Parisian Master of Research in Computer Science

Pablo Piantanida
pablo.piantanida@gmail.com

Laboratoire de Signaux et Systemes (L2S)
CentraleSupelec-CNRS-Universite Paris-Saclay, France

CentraleSupélec

Lecture 1

Stochastic optimization in High-Dimensions

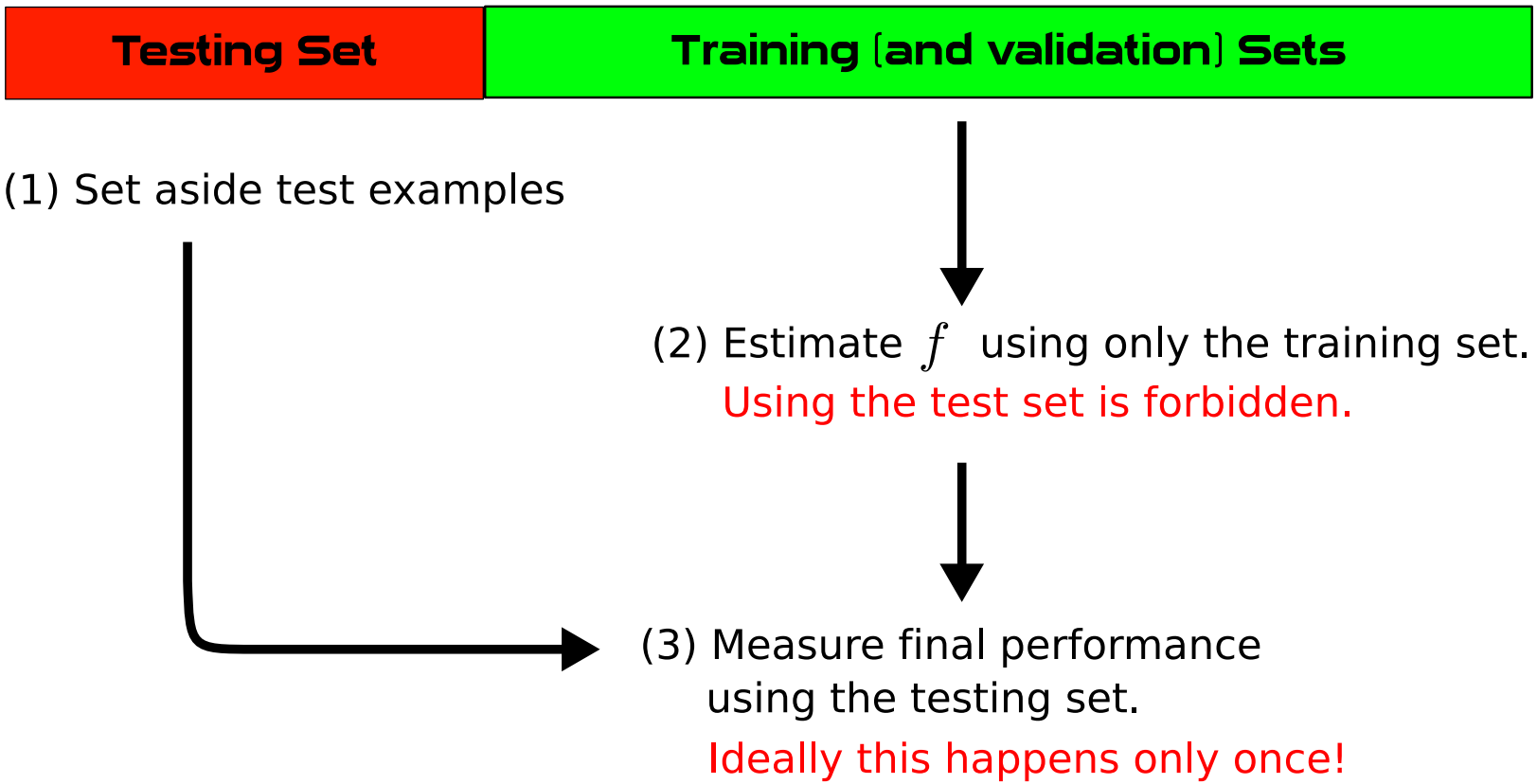
OUTLINE

- Statistical and computational tradeoffs
- Review of Convex Functions
- Gradient Descent (GD)
- Stochastic Gradient Descent (SGD + GSD)
- SDG with Mini-Batching
- Regularization

OUTLINE

- Statistical and computational tradeoffs
- Review of Convex Functions
- Gradient Descent (GD)
- Stochastic Gradient Descent (SGD + GSD)
- SDG with Mini-Batching
- Regularization

THE PRACTICAL PARADIGM



Variations: k-fold cross-validation, etc.

This is the main driver for progress in machine learning.

MATHEMATICAL FORMULATION

- **Assumption**
  Examples are drawn independently from an unknown probability distribution $P(x, y)$ that represents the laws of Nature.

- **Loss Function**
  Function $\ell(\hat{y}, y)$ measures the cost of answering $\hat{y}$ when the true answer is y.

- **Expected Risk**
  We seek to find the function f^* that minimizes:

$$\min_f \quad E(f) = \int \ell(\, f(x), y \,)\; dP(x, y)$$

Note: The test set error is an approximation of the expected risk.

MATHEMATICAL FORMULATION

- **Approximation**

  Not feasible to search f^* among all functions.

  Instead, we search $f^*_{\mathcal{F}}$ that minimizes the Expected Risk $E(f)$ within some richly parametrized family of functions $\mathcal{F}$.

- **Estimation**

  Not feasible to minimize the expectation $E(f)$ because $P(x, y)$ is unknown.

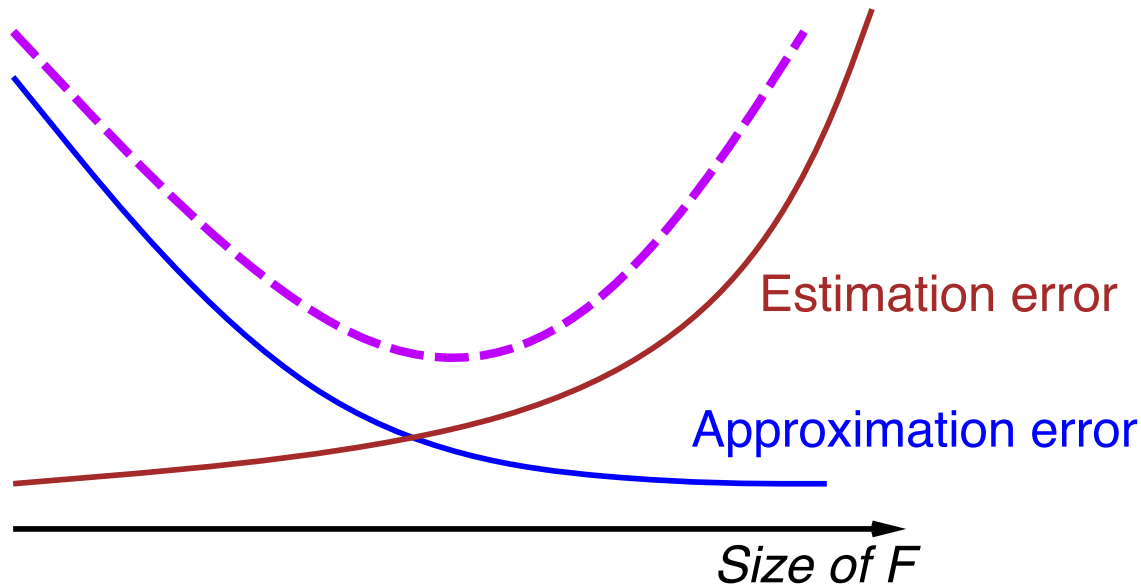
  Instead, we search f_n that minimizes the Empirical Risk $E_n(f)$, that is, the average loss over the training set examples.

$$\min_{f \in \mathcal{F}} \quad E_n(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(\, f(x_i), y_i \,)$$

In other words, we optimize a surrogate problem!

