure from SSBD.

The combination $\alpha\mathbf{u} + (1 - \alpha)\mathbf{v} \in C$ is called a convex combination.

Definition (Convex function): (SSBD Defn 12.2) Let C be a convex set. A function $f : C \to \mathbb{R}$ is convex if for every $\mathbf{u}, \mathbf{v} \in C$ and $\alpha \in [0, 1]$,

$$f(\alpha\mathbf{u} + (1-\alpha)\mathbf{v}) \leq \alpha f(\mathbf{u}) + (1-\alpha) f(\mathbf{v}).$$

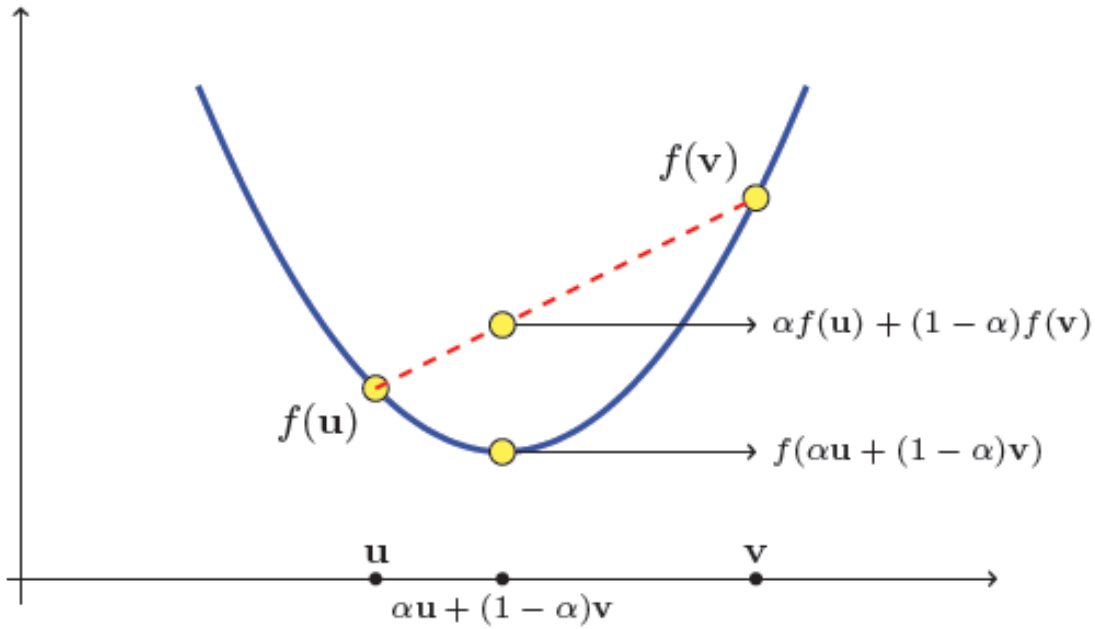


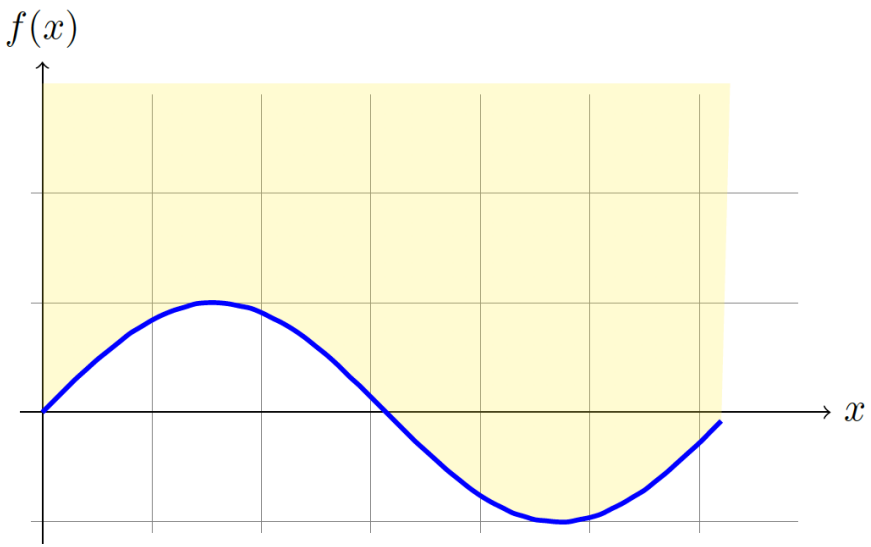
Figure from SSBD.

This property is often called Jensen's inequality.

- The *epigraph* of a function f is the set

$$\text{epigraph}(f) = \{(\mathbf{x}, \beta) : f(\mathbf{x}) \leq \beta\}.$$

A function f is convex if and only if its epigraph is convex.



Epigraph shaded. Figure from SSBD.

- Let $f(\mathbf{u})$ be a local minimum of f. If f is convex, then $f(\mathbf{u})$ is also a global minimum. The see this, note that by definition of local minimum, for any $\mathbf{v}$, there is an α such that

$$\begin{aligned} f(\mathbf{u}) &\leq f(\mathbf{u} + \alpha(\mathbf{v} - \mathbf{u})) \\ &= f(\alpha\mathbf{v} + (1-\alpha)\mathbf{u}) \\ &\leq \alpha f(\mathbf{v}) + (1-\alpha) f(\mathbf{u}). \end{aligned}$$

  Rearranging, we see that $f(\mathbf{u}) \leq f(\mathbf{v})$.

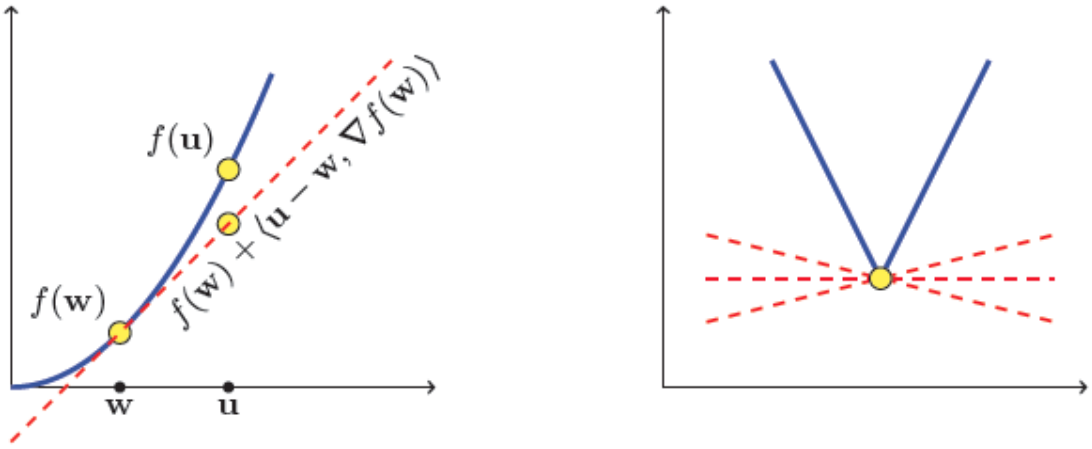


Figure from SSBD.

A differentiable f with convex domain is convex iff

$$\forall \mathbf{u}, f(\mathbf{u}) \geq f(\mathbf{w}) + \nabla f(\mathbf{w})^T(\mathbf{u} - \mathbf{w}),$$

i.e. the gradient at $\mathbf{w}$ defines a plane that lies entirely below f.
If f is convex but not necessarily differentiable, there is also (possibly more than one) vector $\mathbf{v}$ such that

$$\forall \mathbf{u}, f(\mathbf{u}) \geq f(\mathbf{w}) + \mathbf{v}^T(\mathbf{u} - \mathbf{w}),$$

where $\mathbf{v}$ is called a subgradient denoted $\partial f(\mathbf{w})$.

Lemma: (SSBD Lemma 12.3) Let $f : \mathbb{R} \to \mathbb{R}$ be a scalar differentiable function, and let f' and f'' be the first and second derivatives respectively. Then the following are equivalent.

1. f is convex.
2. f' is monotonically non-decreasing.
3. f'' is non-negative.

- The function $f(x) = x^2$ is convex; $f'(x) = 2x$ is monotonically non-decreasing and $f''(x) = 2$ is non-negative.
- The function $f(x) = \log(1 + \exp(x))$ is convex as $f'(x) = \frac{\exp(x)}{1+\exp(x)} = \frac{1}{\exp(-x)+1}$ is non-decreasing.

Claim: Let $S \subseteq \mathbb{R}^n$ be a convex set. Let $f : S \to \mathbb{R}$ be a twice continuously differentiable function on S and let $H_f(\mathbf{x})$ be the Hessian matrix of f at $\mathbf{x}$, If $H_f(\mathbf{x})$ is positive semi-definite for all $\mathbf{x} \in S$ then, f is convex on S.

- To see this, from the Taylor expansion of f, there exists some $\mathbf{z}$ where

$$f(\mathbf{u}) = f(\mathbf{w}) + \nabla f(\mathbf{w})^T(\mathbf{u} - \mathbf{w}) + \frac{1}{2}(\mathbf{u} - \mathbf{w})^T H_f(\mathbf{z})(\mathbf{u} - \mathbf{w})$$

.

- $H_f(\mathbf{z})$ being positive semi-definite implies that

$$f(\mathbf{u}) \geq f(\mathbf{w}) + \nabla f(\mathbf{w})^T(\mathbf{u} - \mathbf{w}),$$

as required.

Claim: (SSBD Claim 12.4) Assume that $f : \mathbb{R}^d \to \mathbb{R}$ can be written as $f(\mathbf{w}) = g(\langle \mathbf{w}, \mathbf{x} \rangle + y)$, for some $\mathbf{x} \in \mathbb{R}^d$,$y \in \mathbb{R}$, and $g : \mathbb{R} \to \mathbb{R}$. Then convexity of g implies convexity of f.
The claim implies that

- $f(\mathbf{w}) = (\langle \mathbf{w}, \mathbf{x} \rangle - y)^2$ is convex, and
- $f(\mathbf{w}) = \log(1 + \exp(-y \langle \mathbf{w}, \mathbf{x} \rangle))$ for $y \in \{-1, 1\}$ is convex.

Claim: (SSBD Claim 12.5) For $i = 1, \ldots, r$, let $f_i : \mathbb{R}^d \to \mathbb{R}$ be a convex function. The following functions from $\mathbb{R}^d$ to $\mathbb{R}$ are also convex.

- $g(\mathbf{x}) = \max_{i \in [r]} f_i(\mathbf{x})$
- $g(\mathbf{x}) = \sum_{i=1}^{r} w_i f_i(\mathbf{x})$, where for all i, $w_i \geq 0$.

This implies that

- $f(\mathbf{w}) = \max\{0, 1 - y\langle \mathbf{w}, \mathbf{x} \rangle\}$ for $y \in \{-1, 1\}$ is convex.
- $f(\mathbf{w}) = \sum_{i=1}^{m} \max\{0, 1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle\}$ for $y_i \in \{-1, 1\}$ is convex.

Exercise O1-5 (Archipelago):

With kernel methods, we often optimize a function of the form

$$\sum_{i=1}^{m} \ell\left(y_i, \sum_{j=1}^{m} \alpha_j K(\mathbf{x}_j, \mathbf{x}_i)\right) + \lambda \sum_{i,j=1}^{m} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$$

where ℓ is convex with respect to α.

Since the kernel $K(\mathbf{x}_j, \mathbf{x}_i)$ behaves like a similarity function, you decide to learn a similarity function and use it as a kernel in order to improve performance. Are there any issues if you use an arbitrary similarity function as the kernel?

Convex Lipschitz Problems

- We will look at convex Lipschitz problems. Recall the definition of Lipschitzness.
  Definition (Lipschitzness): Let $C \subset \mathbb{R}^d$. A function $f : \mathbb{R}^d \to \mathbb{R}^k$ is ρ-Lipschitz over C if for every $\mathbf{w}_1, \mathbf{w}_2 \in C$ we have that $\|f(\mathbf{w}_1) - f(\mathbf{w}_2)\| \leq \rho\|\mathbf{w}_1 - \mathbf{w}_2\|$.
  In this section, we use Lipschitzness with respect to the Euclidean norm.
- Recall Lipschitz loss functions. These are Lipschitz with respect to scalar input a.
  - With the hinge-loss (used e.g. in soft-margin support vector machine), the loss function $\phi(a, y) = \max\{0, 1 - ya\}$ is 1-Lipschitz with respect to $y \in \{-1, 1\}$.
  - The absolute loss $\phi(a, y) = |a - y|$ is 1-Lipshitz for all $y \in \mathbb{R}$.
  - The logistic loss $\phi(a, y) = \log_2(1 + e^{-ya})$ is $\frac{1}{\ln 2}$-Lipshitz for $y \in \{-1, 1\}$ as $\sup |\phi'(a)| = \sup_a \left| \frac{1}{\ln 2} \frac{-ye^{-ya}}{1+e^{-ya}} \right| = \frac{1}{\ln 2}$
  - The square loss $\phi(a, y) = (a - y)^2$ is $4B$-Lipschitz when $|a|, |y| \leq B$.

- The linear function $f(\mathbf{w}) = \langle \mathbf{w}, \mathbf{v} \rangle + b$, where $\mathbf{v} \in \mathbb{R}^d$ is $\|\mathbf{v}\|$-Lipschitz. Using the Cauchy-Schwartz inequality, we have

$$|f(\mathbf{w}_1) - f(\mathbf{w}_2)| = |\langle \mathbf{v}, \mathbf{w}_1 - \mathbf{w}_2 \rangle| \leq \|\mathbf{v}\| \|\mathbf{w}_1 - \mathbf{w}_2\|.$$

- We would like to compose the linear function with a loss function.
  Claim: (SSBD Claim 12.7) Let $f(\mathbf{w}) = g_1(g_2(\mathbf{w}))$, where g_1 is ρ_1-Lipschitz and g_2 is ρ_2-Lipschitz. Then f is $(\rho_1 \rho_2)$-Lipschitz. In particular, if g_2 is the linear function $g_2(\mathbf{w}) = \langle \mathbf{v}, \mathbf{w} \rangle + b$ for some $\mathbf{v} \in \mathbb{R}^d$, $b \in \mathbb{R}$, then f is $(\rho_1 \|\mathbf{v}\|)$-Lipschitz.
  Proof:

$$\begin{aligned} |f(\mathbf{w}_1) - f(\mathbf{w}_2)| &= |g_1(g_2(\mathbf{w}_1)) - g_1(g_2(\mathbf{w}_2))| \\ &\leq \rho_1 \|g_2(\mathbf{w}_1) - g_2(\mathbf{w}_2)\| \\ &\leq \rho_1 \rho_2 \|\mathbf{w}_1 - \mathbf{w}_2\|. \end{aligned}$$

Examples:

- Linear functions composed with
  - hinge loss $\ell(\mathbf{w}, \mathbf{x}, y) = \max\{0, 1 - y(\langle \mathbf{w}, \mathbf{x} \rangle + b)\}$
  - absolute loss $\ell(\mathbf{w}, \mathbf{x}, y) = |(\langle \mathbf{w}, \mathbf{x} \rangle + b) - y|$, and

  are all convex and $\|\mathbf{x}\|$-Lipschitz with respect to $\mathbf{w}$.
- Empirical risk function using these same loss functions $\frac{1}{m} \sum_{i=1}^{m} \ell(\mathbf{w}, \mathbf{x}_i, y_i)$ are all convex and R-Lipschitz with respect to $\mathbf{w}$, where $R = \max_i \|\mathbf{x}_i\|$.