TRADITIONAL TRADEOFF

$$E(f_n) - E(f^*) = \big(E(f_F^*) - E(f^*)\big) \qquad \text{Approximation Error}$$
$$+ \big(E(f_n) - E(f_{\mathcal{F}}^*)\big) \qquad \text{Estimation Error}$$



(Vapnik and Chervonenkis, Ordered risk minimization, 1974).
(Vapnik and Chervonenkis, Theorie der Zeichenerkennung, 1979)

STATISTICAL AND COMPUTATIONAL PERSPECTIVES

- **Statistical Perspective:**
"It is good to optimize an objective function than ensures a fast estimation rate when the number of examples increases."

- **Optimization Perspective:**
"To efficiently solve large problems, it is preferable to choose an optimization algorithm with strong asymptotic properties, e.g. superlinear."

- **Incorrect Conclusion:**
"To address large-scale learning problems, use a superlinear algorithm to optimize an objective function with fast estimation rate.

COMPUTATIONAL CONSTRAINTS

- Baseline large-scale learning algorithm

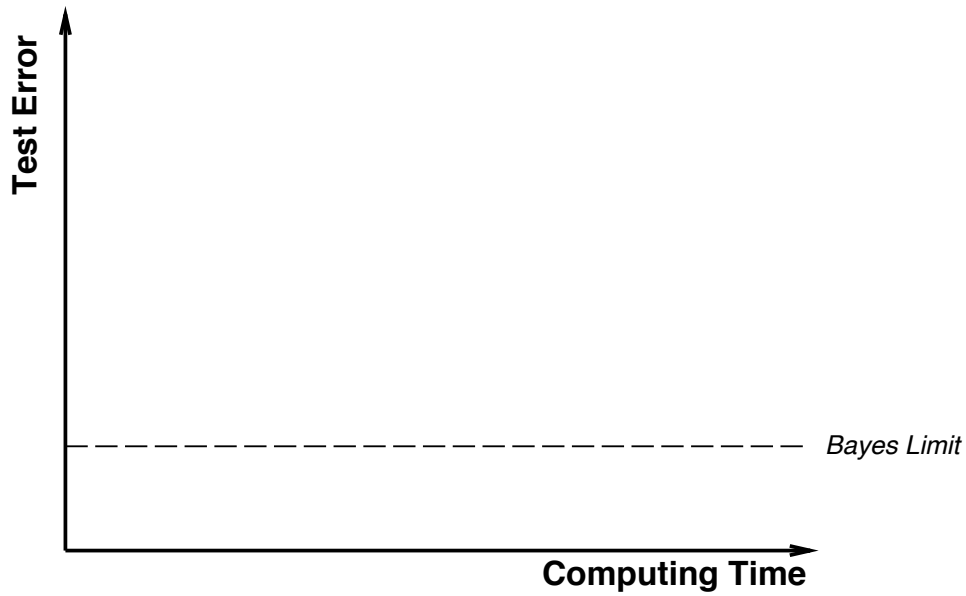
Randomly discarding data is the simplest way to handle large datasets.

  – What are the statistical benefits of processing more data?
  – What is the computational cost of processing more data?

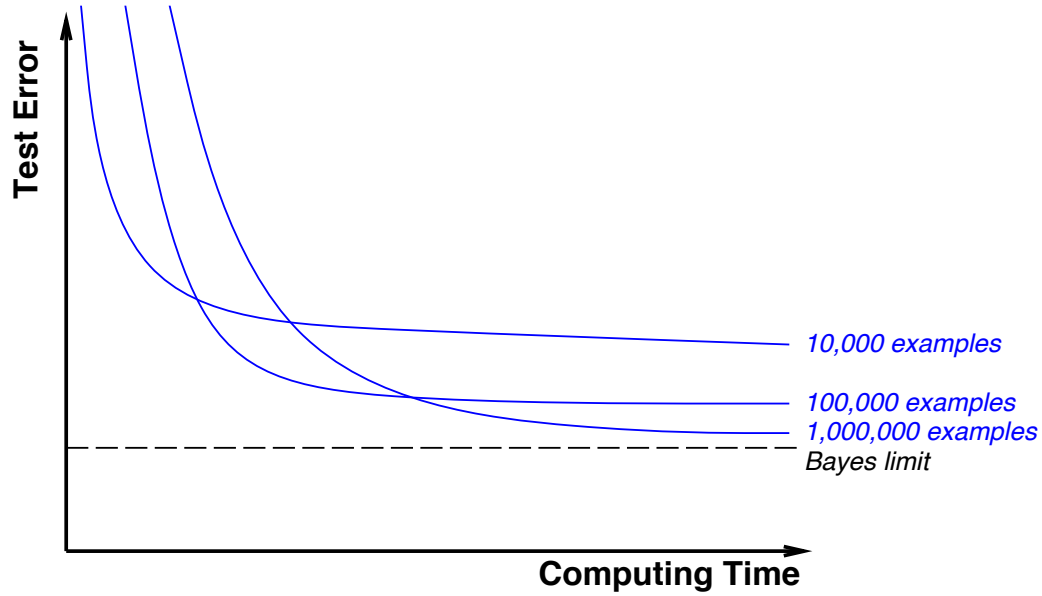
- We need a theory that joins Statistics and Computation!

– 1967: Vapnik and Chervonenkis theory does not discuss computation.
– 1981: Valiant's learnability excludes exponential time algorithms, but (i) polynomial time already too slow, (ii) few actual results.