Outline

Online Learning

- Stochastic gradient descent can be viewed as an online learning method.
- In online learning, an algorithm repeatedly does the following: outputs a prediction, receives the true label and suffers a loss for the prediction, updates the predictor using the label.
- Previously, we have analyzed *expected loss* of an algorithm under some probabilistic assumptions on the data generating mechanism.
- For online learning, we can often obtain stronger results for arbitrary sequences of inputs, even adversarially generated ones.

- For adversarial inputs, the error of the algorithm can be made arbitrarily large.
- Instead, we analyse the *regret* of the algorithm

$$\text{Regret}_A(\mathcal{H}, T) = \sum_{i=1}^{T} \ell_t(h^{(t)}, z_t) - \min_{h \in \mathcal{H}} \sum_{i=1}^{T} \ell_t(h, z_t),$$

  where $h^{(t)}$ is the hypothesis used by A at time t.
- We also overload the notation for the regret with respect to a particular hypothesis

$$\text{Regret}_A(h', T) = \sum_{i=1}^{T} \ell_t(h^{(t)}, z_t) - \sum_{i=1}^{T} \ell_t(h', z_t),$$

- The analysis is usually done for arbitrary sequence of inputs z_t, in contrast to the batch case. We may also allow a different ℓ_t for each t.

Realizable Case

- With the realizable case and $0-1$ loss, the regret is just the number of mistakes that the online algorithm makes.
- First consider the case of finite $\mathcal{H}$ and the following simple algorithm, *Consistent*:
  - Initialize $V_1 = \mathcal{H}$.
  - For each round t
    - Receive $\mathbf{x}_t$, randomly select $h \in V_t$ and use $h(\mathbf{x}_t)$ as prediction
    - Receive y_t and set $V_{t+1} = \{h \in V_t : h(x_t) = y_t\}$.
  - V_t contains hypotheses that agrees with all the labels so far and is often called the *version space*.

- *Consistent* removes at least one hypothesis from the version space each time an error is made.
  - In the realizable case, at least one hypothesis is never wrong, hence the mistake bound for *Consistent* is no more than $|\mathcal{H}| - 1$.
- Is this a good algorithm? Can we do better in the worst case?

Halving Algorithm

- Instead of randomly selecting $h \in V_t$ to do the prediction, predict using the majority prediction of the surviving hypotheses, $\arg\max_{r \in \{0,1\}} |\{h \in V_t : h(\mathbf{x}_t) = r\}|$.

- The halving algorithm has mistake bound of no more than $\log_2(|\mathcal{H}|)$.
  - Each error causes at least half of the version space to be removed: $|V_{t+1}| \leq |V_t|/2$.
  - After M mistakes $|V_{T+1}| \leq |\mathcal{H}|2^{-M}$.
  - Realizable, so $|V_{T+1}| \geq 1$, so $|\mathcal{H}|2^{-M} \geq 1$, giving $M \leq \log_2(|\mathcal{H}|)$.

Weighted Majority

- For the non-realizable case, we do not want to remove hypotheses that make mistakes totally.
- In the weighted majority algorithm, we initialize weight w_i for hypothesis i to 1 and multiply it by $1/2$ (other constants less than 1 would also work) when it makes a mistake (instead of multiply by 0 as in the halving algorithm).

- The weighted majority algorithm has mistake bound

$$M^{(T)} \leq \frac{1}{\lg 4/3}(m_i^{(T)} + \lg n)$$

for every i, where $M^{(T)}$ is mistake bound of weighted majority, and $m_i^{(T)}$ is the number of mistakes of hypothesis i.

Proof:

- With m_i mistakes, hypothesis i has weight $w_i^{(T)} = (1/2)^{m_i^{(T)}}$.

- Let $W^{(t)}$ be sum of weights at round t, with $W^{(1)} = n$.
- After each mistake, at least half the weight is multiplied by $1/2$, while the rest remains unchanged. So

$$W^{(t+1)} \leq \frac{3}{4} W^{(t)}.$$

- After $M^{(T)}$ mistakes, we have $W^{(T+1)} \leq n(3/4)^{M^{(T)}}$. But we also have $W^{(T+1)} \geq (1/2)^{m_i^{(T)}}$ giving

$$(1/2)^{m_i^{(T)}} \leq n(3/4)^{M^{(T)}}.$$

- Taking log and rearranging gives the result

$$M^{(T)} \leq \frac{1}{\lg 4/3}(m_i^{(T)} + \lg n).$$

□

Perceptron Algorithm

- The Perceptron algorithm can be considered an online learning algorithm for linear threshold functions in the realizable case.
  - Initialize $\mathbf{w}^{(1)} = (0, \ldots, 0)$
  - for $i = 1, 2, \ldots$
    - if $y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle \leq 0$ then
    - $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_i \mathbf{x}_i$

Theorem: Assume that there is a $\mathbf{w}$ with $\|\mathbf{w}\| \leq B$ such that $y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \geq 1$ and $\|\mathbf{x}_i\| \leq R$ for all $\mathbf{x}_i$. Then the Perceptron algorithm will make at most $(RB)^2$ mistakes.

Proof:

- We will show that after T updates (equivalently mistakes), $\langle \mathbf{w}, \mathbf{w}^{(T+1)} \rangle \geq T$ and $\|\mathbf{w}\| \|\mathbf{w}^{(T+1)}\| \leq \sqrt{T} RB$.
- By Cauchy-Schwartz, $\langle \mathbf{w}, \mathbf{w}^{(T+1)} \rangle \leq \|\mathbf{w}\| \|\mathbf{w}^{(T+1)}\|$. Combined with above, $T \leq \sqrt{T} RB$ giving the desired result.

- We now show $\langle \mathbf{w}, \mathbf{w}^{(T+1)} \rangle \geq T$.
  - $\mathbf{w}^{(1)}$ is initialized to $(0, \ldots, 0)$, so it holds for $T = 0$.
  - We have

$$\begin{aligned} \langle \mathbf{w}, \mathbf{w}^{(t+1)} \rangle - \langle \mathbf{w}, \mathbf{w}^{(t)} \rangle &= \langle \mathbf{w}, \mathbf{w}^{(t+1)} - \mathbf{w}^{(t)} \rangle \\ &= \langle \mathbf{w}, y_i \mathbf{x}_i \rangle = y_i \langle \mathbf{w}, \mathbf{x}_i \rangle \geq 1 \end{aligned}$$

  - Hence after T iterations

$$\langle \mathbf{w}, \mathbf{w}^{(T+1)} \rangle = \sum_{t=1}^{T} \left(\langle \mathbf{w}, \mathbf{w}^{(t+1)} \rangle - \langle \mathbf{w}, \mathbf{w}^{(t)} \rangle \right) \geq T.$$

- We now show that $\|\mathbf{w}^{(T+1)}\| \leq \sqrt{T}R$.

$$\begin{aligned}\|\mathbf{w}^{(t+1)}\|^2 &= \|\mathbf{w}^{(t)} + y_i\mathbf{x}_i\|^2 \\ &= \|\mathbf{w}^{(t)}\|^2 + 2y_i\langle\mathbf{w}^{(t)}, \mathbf{x}_i\rangle + y_i^2\|\mathbf{x}\|^2 \\ &\leq \|\mathbf{w}^{(t1)}\|^2 + R^2\end{aligned}$$

- Iterating the recurrence gives us $\|\mathbf{w}^{(T+1)}\|^2 \leq TR^2$ as required.