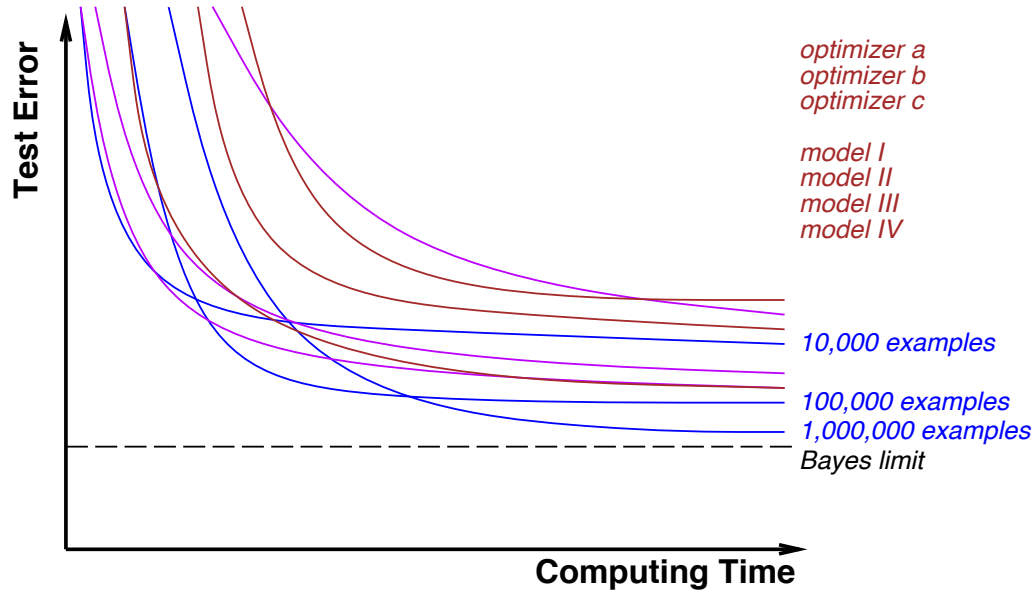
TRADEOFF BETWEEN LOSS AND LEARNING TIME



TRADEOFF BETWEEN LOSS AND LEARNING TIME

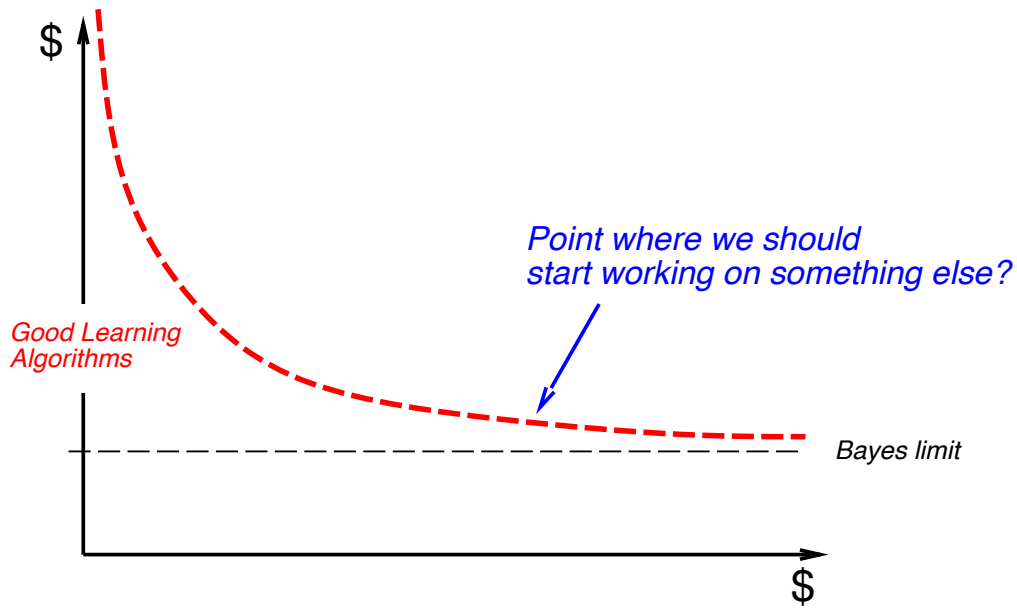


TRADEOFF BETWEEN LOSS AND LEARNING TIME



Vary the number of examples, the statistical models, the algorithms,. . .

TRADEOFF BETWEEN LOSS AND LEARNING TIME



Changing the units along the axes. . .

LEARNING WITH APPROXIMATE OPTIMIZATION

Computing $f_n = \arg\min_{f \in \mathcal{F}} E_n(f)$ is often costly.

Since we already optimize a **surrogate** function
why should we compute its optimum f_n exactly?

Let's assume our optimizer returns $\tilde{f}_n$
such that $E_n(\tilde{f}_n) < E_n(f_n) + \rho$.

For instance, one could stop an iterative
optimization algorithm long before its convergence.

LEARNING WITH APPROXIMATE OPTIMIZATION

$$
\begin{aligned}
E(\tilde{f}_n) - E(f^*) &= E(f^*_{\mathcal{F}}) - E(f^*) && \text{Approximation error} \\
&+ E(f_n) - E(f^*_{\mathcal{F}}) && \text{Estimation error} \\
&+ E(\tilde{f}_n) - E(f_n) && \text{Optimization error}
\end{aligned}
$$

Problem:

Choose $\mathcal{F}$, n, and ρ to make this as small as possible,

subject to budget constraints $\left\{ \begin{array}{l} \text{max number of examples } n \\ \text{max computing time } T \end{array} \right.$

LEARNING WITH APPROXIMATE OPTIMIZATION

Approximation error bound: (Approximation theory)
– decreases when $\mathcal{F}$ gets larger.

Estimation error bound: (Vapnik-Chervonenkis theory)
– decreases when n gets larger.
– increases when $\mathcal{F}$ gets larger.

Optimization error bound: (Vapnik-Chervonenkis theory plus tricks)
– increases with ρ.

Computing time T: (Algorithm dependent)
– decreases with ρ
– increases with n
– increases with $\mathcal{F}$

LEARNING WITH APPROXIMATE OPTIMIZATION

We can give *rigorous definitions*.

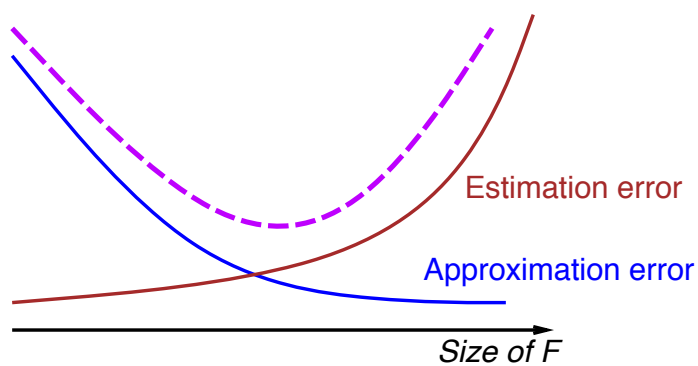
- **Definition 1:**
  We have a **small-scale learning** problem when the **active budget constraint is the number of examples n.**

- **Definition 2:**
  We have a **large-scale learning** problem when the **active budget constraint is the computing time T.**

LEARNING WITH APPROXIMATE OPTIMIZATION

The active budget constraint is the number of examples.

- To reduce the estimation error, take n as large as the budget allows.
- To reduce the optimization error to zero, take $\rho = 0$.
- We need to adjust the size of $\mathcal{F}$.



See Structural Risk Minimization (Vapnik 74) and later works.

OUTLINE

- Statistical and computational tradeoffs
- Review of Convex Functions
- Gradient Descent (GD)
- Stochastic Gradient Descent (SGD + GSD)
- SDG with Mini-Batching
- Regularization

OPTIMIZATION

- Much of machine learning can be written as an optimization problem

loss function

$$\min_{x} \sum_{i=1}^{N} f(x; y_i)$$

model

training examples

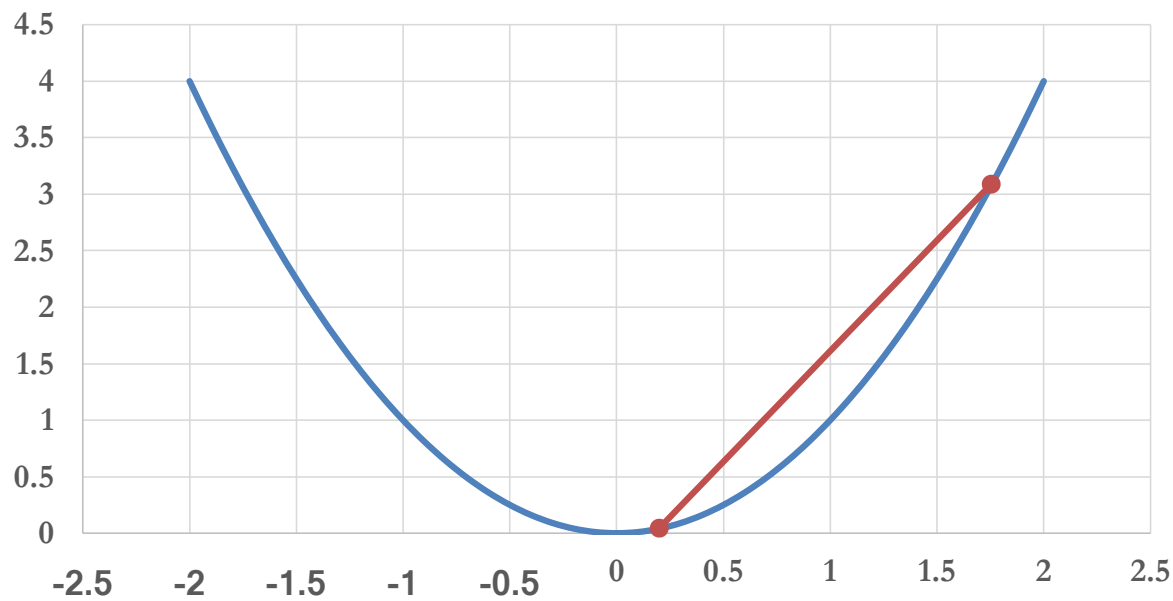
- Example loss functions: logistic regression, linear regression, principle component analysis, neural network loss

TYPES OF OPTIMIZATION

- Convex optimization
  - The **easy case**
  - Includes logistic regression, linear regression, SVM
- Non-convex optimization
  - **NP-hard in general**
  - Includes deep learning

REVIEW OF CONVEX FUNCTIONS

$$\forall \alpha \in [0,1],\ f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y)$$

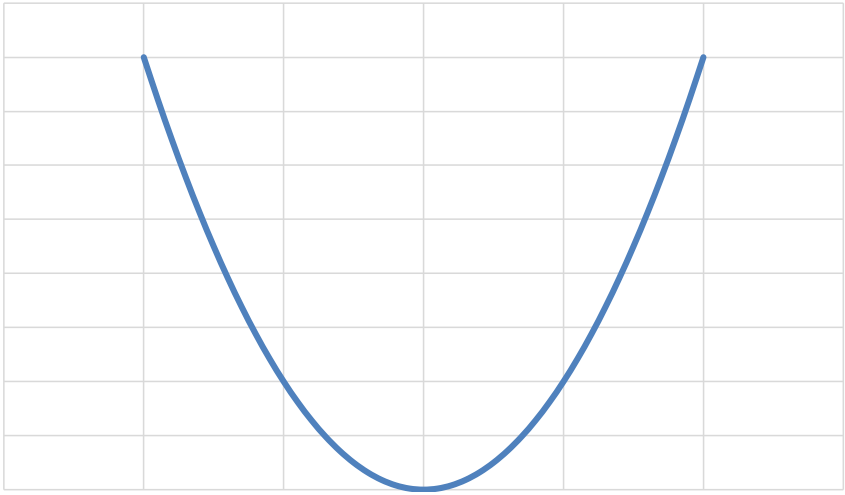


$f(x) = x^2$

REVIEW OF CONVEX FUNCTIONS

Example: Quadratic

$$f(x) = x^2$$

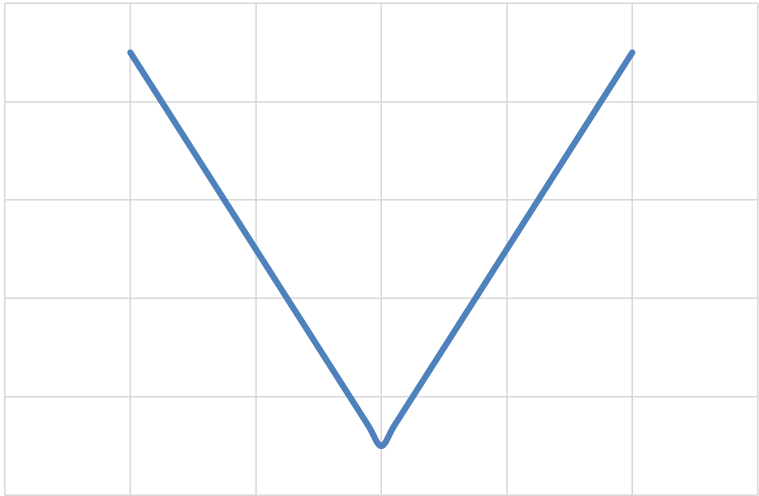


$$\begin{aligned}(\alpha x + (1-\alpha)y)^2 &= \alpha^2 x^2 + 2\alpha(1-\alpha)xy + (1-\alpha)^2 y^2 \\ &= \alpha x^2 + (1-\alpha)y^2 - \alpha(1-\alpha)(x^2 + 2xy + y^2) \\ &\leq \alpha x^2 + (1-\alpha)y^2\end{aligned}$$

REVIEW OF CONVEX FUNCTIONS

Example: Abs

$$f(x) = |x|$$

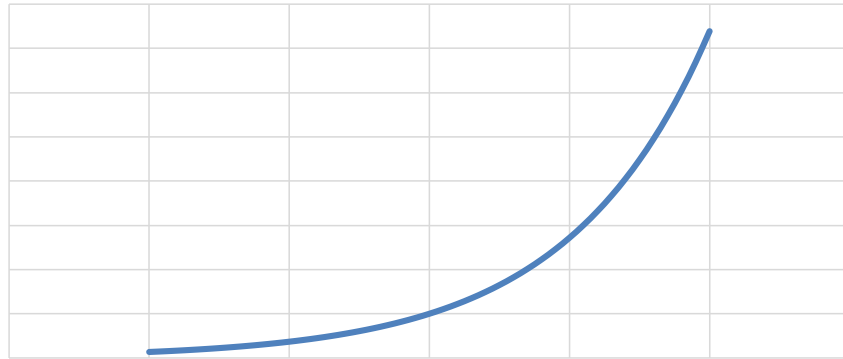


$$\begin{aligned} |\alpha x + (1-\alpha)y| &\leq |\alpha x| + |(1-\alpha)y| \\ &= \alpha|x| + (1-\alpha)|y| \end{aligned}$$

REVIEW OF CONVEX FUNCTIONS

Example: Exponential

$$f(x) = e^x$$



$$\begin{aligned}
e^{\alpha x+(1-\alpha)y} &= e^y e^{\alpha(x-y)} = e^y \sum_{n=0}^{\infty} \frac{1}{n!}\alpha^n (x-y)^n \\
&\leq e^y \left(1 + \alpha \sum_{n=1}^{\infty} \frac{1}{n!}(x-y)^n\right) \qquad (\text{if } x > y) \\
&= e^y \left((1-\alpha) + \alpha e^{x-y}\right) \\
&= (1-\alpha)e^y + \alpha e^x
\end{aligned}$$

PROPERTIES OF CONVEX FUNCTIONS (CONTINUED)