□

Follow-the-Leader

- A natural algorithm, called *follow-the-leader (FTL)*: predict using hypothesis that has done the best in the past

$$h_t = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{t-1} \ell_i(h, z_i)$$

- We can bound the performance of FTL using the following lemma

  Lemma: ([2] Lemma 2.1) Let $h_1, h_2, \ldots$ be the sequence of hypotheses produced by FTL. Then for any $h' \in \mathcal{H}$, we have

$$\sum_{t=1}^{T} \ell_t(h_t, z_t) - \ell_t(h', z_t) \leq \sum_{t=1}^{T} \ell_t(h_t, z_t) - \ell_t(h_{t+1}, z_t)$$

- The upper bound shows that the regret of FTL is bounded by $\sum_{t=1}^{T} \ell_t(h_t, z_t) - \ell_t(h_{t+1}, z_t)$, i.e. difference of FTL with another version of FTL that sees one step into the future.
- The term $\ell_t(h_t, z_t) - \ell_t(h_{t+1}, z_t)$ measures the *stability* of the algorithm.

Proof:

- Subtracting $\sum_{t=1}^{T} \ell_t(h_t, z_t)$ from both side, suffices to show

$$\sum_{t=1}^{T} \ell_t(h_{t+1}, z_t) \leq \sum_{t=1}^{T} \ell_t(h', z_t).$$

- Proof by induction. Base case $T = 1$ follows optimality of h_1. Assume inductive hypothesis holds for $T - 1$

$$\sum_{t=1}^{T-1} \ell_t(h_{t+1}, z_t) \leq \sum_{t=1}^{T-1} \ell_t(h', z_t).$$

- Add $\ell_T(h_{T+1}, z_T)$ to both sides to get

$$\sum_{t=1}^{T} \ell_t(h_{t+1}, z_t) \leq \ell_T(h_{T+1}, z_T) + \sum_{t=1}^{T-1} \ell_t(h', z_t).$$

- This holds for all h' including $h' = h_{T+1}$ giving

$$\sum_{t=1}^{T} \ell_t(h_{t+1}, z_t) \leq \sum_{t=1}^{T} \ell_t(h_{T+1}, z_t) = \min_{h \in \mathcal{H}} \sum_{t=1}^{T} \ell_t(h, z_t).$$

by definition of h_{T+1}. This concludes the inductive proof.

Example: Online Quadratic Optimization (FTL works well)

- At each round $\ell_t(\mathbf{w}, \mathbf{z}_t) = \frac{1}{2}\|\mathbf{w} - \mathbf{z}_t\|^2$ where $\mathbf{w} \in \mathbb{R}^d$.
- FTL is just the average of the observed targets

$$\mathbf{w}_t = \frac{1}{t-1} \sum_{i=1}^{t-1} \mathbf{z}_i.$$

- Rewrite

$$\mathbf{w}_{t+1} = \frac{1}{t}(\mathbf{z}_t + (t-1)\mathbf{w}_t) = \left(1 - \frac{1}{t}\right)\mathbf{w}_t + \frac{1}{t}\mathbf{z}_t,$$

giving

$$\mathbf{w}_{t+1} - \mathbf{z}_t = \left(1 - \frac{1}{t}\right)(\mathbf{w}_t - \mathbf{z}_t).$$

- Substituting

$$\begin{aligned}\ell_t(\mathbf{w}_t, \mathbf{z}_t) - \ell_t(\mathbf{w}_{t+1}, \mathbf{z}_t) &= \frac{1}{2}\|\mathbf{w}_t - \mathbf{z}_t\|^2 - \frac{1}{2}\|\mathbf{w}_{t+1} - \mathbf{z}_t\|^2 \\ &= \frac{1}{2}\left(1 - \left(1 - \frac{1}{t}\right)^2\right)\|\mathbf{w}_t - \mathbf{z}_t\|^2 \\ &= \frac{1}{t}\|\mathbf{w}_t - \mathbf{z}_t\|^2\end{aligned}$$

- Let $L = \max_t \|\mathbf{z}_t\|$. Then $\|\mathbf{w}_t\| \leq L$ as it is an average and $\|w_t - \mathbf{z}_t\| \leq 2L$, giving

$$\sum_{t=1}^{T}(\ell_t(\mathbf{w}_t, \mathbf{z}_t) - \ell_t(\mathbf{w}_{t+1}, \mathbf{z}_t)) \leq (2L)^2 \sum_{t=1}^{T}\frac{1}{t}.$$

- As $\sum_{t=1}^{T}\frac{1}{t} \leq \log(T) + 1$, regret of FTL is no more than $4L^2(\log(T) + 1)$.

Example: Online Linear Optimization (FTL not competitive)

- Let $w \in [-1, 1] \subset \mathbb{R}$ and consider loss that is a linear function in one dimension of the form $\ell_t(w, z_t) = wz_t$, where

$$z_t = \begin{cases} -0.5 & \text{if } t = 1 \\ 1 & \text{if } t \text{ is even} \\ -1 & \text{if } t \text{ is odd} \end{cases}$$

- FTL will set $w_t = 1$ for t odd and $w_t = -1$ otherwise giving cumulative loss of at least $T - 1$.
- Best value of w is 0, giving cumulative loss 0, so regret is at least $T - 1$.

- For quadratic optimization, the FTL predictors w_t get closer and closer together: stable, low regret
- For linear optimization, FTL do not get closer together: not stable, high regret
- Can we stabilize the linear case?
  - Add regularizer!

Follow-the-regularized-leader

- Add regularizer to get *follow-the-regularized-leader (FTRL)*: predict using hypothesis

$$h_t = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{t-1} \ell_i(h, z_i) + R(h)$$

- We can bound the performance of FTRL using the following lemma

  Lemma: ([2] Lemma 2.3) Let $h_1, h_2, \ldots$ be the sequence of hypotheses produced by FTRL. Then for any $h' \in \mathcal{H}$, we have

$$\sum_{t=1}^{T} \ell_t(h_t, z_t) - \ell_t(h', z_t) \leq R(h') - R(h_1) + \sum_{t=1}^{T} \ell_t(h_t, z_t) - \ell_t(h_{t+1}, z_t)$$

- *Proof:*
  - Running FTRL on $\ell_1(\cdot, z_1), \ldots, \ell_T(\cdot, z_T)$ is equivalent to running FTL on $\ell_0(\cdot, z_0), \ell_1(\cdot, z_1), \ldots, \ell_T(h_T, z_T)$ where $\ell_0(\cdot, z_0) = R(\cdot)$.
  - Applying [2] Lemma 2.1, we get

$$R(h_0) - R(h') + \sum_{t=1}^{T} \ell_t(h_t, z_t) - \ell_t(h', z_t) \leq R(h_0) - R(h_1) + \sum_{t=1}^{T} \ell_t(h_t, z_t) - \ell_t(h_{t+1}, z_t)$$

  - Canceling $R(h_0)$ on both sides and rearranging gives the result.

□

- We now reconsider online linear optimization using FTRL
  - We have linear function as the loss $\ell_t(\mathbf{w}, \mathbf{z}_t) = \langle \mathbf{w}, \mathbf{z}_t \rangle$ with $\mathbf{w} \in \mathbb{R}^d$.
  - Use quadratic regularizer $R(\mathbf{w}) = \frac{1}{2\eta}\|\mathbf{w}\|^2$.
  - For FTRL, we have

$$\begin{aligned} \mathbf{w}_t &= \arg\min_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{t-1} \langle \mathbf{w}, \mathbf{z}_i \rangle + \frac{1}{2\eta}\|\mathbf{w}\|^2 \\ &= \arg\min_{\mathbf{w}\in\mathbb{R}^d} \langle \mathbf{w}, \sum_{i=1}^{t-1} \mathbf{z}_i \rangle + \frac{1}{2\eta}\|\mathbf{w}\|^2 \end{aligned}$$

  - Differentiating and setting to zero, we get

$$\mathbf{w}_t = -\eta \sum_{i=1}^{t-1} \mathbf{z}_i$$

  - This can be computed online according to

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{z}_t; \qquad \mathbf{w}_t - \mathbf{w}_{t+1} = \eta \mathbf{z}_t$$

    which is the online gradient descent update.