- Non-negative combinations of convex functions are convex

$$h(x) = af(x) + bg(x)$$

- Affine scalings of convex functions are convex

$$h(x) = f(Ax + b)$$

- Compositions of convex functions are **NOT** generally convex
  - Neural nets are like this

$$h(x) = f(g(x))$$

CONVEX FUNCTIONS: ALTERNATIVE DEFINITIONS

- First-order condition

$$\langle x - y, \nabla f(x) - \nabla f(y) \rangle \geq 0$$

- Second-order condition

$$\nabla^2 f(x) \succeq 0$$

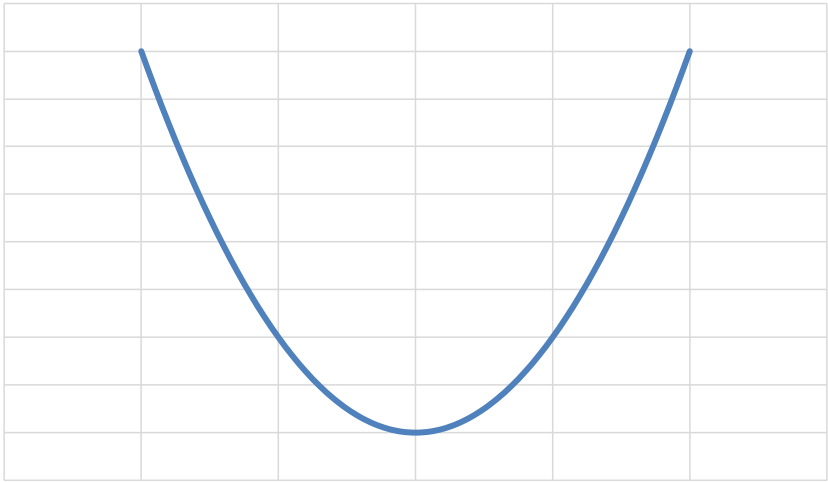
  - This means that the matrix of second derivatives is positive semidefinite

$$A \succeq 0 \Leftrightarrow \forall x,\ \langle x, Ax \rangle \geq 0$$

CONVEX FUNCTIONS: EXAMPLES

Example: Quadratic

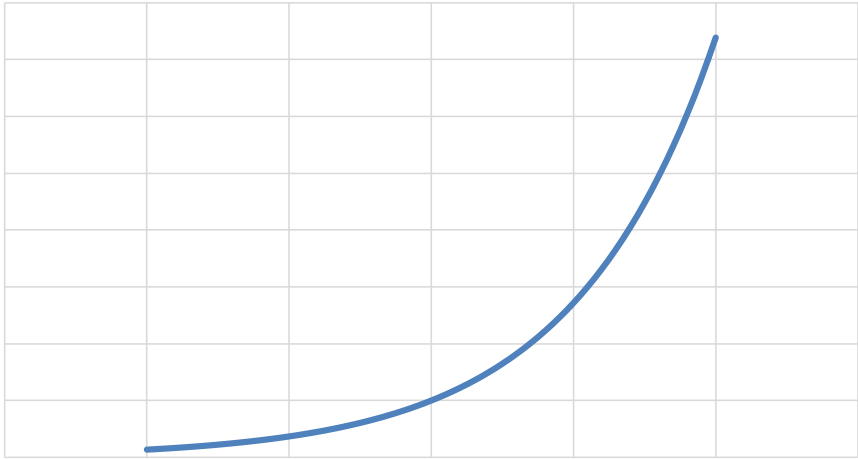
$$f(x) = x^2$$



$$f''(x) = 2 \geq 0$$

CONVEX FUNCTIONS: EXAMPLES

Example: Exponential

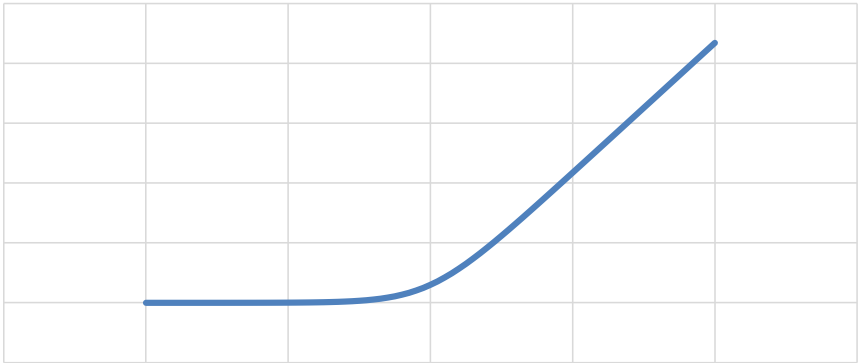


$$f(x) = e^x$$

$$f''(x) = e^x \geq 0$$

CONVEX FUNCTIONS: EXAMPLES

Example: Logistic Loss



$$f(x) = \log(1 + e^x)$$

$$f'(x) = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}}$$

$$f''(x) = -\frac{-e^{-x}}{(1 + e^{-x})^2} = \frac{1}{(1 + e^x)(1 + e^{-x})} \geq 0.$$

STRONGLY CONVEX FUNCTIONS

- Basically the easiest class of functions for optimization
  - First-order condition:

$$\langle x - y, \nabla f(x) - \nabla f(y) \rangle \geq \mu \|x - y\|^2$$

  - Second-order condition:

$$\nabla^2 f(x) \succeq \mu I$$

  - Equivalently:

$$h(x) = f(x) - \tfrac{\mu}{2}\|x\|^2 \text{ is convex}$$

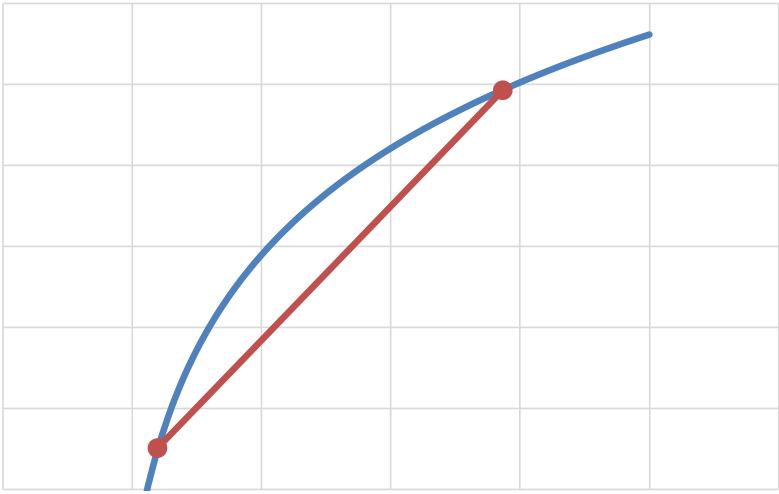
CONCAVE FUNCTIONS

- A function is concave if its negation is convex

$$f \text{ is convex} \Leftrightarrow h(x) = -f(x) \text{ is concave}$$

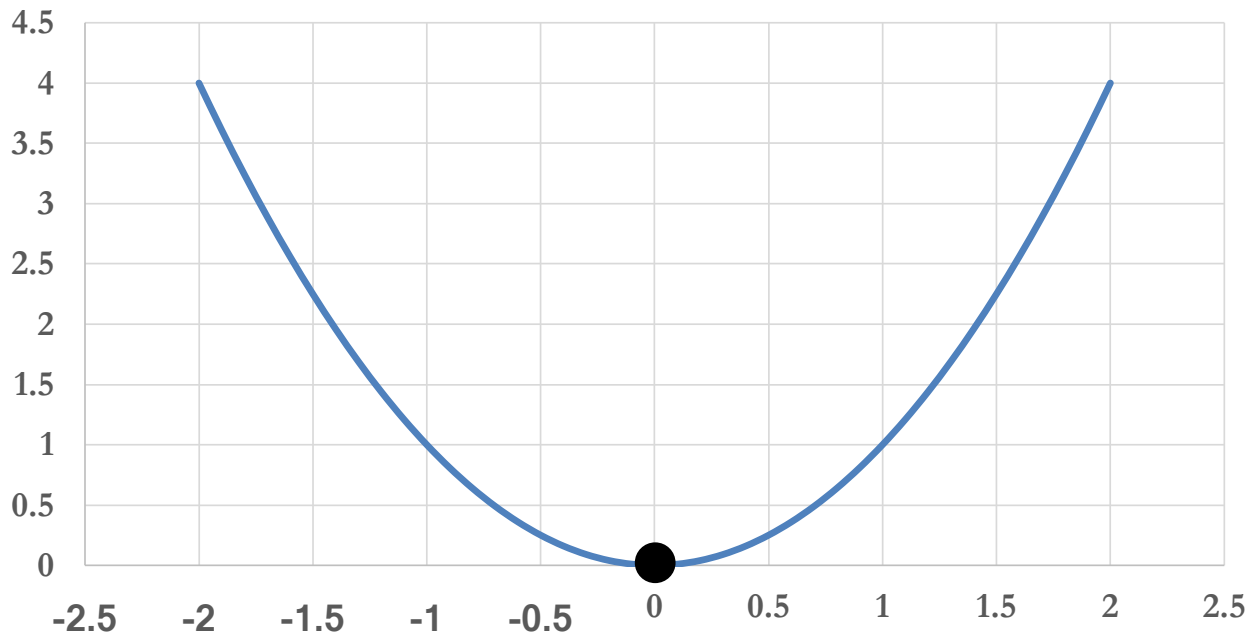
- Example: $f(x) = \log(x)$

$$f''(x) = -\frac{1}{x^2} \leq 0$$



WHY CARE ABOUT CONVEX FUNCTIONS?

- Goal is to minimize a convex function

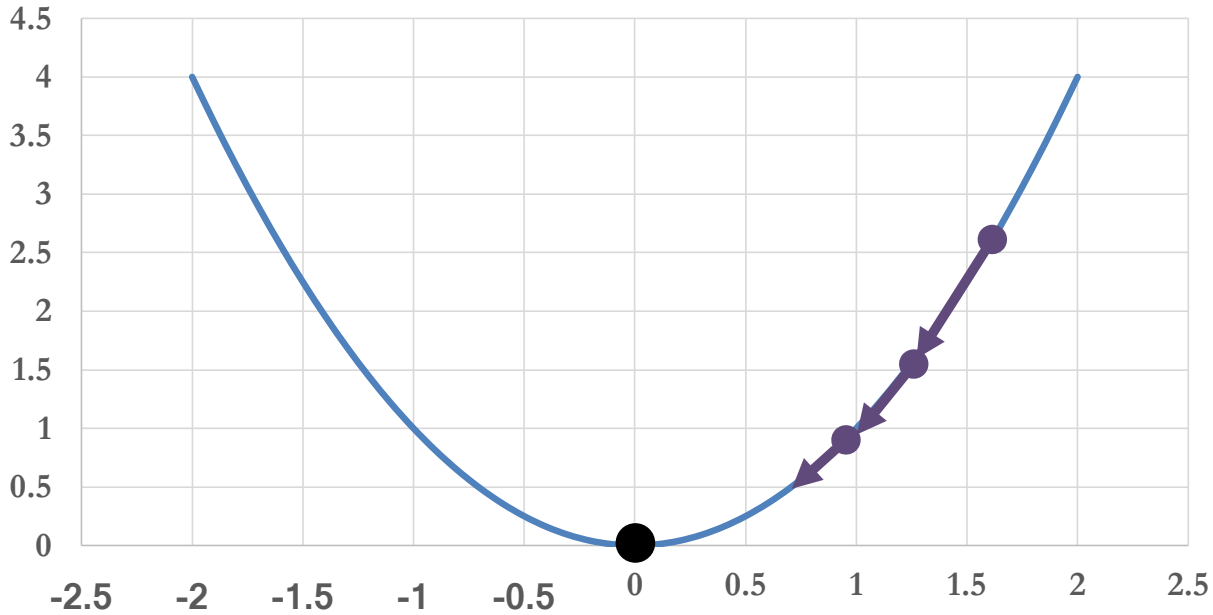


OUTLINE

- Statistical and computational tradeoffs
- Review of Convex Functions
- Gradient Descent (GD)
- Stochastic Gradient Descent (SGD + GSD)
- SDG with Mini-Batching
- Regularization

GRADIENT DESCENT

$$x \leftarrow x - \alpha \nabla f(x)$$



GRADIENT DESCENT CONVERGES

- Iterative definition of gradient descent

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

- Assumptions/terminology:

$$\text{Global optimum is } x^*$$

$$\text{Bounded second derivative } \mu I \preceq \nabla^2 f(x) \preceq \text{Ł} I$$

GRADIENT DESCENT CONVERGES

$$
\begin{aligned}
x_{t+1} - x^* &= x_t - x^* - \alpha(\nabla f(x_t) - \nabla f(x^*)) \\
&= x_t - x^* - \alpha \nabla^2 f(z_t)(x_t - x^*) \\
&= (I - \alpha \nabla^2 f(z_t))(x_t - x^*).
\end{aligned}
$$

- Taking the norm

$$
\begin{aligned}
\|x_{t+1} - x^*\| &\leq \left\| I - \alpha \nabla^2 f(z_t) \right\| \|x_t - x^*\| \\
&\leq \max\left(|1 - \alpha\mu|, |1 - \alpha L|\right) \|x_t - x^*\|
\end{aligned}
$$

GRADIENT DESCENT CONVERGES

- So if we set $\alpha = 2/(L+\mu)$ then

$$\|x_{t+1} - x^*\| \leq \frac{L-\mu}{L+\mu}\|x_t - x^*\|$$

- And recursively

$$\|x_T - x^*\| \leq \left(\frac{L-\mu}{L+\mu}\right)^T \|x_0 - x^*\|$$

  - Called **convergence at a linear rate** or sometimes (confusingly) exponential rate

THE PROBLEM WITH GRADIENT DESCENT

- Large-scale optimization

$$h(x) = \frac{1}{N} \sum_{i=1}^{N} f(x; y_i)$$

- Computing the gradient takes O(N) time

$$\nabla h(x) = \frac{1}{N} \sum_{i=1}^{N} \nabla f(x; y_i)$$

GRADIENT DESCENT WITH MORE DATA

- Suppose we add more examples to our training set
  - For simplicity, imagine we just add an extra copy of every training example

$$\nabla h(x) = \frac{1}{2N}\sum_{i=1}^{N}\nabla f(x; y_i) + \frac{1}{2N}\sum_{i=1}^{N}\nabla f(x; y_i)$$

- **Same objective function**
  - But gradients take **2x the time to compute** (unless we cheat)

- We want to **scale up to huge datasets**, so how can we do this?

OUTLINE

- Statistical and computational tradeoffs
- Review of Convex Functions
- Gradient Descent (GD)
- Stochastic Gradient Descent (SGD + GSD)
- SDG with Mini-Batching
- Regularization

STOCHASTIC GRADIENT DESCENT

- Idea: rather than using the full gradient, just use one training example
  - Super fast to compute

$$x_{t+1} = x_t - \alpha \nabla f(x_t; y_{\tilde{i}_t})$$

This is an example selected uniformly at random from the dataset.

- In expectation, it's just gradient descent:

$$\begin{aligned}\mathbf{E}\left[x_{t+1}\right] &= \mathbf{E}\left[x_t\right] - \alpha \mathbf{E}\left[\nabla f(x_t; y_{i_t})\right] \\ &= \mathbf{E}\left[x_t\right] - \alpha \frac{1}{N} \sum_{i=1}^{N} \nabla f(x_t; y_i)\end{aligned}$$

STOCHASTIC GRADIENT DESCENT CONVERGENCE

- Can SGD converge using just one example to estimate the gradient?

$$\begin{aligned} x_{t+1} - x^* &= x_t - x^* - \alpha\left(\nabla h(x_t) - \nabla h(x^*)\right) - \alpha\left(\nabla f(x_t; y_{i_t}) - \nabla h(x_t)\right) \\ &= \left(I - \alpha \nabla^2 h(z_t)\right)(x_t - x^*) - \underline{\alpha\left(\nabla f(x_t; y_{i_t}) - \nabla h(x_t)\right)} \end{aligned}$$

- How do we handle this extra noise term?

- **Answer: bound it using the variance!**
  - Variance of a constant plus a random variable is just the variance of that random variable, so we don't need to think about the rest of the expression.

STOCHASTIC GRADIENT DESCENT CONVERGENCE

$$
\begin{aligned}
&\mathbf{Var}\left(x_{t+1} - x^* | x_t\right) \\
&\quad = \mathbf{Var}\left(\left(I - \alpha \nabla^2 h(z_t)\right)(x_t - x^*) - \alpha\left(\nabla f(x_t; y_{i_t}) - \nabla h(x_t)\right)\middle| x_t\right) \\
&\quad = \mathbf{Var}\left(\alpha\left(\nabla f(x_t; y_{i_t}) - \nabla h(x_t)\right) | x_t\right) \\
&\quad = \alpha^2 \mathbf{Var}\left(\nabla f(x_t; y_{i_t}) - \nabla h(x_t) | x_t\right) \\
&\quad = \alpha^2 \mathbf{E}\left[\|\nabla f(x_t; y_{i_t}) - \nabla h(x_t)\|^2 \middle| x_t\right]
\end{aligned}
$$

STOCHASTIC GRADIENT DESCENT CONVERGENCE

$$\begin{aligned}
&\mathbf{E}\left[\|x_{t+1} - x^*\|^2 \middle| x_t\right] \\
&\qquad = \|\mathbf{E}\left[x_{t+1} - x^* | x_t\right]\|^2 + \mathbf{Var}\left(x_{t+1} - x^* | x_t\right) \\
&\qquad \leq (1 - \alpha\mu)^2 \|x_t - x^*\|^2 + \alpha^2 M \qquad (\text{for } \alpha \ll 1)
\end{aligned}$$

- So by the law of total expectation

$$\mathbf{E}\left[\|x_{t+1} - x^*\|^2\right] \leq (1 - \alpha\mu)^2 \mathbf{E}\left[\|x_t - x^*\|^2\right] + \alpha^2 M$$

STEP SIZES AND CONVERGENCE

- Stochastic gradient descent

$$x_{t+1} = x_t - \alpha \nabla f(x_t; y_{\tilde{i}_t})$$

- Much faster per iteration than gradient descent
  - Because we don't have to process the entire training set

- But converges to a noise ball

$$\lim_{T \to \infty} \mathbf{E}\left[\|x_T - x^*\|^2\right] \leq \frac{\alpha M}{2\mu - \alpha\mu^2}$$

CONTROLLING THE ACCURACY

- Want the noise ball to be as small as possible for accurate solutions

- Noise ball **proportional to the step size/learning rate**

$$\lim_{T\to\infty} \mathbf{E}\left[\|x_T - x^*\|^2\right] \leq \frac{\alpha M}{2\mu - \alpha\mu^2} = O(\alpha)$$

- So **should we make the step size as small as possible?**

EFFECT OF STEP SIZE ON CONVERGENCE

- Let's go back to the convergence rate proof for SGD
  - From the previous lecture, we have

$$\mathbf{E}\left[\|x_{t+1} - x^*\|^2 \middle| x_t\right] \leq (1 - \alpha\mu)^2 \|x_t - x^*\|^2 + \alpha^2 M.$$

- If we're far from the noise ball i.e. $\|x_t - x^*\|^2 \geq \frac{2\alpha M}{\mu}$

$$\mathbf{E}\left[\|x_{t+1} - x^*\|^2 \middle| x_t\right] \leq (1 - \alpha\mu)^2 \|x_t - x^*\|^2 + \frac{\alpha\mu}{2} \|x_t - x^*\|^2 .$$

EFFECT OF STEP SIZE ON CONVERGENCE

$$\begin{aligned}\mathbf{E}\left[\|x_{t+1}-x^*\|^2 \middle| x_t\right] &\leq (1-\alpha\mu)^2 \|x_t - x^*\|^2 + \frac{\alpha\mu}{2}\|x_t - x^*\|^2 \\ &\leq \left(1-\frac{\alpha\mu}{2}\right)\|x_t - x^*\|^2 \qquad (\text{if } \alpha\mu < 1) \\ &\leq \exp\left(-\frac{\alpha\mu}{2}\right)\|x_t - x^*\|^2 .\end{aligned}$$

- So to contract by a factor of **C**, we need to run **T** steps, where

$$1 = \exp\left(-\frac{\alpha\mu T}{2}\right) C \Leftrightarrow T = \frac{2}{\alpha\mu}\log C$$

THE FULL EFFECT OF STEP SIZE

- Noise ball **proportional to the step size**

$$\lim_{T\to\infty} \mathbf{E}\left[\|x_T - x^*\|^2\right] \leq \frac{\alpha M}{2\mu - \alpha\mu^2} = O(\alpha)$$

- Convergence time **inversely proportional to the step size**

$$T = \frac{2}{\alpha\mu}\log C$$

- So **there's a trade-off!**

CAN WE GET THE BEST OF BOTH WORLDS?

- When do we want the step size to be large?
  - **At the beginning of execution!** Near the end? Both?
- When do we want the step size to be small?
  - At the beginning of execution? **Near the end!** Both?
- What about using a **decreasing step size scheme?**

STOCHASTIC GRADIENT DESCENT IS SUPER POPULAR

But how SGD is implemented in practice is not exactly what I've just shown you…

…and we'll see how it's different in the upcoming lectures.

OUTLINE

- Statistical and computational tradeoffs
- Review of Convex Functions
- Gradient Descent (GD)
- Stochastic Gradient Descent (SGD + GSD)
- SDG with Mini-Batching
- Regularization

GRADIENT DESCENT VERSUS SGD

- Gradient descent: **all examples at once**

$$x_{t+1} = x_t - \alpha_t \frac{1}{N} \sum_{i=1}^{N} \nabla f(x_t; y_i)$$

- Stochastic gradient descent: **one example at a time**

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t; y_{i_t})$$

- Is it really **all or nothing**? Can we do something intermediate?

MINI-BATCH STOCHASTIC GRADIENT DESCENT

- An intermediate approach

$$x_{t+1} = x_t - \alpha_t \frac{1}{|B_t|} \sum_{i \in B_t} \nabla f(x_t; y_i)$$

where B_t is sampled uniformly from the set of all subsets of $\{1, \ldots, N\}$ of size b.

  - The b parameter is the **batch size**
  - Typically choose b << N.

- Also called **mini-batch gradient descent**

ADVANTAGES OF MINI-BATCH

- Takes **less time to compute each update** than gradient descent
  - Only needs to sum up b gradients, rather than N

$$x_{t+1} = x_t - \alpha_t \frac{1}{|B_t|} \sum_{i \in B_t} \nabla f(x_t; y_i)$$

- But takes **more time for each update** than SGD
  - So what's the benefit?

- It's more like gradient descent, so **maybe it converges faster** than SGD?