- Continued ...
  - We can now bound the regret using [2] Lemma 2.3

$$\begin{aligned}
\text{Regret}_{\text{FTRL}}(\mathbf{w}^*, T) &\leq R(w^*) - R(w_1) + \sum_{t=1}^{T} \ell_t(h_t, z_t) - \ell_t(h_{t+1}, z_t) \\
&\leq \frac{1}{2\eta}\|\mathbf{w}^*\|^2 + \sum_{t=1}^{T} \langle \mathbf{w}_t - \mathbf{w}_{t+1}, \mathbf{z}_t \rangle \\
&= \frac{1}{2\eta}\|\mathbf{w}^*\|^2 + \eta \sum_{t=1}^{T} \|\mathbf{z}_t\|^2
\end{aligned}$$

- The first term can be viewed as squared bias, and smaller η corresponds to stronger regularization and smaller learning rate.
- The second term can be viewed as lower variance in terms of changing weights ($\mathbf{w}_{t+1} - \mathbf{w}_t = -\eta z_t$). Smaller η corresponds to being more stable.

The previous proof gives more intuition, through connection to regularization and FTRL. Here, we give a theorem from the textbook with slightly better bound next (proof in Appendix).

Lemma: (SSBD Lemma 14.1) Let $\mathbf{v}_1, \ldots, \mathbf{v}_T$ be an arbitrary sequence of vectors. Any algorithm with an initialization $\mathbf{w}^{(1)} = 0$ and an update rule of the form $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{v}_t$ satisfies

$$\sum_{t=1}^{T} (\mathbf{w}^{(t)} - \mathbf{w}^*)^T \mathbf{v}_t \leq \frac{\|\mathbf{w}^*\|^2}{2\eta} + \frac{\eta}{2} \sum_{t=1}^{T} \|\mathbf{v}_t\|^2$$

for any $\mathbf{w}^*$. In particular, for every $B, \rho > 0$, if for all t we have that $\|\mathbf{v}_t\| \leq \rho$ and if we set $\eta = \sqrt{\frac{B^2}{\rho^2 T}}$. Then for every $\mathbf{w}^*$, with $\|\mathbf{w}^*\| \leq B$ we have

$$\frac{1}{T} \sum_{t=1}^{T} (\mathbf{w}^{(t)} - \mathbf{w}^*)^T \mathbf{v}_t \leq \frac{B\rho}{\sqrt{T}}.$$

Online Gradient Descent for Convex Functions

- The method for linear optimization can be generalized to convex function of the parameter vector $\mathbf{w}$.
  - This allows the algorithm to be used for learning linear function with convex losses, e.g. linear regression with squared loss, or logistic regression.
- We assume that $\ell_t(\cdot, z_t)$ is a convex function for each t and $z_t = (\mathbf{x}_t, y_t)$ and do online gradient descent with a linear function.
- From convexity, we have

$$\ell_t(\mathbf{w}^{(t)}, z_t) - \ell_t(\mathbf{w}^*, z_t) \leq (\mathbf{w}^{(t)} - \mathbf{w}^*)^T \nabla \ell_t(\mathbf{w}^{(t)}, z_t),$$

where $\nabla \ell_t(\mathbf{w}^{(t)}, z_t)$ is the gradient or subgradient.

- Lemma (SSBD 14.1) shows that any algorithm that has update $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{v}_t$ satisfies

$$\sum_{t=1}^{T} (\mathbf{w}^{(t)} - \mathbf{w}^*)^T \mathbf{v}_t \leq \frac{\|\mathbf{w}^*\|^2}{2\eta} + \frac{\eta}{2} \sum_{t=1}^{T} \|\mathbf{v}_t\|^2$$

  for any $\mathbf{w}^*$ and an arbitrary sequence of vectors $\mathbf{v}_t$.
- It turns out that if $\ell_t(\cdot, z_t)$ is ρ-Lipschitz and $\mathbf{v}_t = \nabla \ell_t(\mathbf{w}^{(t)}, z_t)$ then $\|\mathbf{v}_t\| \leq \rho$.

Lemma: (SSBD Lemma 14.7) Let A be a convex open set and let $f : A \to R$ be a convex function. Then, f is ρ-Lipschitz over A iff for all $\mathbf{w} \in A$ and $\mathbf{v} \in \partial f(\mathbf{w})$ we have that $\|\mathbf{v}\| \leq \rho$.

See Appendix for proof.

- Combining Lemma (SSBD 14.1) and Lemma (SSBD 14.7): Assume that $\ell_t(\cdot, z_t)$ is ρ-Lipschitz, $\mathbf{v}_t$ are subgradients, $\mathcal{H}$ contains functions with weight norm bounded by B, then setting $\eta = \frac{B}{\rho\sqrt{T}}$, we get the following for online gradient descent

$$\text{Regret}_{SGD}(\mathcal{H}, T) \leq B\rho\sqrt{T}.$$

Batch Optimization/Gradient Descent

- We now consider gradient descent on the training set, i.e. $f(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} \ell(\mathbf{w}, z_i)$ is the empirical risk function.
- Running T steps of online gradient descent on the same $f(\mathbf{w})$ gives us the gradient descent algorithm.
- Consider using $\bar{\mathbf{w}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{w}^{(t)}$ as the final hypothesis.
- Let $\mathbf{w}^*$ be any vector (including the optimal vector)

$$
\begin{aligned}
f(\bar{\mathbf{w}}) - f(\mathbf{w}^*) &= f\left(\frac{1}{T}\sum_{t=1}^{T} \mathbf{w}^{(t)}\right) - f(\mathbf{w}^*) \\
&\leq \frac{1}{T}\sum_{t=1}^{T} f(\mathbf{w}^{(t)}) - f(\mathbf{w}^*) \\
&= \frac{1}{T}\sum_{t=1}^{T} (f(\mathbf{w}^{(t)}) - f(\mathbf{w}^*))
\end{aligned}
$$

- Because of convexity, (also hold for subgradient), $f(\mathbf{w}^{(t)}) - f(\mathbf{w}^*) \leq (\mathbf{w}^{(t)} - \mathbf{w}^*)^T \nabla f(\mathbf{w}^{(t)})$. Combining we get

$$f(\bar{\mathbf{w}}) - f(\mathbf{w}^*) \leq \frac{1}{T} \sum_{t=1}^{T} (\mathbf{w}^{(t)} - \mathbf{w}^*)^T \nabla f(\mathbf{w}^{(t)}).$$

Combining the results, we get

Corollary: (SSBD Corollary 14.2) Let f be a convex, ρ-Lipschitz function, and let $\mathbf{w}^* \in \arg\min_{\{\mathbf{w}:\|\mathbf{w}\|\leq B\}} f(\mathbf{w})$. If we run the GD algorithm on f for T steps with $\eta = \sqrt{\frac{B^2}{\rho^2 T}}$, then the output vector $\bar{\mathbf{w}}$ satisfies

$$f(\bar{\mathbf{w}}) - f(\mathbf{w}^*) \leq \frac{B\rho}{\sqrt{T}}.$$

Furthermore, for every $\epsilon > 0$, to achieve $f(\bar{\mathbf{w}}) - f(\mathbf{w}^*) \leq \epsilon$, it

suffices to run the GD algorithm for a number of iterations that satisfies

$$T \geq \frac{B^2\rho^2}{\epsilon^2}.$$

Example:

Gradient descent to minimize the empirical risk of linear function with the hinge loss and absolute loss, all satisfy

$$\frac{1}{m}\sum_{i=1}^{m}\ell(\bar{\mathbf{w}},\mathbf{x}_i,y_i)-\frac{1}{m}\sum_{i=1}^{m}\ell(\mathbf{w}^*,\mathbf{x}_i,y_i)\leq\frac{BR}{\sqrt{T}},$$

where $R=\max_i\|\mathbf{x}_i\|$. Furthermore, for every $\epsilon>0$, to achieve $\frac{1}{m}\sum_{i=1}^{m}\ell(\bar{\mathbf{w}},\mathbf{x}_i,y_i)-\frac{1}{m}\sum_{i=1}^{m}\ell(\mathbf{w}^*,\mathbf{x}_i,y_i)\leq\epsilon$, it suffices to run the GD algorithm for a number of iterations that satisfies

$$T\geq\frac{B^2R^2}{\epsilon^2}.$$

For hinge and absolute loss, we can bound the additional loss compared to optimal by $\frac{BR}{\sqrt{T}}$. Let's see the effect of feature scaling through an example:

Stochastic Gradient Descent

- We can relax the requirement that we obtain the (sub)-gradient at each step, and only require the expectation of $\mathbf{v}_t$ is in $\partial f(\mathbf{w}^{(t)})$, giving stochastic gradient descent (SGD).
- The corresponding bound holds in expectation.
- Examples include gradient of a single randomly selected example, or a mini-batch.

Theorem: (SSBD Theorem 14.8) Let $B, \rho > 0$. Let f be a convex function and let $\mathbf{w}^* \in \arg\min_{\mathbf{w}:\|w\| \leq B} f(\mathbf{w})$. Assume that SGD is run for T iterations with $\eta = \sqrt{\frac{B^2}{\rho^2 T}}$. Assume also that for all t, $\|\mathbf{v}_t\| \leq \rho$ with probability 1. Then,

$$E[f(\bar{\mathbf{w}})] - f(\mathbf{w}^*) \leq \frac{B\rho}{\sqrt{T}}.$$

Therefore, for any $\epsilon > 0$, to achieve $E[f(\bar{\mathbf{w}})] - f(\mathbf{w}^*) \leq \epsilon$, it suffices to run the SGD algorithm for a number of iterations that satisfies

$$T \geq \frac{B^2 \rho^2}{\epsilon^2}.$$

Proof skipped.

- If data is not repeated, SGD is optimizing the expected loss directly.
- Immediately gives generalization bound.
- Which is faster, SGD or GD?
  - Without more regularity conditions, the expected rate of convergence for SGD appears to be the same as the rate of convergence of GD
  - Theoretically not much advantage in batch optimization.
  - Each gradient computation in batch optimization requires going through the whole dataset: GD is computationally more expensive.

Variants

- Let $\mathcal{H} = \{\mathbf{w} : \|\mathbf{w}\| \leq B\}$. We can restrict the hypothesis to $\mathcal{H}$ by adding a projection step after every gradient update step: replace $\mathbf{w}$ by the value in $\mathcal{H}$ closest to it
  1. $\mathbf{w}^{(t+\frac{1}{2})} = \mathbf{w}^{(t)} - \eta \mathbf{v}_t$
  2. $\mathbf{w}^{(t+1)} = \arg\min_{\mathbf{w} \in \mathcal{H}} \|\mathbf{w} - \mathbf{w}^{(t+\frac{1}{2})}\|$

  The performance bound still holds.

- Similar performance bounds also hold with variable learning rate $\eta_t = \frac{B}{\rho\sqrt{t}}$.

Strongly Convex Function

Definition (Strongly Convex Function): (SSBD Definition 13.4) A function f is λ-strongly convex if for all $\mathbf{w}, \mathbf{u}$ and $\alpha \in (0,1)$ we have

$$f(\alpha \mathbf{w} + (1-\alpha)\mathbf{u}) \leq \alpha f(\mathbf{w}) + (1-\alpha) f(\mathbf{u}) - \frac{\lambda}{2}\alpha(1-\alpha)\|\mathbf{w}-\mathbf{u}\|^2.$$

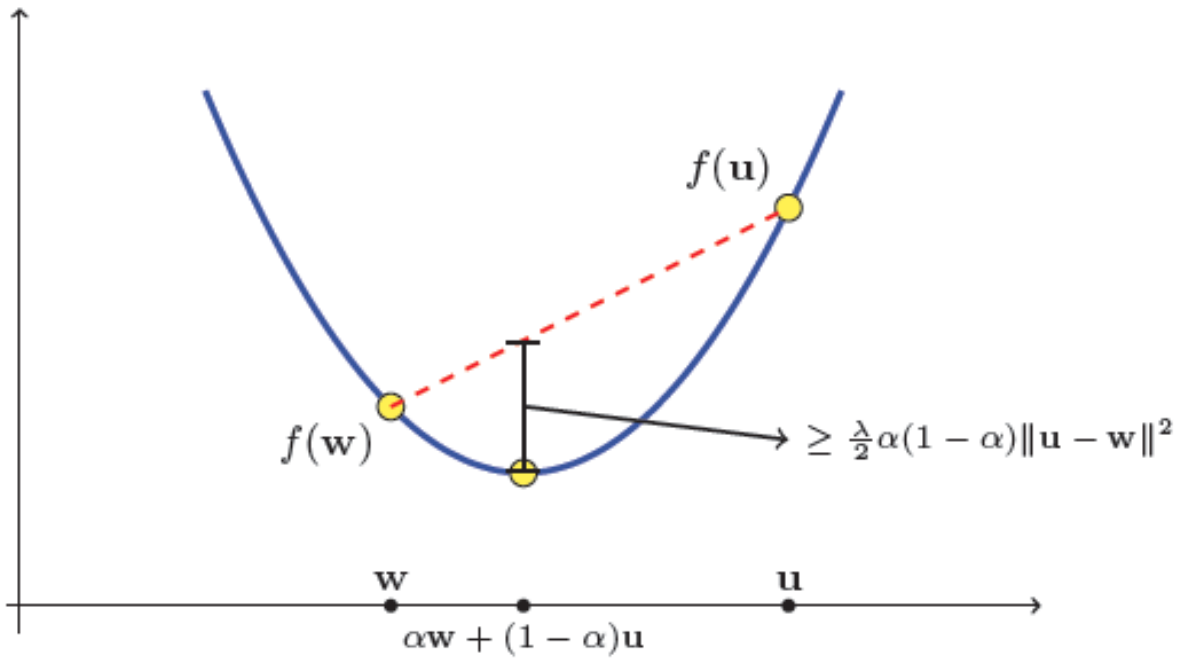


Figure from SSBD.

- For differentiable function equivalent definition is

$$f(\mathbf{u}) \geq f(\mathbf{w}) + \nabla f(\mathbf{w})^T(\mathbf{u} - \mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w} - \mathbf{u}\|^2.$$

- If f is twice continuous differentiable, then the smallest eigenvalue of the Hessian matrix must be at least λ.

Lemma: (SSBD Lemma 13.5)

1. The function $f(w) = \lambda\|\mathbf{w}\|^2$ is 2λ-strongly convex.
2. If f is λ-strongly convex and g is convex, then $f + g$ is λ-strongly convex.

The lemma shows that Regularized Loss Minimization with a convex loss and Tikhonov regularization, i.e. minimizing $L_S(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|^2$ is λ strongly convex.

SGD with λ-strongly convex function, $\eta_t = 1/(\lambda t)$ and satisfies a faster convergence rate:

Theorem: (SSBD Theorem 14.11) Assume that f is λ-strongly convex and that $E[\|\mathbf{v}_t\|^2] \leq \rho^2$. Let $\mathbf{w}^* \in \arg\min_{\mathbf{w}\in\mathcal{H}} f(\mathbf{w})$ be an optimal solution. Then,

$$E[f(\bar{\mathbf{w}})] - f(\mathbf{w}^*) \leq \frac{\rho^2}{2\lambda T}(1 + \log(T)).$$

Remark: For some variants, the $\log(T)$ term can be eliminated.

GD, Newton's method, and SGD on Strongly Convex Functions

For strongly convex functions:

- Gradient descent has linear convergence, i.e. $\lim_{t\to\infty} \frac{\epsilon_{t+1}}{\epsilon_t} = \mu < 1$.
  - Correspond to $O(\log(1/\epsilon))$ iterations to achieve error ϵ.
- Newton's method has quadratic convergence, i.e. $\lim_{t\to\infty} \frac{\epsilon_{t+1}}{(\epsilon_t)^2} \leq M$ for some constant M.
  - Correspond to $O(\log\log(1/\epsilon))$ iterations to achieve error ϵ. Each additional (constant number of) iteration would double the number of bits of precision.
- Stochastic gradient descent converges sublinearly, i.e. $\lim_{t\to\infty} \frac{\epsilon_{t+1}}{\epsilon_t} = 1$.
  - SGD requires $O(1/\epsilon)$ to achieve error ϵ.

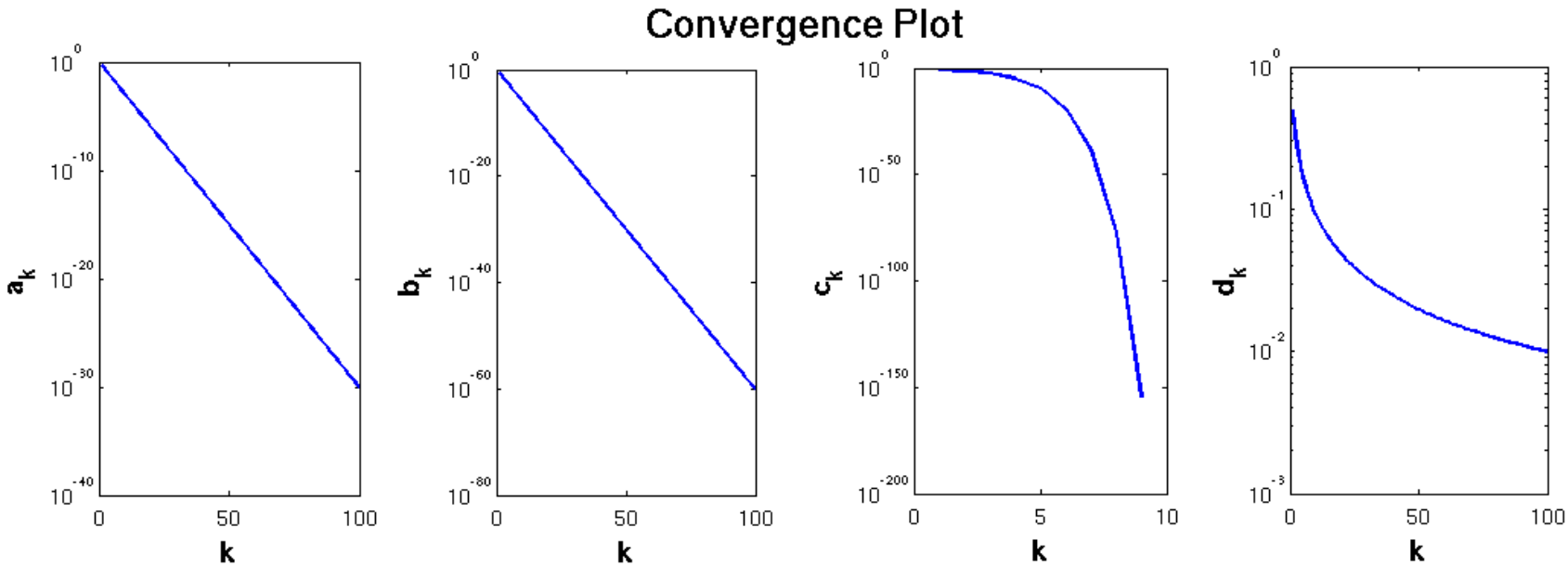


Figure: From Wikipedia. First two plots show linear convergence, the third shows quadratic, the last sublinear convergence. Note log scale on vertical axis.

Exercise O1-6 (Archipelago):
For strongly convex problems, stochastic gradient has sublinear convergence, gradient descent has linear convergence and second order methods such as Newton's method has quadratic convergence.
Hence, for practical machine learning, Newton's method should be preferred to gradient descent, which should be preferred to stochastic gradient descent. True or False? What else should be considered?

SGD vs GD for Strongly Convex Functions

- Runtime for SGD:
  - SGD requires $O(1/\epsilon)$ iterations to achieve expected error ϵ.
  - Each iteration requires constant time.
  - Total time to achieve expected error ϵ is $O(1/\epsilon)$.
- Runtime for GD:
  - For strongly convex functions, GD requires $O(\log(1/\epsilon))$ iterations
    - The constant depends on the eigenvalues of the Hessian.
    - This is called linear convergence – straight line when error is plotted on a log scale as a function of iterations. Each additional (constant number of) iteration would halve the optimization error.
  - But each iteration requires $O(m)$ time to compute the gradient, where m is the number of training examples.
  - Total time is $O(m \log(1/\epsilon))$.

- Which is better in practice ($O(1/\epsilon)$ vs $O(m \log(1/\epsilon))$?
  - Depends on m and ϵ.
  - If m is small, GD may be better.
  - If m is very large and high accuracy is not necessary, SGD may be faster.
  - Machine learning problems may not require very high accuracy.
    - Estimation error is around $O(1/m)$ and $O(1/\sqrt{m})$ ignoring log factors.
    - Can monitor error on validation set and stop when it increases (early stopping is also a form of regularization).

- What about second order methods?
  - When close to the solution, convergence is quadratic – number of iterations required is $O(\log\log(1/\epsilon))$. Each additional (constant number of) iteration would double the number of bits of precision.
  - But each iteration takes $O(m)$ time.
  - Worse, Hessian matrix size is quadratic in the number of parameters. Approximate when number of parameters is large, e.g. LBFGS, conjugate gradient.
  - May be better for when m is small, but tradeoff similar to GD when m large.

Smoothness

A continuously differentiable function f is β-smooth if the ∇f is β-Lipshitz, i.e.

$$\|\nabla f(\mathbf{w}) - \nabla f(\mathbf{v})\| \leq \beta \|\mathbf{w} - \mathbf{v}\|.$$

- For convex β-smooth functions, gradient descent has a $\epsilon = O(1/t)$ convergence rate.
- This can be improved to $O(1/t^2)$ using acceleration methods, e.g. Nesterov momentum.

Reading

1. SSBD Chapters 8, 12, 14, 15, 16.
2. Goodfellow, Ian, Yoshua Bengio, and Aaron Courville, "Deep Learning", `http://www.deeplearningbook.org/`. Chapters 4, 6, and 8.

References I

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[2] Shai Shalev-Shwartz et al. "Online learning and online convex optimization". In: *Foundations and Trends® in Machine Learning* 4.2 (2012), pp. 107–194.

Outline

Proofs for Convex Problems

Lemma: (SSBD Lemma 14.1) Let $\mathbf{v}_1, \ldots, \mathbf{v}_T$ be an arbitrary sequence of vectors. Any algorithm with an initialization $\mathbf{w}^{(1)} = 0$ and an update rule of the form $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{v}_t$ satisfies

$$\sum_{t=1}^{T} (\mathbf{w}^{(t)} - \mathbf{w}^*)^T \mathbf{v}_t \leq \frac{\|\mathbf{w}^*\|^2}{2\eta} + \frac{\eta}{2} \sum_{t=1}^{T} \|\mathbf{v}_t\|^2$$

for any $\mathbf{w}^*$. In particular, for every $B, \rho > 0$, if for all t we have that $\|\mathbf{v}_t\| \leq \rho$ and if we set $\eta = \sqrt{\frac{B^2}{\rho^2 T}}$. Then for every $\mathbf{w}^*$, with $\|\mathbf{w}^*\| \leq B$ we have

$$\frac{1}{T} \sum_{t=1}^{T} (\mathbf{w}^{(t)} - \mathbf{w}^*)^T \mathbf{v}_t \leq \frac{B\rho}{\sqrt{T}}.$$

Proof:

- By algebriac manipulation, we have

$$
\begin{aligned}
(\mathbf{w}^{(t)} - \mathbf{w}^*)^T \mathbf{v}_t &= \frac{1}{\eta}(\mathbf{w}^{(t)} - \mathbf{w}^*)^T \eta \mathbf{v}_t \\
&= \frac{1}{2\eta}(-\|\mathbf{w}^{(t)} - \mathbf{w}^* - \eta \mathbf{v}_t\|^2 + \|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2 + \eta^2 \|\mathbf{v}_t\|^2) \\
&= \frac{1}{2\eta}(-\|\mathbf{w}^{(t+1)} - \mathbf{w}^*\|^2 + \|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2) + \frac{\eta}{2}\|\mathbf{v}_t\|^2,
\end{aligned}
$$

where the last inequality comes from the update rule.

- Summing over t

$$\sum_{t=1}^{T}(\mathbf{w}^{(t)} - \mathbf{w}^*)^T \mathbf{v}_t$$

$$= \frac{1}{2\eta}\sum_{t=1}^{T}(-\|\mathbf{w}^{(t+1)} - \mathbf{w}^*\|^2 + \|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2) + \frac{\eta}{2}\sum_{t=1}^{T}\|\mathbf{v}_t\|^2.$$

- The first part on the right telescopes to get $\|\mathbf{w}^{(1)} - \mathbf{w}^*\|^2 - \|\mathbf{w}^{(T+1)} - \mathbf{w}^*\|^2$.

- Plugging back in

$$\sum_{t=1}^{T}(\mathbf{w}^{(t)} - \mathbf{w}^*)^T\mathbf{v}_t$$

$$= \frac{1}{2\eta}(\|\mathbf{w}^{(1)} - \mathbf{w}^*\|^2 - \|\mathbf{w}^{(T+1)} - \mathbf{w}^*\|^2) + \frac{\eta}{2}\sum_{t=1}^{T}\|\mathbf{v}_t\|^2$$

$$\leq \frac{1}{2\eta}\|\mathbf{w}^{(1)} - \mathbf{w}^*\|^2 + \frac{\eta}{2}\sum_{t=1}^{T}\|\mathbf{v}_t\|^2$$

$$\leq \frac{1}{2\eta}\|\mathbf{w}^*\|^2 + \frac{\eta}{2}\sum_{t=1}^{T}\|\mathbf{v}_t\|^2,$$

where we have used $\|\mathbf{w}^{(1)}\| = 0$.

- The second part follows from bounding $\|\mathbf{w}^*\|$ with B, $\|\mathbf{v}_t\|$ by ρ, dividing by T and plugging in the value for η. $\square$

Lemma: (SSBD Lemma 14.7) Let A be a convex open set and let $f : A \to R$ be a convex function. Then, f is ρ-Lipschitz over A iff for all $\mathbf{w} \in A$ and $\mathbf{v} \in \partial f(\mathbf{w})$ we have that $\|\mathbf{v}\| \leq \rho$.

Proof: Assume $\mathbf{v} \in \partial f(\mathbf{w})$ and $\|\mathbf{v}\| \leq \rho$. By defn of subgradient, for all $\mathbf{u}$

$$f(\mathbf{w}) - f(\mathbf{u}) \leq \langle \mathbf{v}, \mathbf{w} - \mathbf{u} \rangle.$$

Using Cauchy-Schwatz, we get

$$f(\mathbf{w}) - f(\mathbf{u}) \leq \langle \mathbf{v}, \mathbf{w} - \mathbf{u} \rangle \leq \|\mathbf{v}\| \|\mathbf{w} - \mathbf{u}\| \leq \rho \|\mathbf{w} - \mathbf{u}\|.$$

Can similarly show $f(\mathbf{u}) - f(\mathbf{w}) \leq \rho \|\mathbf{w} - \mathbf{u}\|$, hence f is ρ-Lipschitz.

We now assume that f is ρ-Lipschitz and show that $\|\mathbf{v}\| \leq \rho$.

- Let $\mathbf{u} = \mathbf{w} + \epsilon\mathbf{v}/\|\mathbf{v}\|$. Then $(\mathbf{u} - \mathbf{w})^T\mathbf{v} = \epsilon\|\mathbf{v}\|$ and $\|\mathbf{u} - \mathbf{w}\| = \epsilon$.
- From definition of subgradient

$$f(\mathbf{u}) - f(\mathbf{w}) \geq \mathbf{v}^T(\mathbf{u} - \mathbf{w}) = \epsilon\|\mathbf{v}\|.$$

- From Lipschitzness, we have

$$\rho\epsilon = \rho\|\mathbf{u} - \mathbf{w}\| \geq f(\mathbf{u}) - f(\mathbf{w}).$$

- Combining the inequalities, we get $\|\mathbf{v}\| \leq \rho$.

□

Strong Convexity

Theorem: (SSBD Theorem 14.11) Assume that f is λ-strongly convex and that $E[\|\mathbf{v}_t\|^2] \leq \rho^2$. Let $\mathbf{w}^* \in \arg\min_{\mathbf{w}\in\mathcal{H}} f(\mathbf{w})$ be an optimal solution. Then,

$$E[f(\bar{\mathbf{w}})] - f(\mathbf{w}^*) \leq \frac{\rho^2}{2\lambda T}(1 + \log(T)).$$

Proof:

- Let $\nabla^{(t)} = E[\mathbf{v}_t|\mathbf{w}^{(t)}]$. From definition of strong convexity and $\nabla^{(t)}$ being a subgradient

$$\langle \mathbf{w}^{(t)} - \mathbf{w}^*, \nabla^{(t)} \rangle \geq f(\mathbf{w}^{(t)}) - f(\mathbf{w}^*) + \frac{\lambda}{2}\|\mathbf{w}^{(t)} - \mathbf{w}^*\|.$$

- From the proof of SSBD Lemma 14.1, we have

$$\langle \mathbf{w}^{(t)} - \mathbf{w}^*, \nabla^{(t)} \rangle \leq \frac{1}{2\eta_t} E[\|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2 - \|\mathbf{w}^{(t+1)} - \mathbf{w}^*\|^2] + \frac{\eta_t}{2}\|\mathbf{v}_t\|^2.$$

- Combining the two, taking expectation, summing over t, and using $E[\|\mathbf{v}_t\|^2] \leq \rho^2$, we get

$$\sum_{t=1}^{T}(E[f(\mathbf{w}^{(t)})] - f(\mathbf{w}^*)) \leq E\left[\sum_{t=1}^{T}\left(\frac{1}{2\eta_t}E[\|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2 - \|\mathbf{w}^{(t+1)} - \mathbf{w}^*\|^2] - \frac{\lambda}{2}\|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2\right)\right] + \frac{\rho^2}{2}\sum_{t=1}^{T}\eta_t.$$

- Applying $\eta_t = 1/(\lambda t)$, the first sum on the right becomes $-\frac{\lambda T}{2}\|\mathbf{w}^{(T+1)} - \mathbf{w}^*\|^2 \leq 0$.

- This gives

$$\sum_{t=1}^{T}(E[f(\mathbf{w}^{(t)})] - f(\mathbf{w}^*)) \leq \frac{\rho^2}{2\lambda}\sum_{t=1}^{T}\frac{1}{t} \leq \frac{\rho^2}{2\lambda}(1+\log(T)).$$

- Dividing by T and applying Jensen's inequality gives the theorem.

□