MINI-BATCH SGD CONVERGES

- Start by breaking up the update rule into expected update and noise

$$x_{t+1} - x^* = x_t - x^* - \alpha_t \left(\nabla h(x_t) - \nabla h(x^*)\right) - \alpha_t \frac{1}{|B_t|} \sum_{i \in B_t} \left(\nabla f(x_t; y_i) - \nabla h(x_t)\right)$$

- Variance analysis

$$\mathbf{Var}\left(x_{t+1} - x^*\right) = \mathbf{Var}\left(\alpha_t \frac{1}{|B_t|} \sum_{i \in B_t} \left(\nabla f(x_t; y_i) - \nabla h(x_t)\right)\right)$$

MINI-BATCH SGD CONVERGES

$$\text{Let } \Delta_i = \nabla f(x_t; y_i) - \nabla h(x_t), \text{ and } \beta_i = \begin{cases} 1 & i \in B_t \\ 0 & i \notin B_t \end{cases}$$

$$\begin{aligned}
\mathbf{Var}\left(x_{t+1} - x^*\right) &= \frac{\alpha_t^2}{|B_t|^2} \mathbf{Var}\left(\sum_{i \in B_t} \left(\nabla f(x_t; y_i) - \nabla h(x_t)\right)\right) \\
&= \frac{\alpha_t^2}{|B_t|^2} \mathbf{Var}\left(\sum_{i=1}^{N} \beta_i \Delta_i\right) \\
&= \frac{\alpha_t^2}{|B_t|^2} \sum_{i=1}^{N} \sum_{j=1}^{N} \beta_i \beta_j \Delta_i \Delta_j
\end{aligned}$$

MINI-BATCH SGD CONVERGES

- Because we sampled B uniformly at random, for **i ≠ j**

$$\mathbf{E}\left[\beta_i \beta_j\right] = \mathbf{P}\left(i \in B \wedge j \in B\right) = \mathbf{P}\left(i \in B\right)\mathbf{P}\left(j \in B | i \in B\right) = \frac{b}{N} \cdot \frac{b-1}{N-1}$$

$$\mathbf{E}\left[\beta_i^2\right] = \mathbf{P}\left(i \in B\right) = \frac{b}{N}$$

- So we can write the variance as

$$\mathbf{Var}\left(x_{t+1} - x^*\right) = \frac{\alpha_t^2}{|B_t|^2}\left(\sum_{i \neq j} \frac{b(b-1)}{N(N-1)} \Delta_i \Delta_j + \sum_{i=1}^{N} \frac{b}{N} \Delta_i^2\right)$$

MINI-BATCH SGD CONVERGES

$$
\begin{aligned}
\mathbf{Var}\left(x_{t+1} - x^*\right) &= \frac{\alpha_t^2}{b^2}\left(\sum_{i \neq j} \frac{b(b-1)}{N(N-1)} \Delta_i \Delta_j + \sum_{i=1}^{N} \frac{b}{N} \Delta_i^2\right) \\
&= \frac{\alpha_t^2}{bN}\left(\frac{b-1}{N-1} \sum_{i=1}^{N} \sum_{j=1}^{N} \Delta_i \Delta_j + \sum_{i=1}^{N} \left(1 - \frac{b-1}{N-1}\right) \Delta_i^2\right) \\
&= \frac{\alpha_t^2}{bN}\left(\frac{b-1}{N-1} \left(\sum_{i=1}^{N} \Delta_i\right)^2 + \frac{N-b}{N-1} \sum_{i=1}^{N} \Delta_i^2\right) \\
&= \frac{\alpha_t^2 (N-b)}{bN(N-1)} \sum_{i=1}^{N} \Delta_i^2
\end{aligned}
$$

MINI-BATCH SGD CONVERGES

$$
\begin{aligned}
\mathbf{Var}\left(x_{t+1} - x^*\right) &= \frac{\alpha_t^2 (N-b)}{b(N-1)} \cdot \frac{1}{N} \sum_{i=1}^{N} \Delta_i^2 \\
&= \frac{\alpha_t^2 (N-b)}{b(N-1)} \mathbf{E}\left[\|\nabla f(x_t; y_{i_t}) - \nabla h(x_t)\|^2 \middle| x_t \right] \\
&\leq \frac{\alpha_t^2 (N-b)}{b(N-1)} M \\
&\leq \alpha_t^2 \frac{M}{b}
\end{aligned}
$$

- Compared with SGD, **variance decreased by a factor of b**

MINI-BATCH SGD CONVERGES

- Recall that SGD converged to a noise ball of size

$$\lim_{T\to\infty} \mathbf{E}\left[\|x_T - x^*\|^2\right] \leq \frac{\alpha M}{2\mu - \alpha\mu^2}$$

- Since mini-batching decreases variance by a factor of **b**, it will have

$$\lim_{T\to\infty} \mathbf{E}\left[\|x_T - x^*\|^2\right] \leq \frac{\alpha M}{(2\mu - \alpha\mu^2)b}$$

  - **Noise ball smaller** by the same factor!

ADVANTAGES OF MINI-BATCH

- Takes **less time to compute each update** than gradient descent
  - Only needs to sum up b gradients, rather than N

$$x_{t+1} = x_t - \alpha_t \frac{1}{|B_t|} \sum_{i \in B_t} \nabla f(x_t; y_i)$$

- Converges to a **smaller noise ball** than stochastic gradient descent

$$\lim_{T \to \infty} \mathbf{E}\left[\|x_T - x^*\|^2\right] \leq \frac{\alpha M}{(2\mu - \alpha\mu^2)b}$$

HOW TO CHOOSE THE BATCH SIZE?

- **Mini-batching is not a free win**
  - Naively, compared with SGD, it takes **b** times as much effort to get a **b**-times-as-accurate answer
  - But we could have gotten a **b**-times-as-accurate answer by just running SGD for **b** times as many steps with a step size of $\alpha/\mathbf{b}$.

- But it still makes sense to run it for **systems** and **statistical** reasons
  - Mini-batching exposes more parallelism
  - Mini-batching lets us estimate statistics about the full gradient more accurately

- Another use case for **metaparameter optimization**

MINI-BATCH SGD IS VERY WIDELY USED

- Including in basically all neural network training
- **b = 32** is a typical default value for batch size
  - From "Practical Recommendations for Gradient-Based Training of Deep Architectures," Bengio 2012.

OUTLINE

- Statistical and computational tradeoffs
- Review of Convex Functions
- Gradient Descent (GD)
- Stochastic Gradient Descent (SGD + GSD)
- SDG with Mini-Batching
- Regularization

MINIMIZING TRAINING LOSS IS NOT OUR REAL GOAL

- Training loss looks like

$$h(x) = \frac{1}{N} \sum_{i=1}^{N} f(x; y_i)$$

- What we actually want to minimize is **expected loss on new examples**
  - Drawn from some real-world distribution ϕ

$$\bar{h}(x) = \mathbf{E}_{y \sim \phi} \left[f(x; y) \right]$$

  - Typically, assume the training examples were drawn from this distribution

OVERFITTING

- Minimizing the training loss **doesn't generally minimize the expected loss** on new examples
  - They are two different objective functions after all
- Difference between the empirical loss on the training set and the expected loss on new examples is called the **generalization error**
- Even a model that has high accuracy on the training set can have terrible performance on new examples
  - Phenomenon is called **overfitting**

REGULARIZATION

- Add an extra **regularization term** to the objective function

- Most popular type: **L2 regularization**

$$h(x) = \frac{1}{N}\sum_{i=1}^{N} f(x; y_i) + \sigma^2 \|x\|_2^2 = \frac{1}{N}\sum_{i=1}^{N} f(x; y_i) + \sigma^2 \sum_{k=1}^{d} x_i^2$$

- Also popular: **L1 regularization**

$$h(x) = \frac{1}{N}\sum_{i=1}^{N} f(x; y_i) + \gamma \|x\|_1 = \frac{1}{N}\sum_{i=1}^{N} f(x; y_i) + \gamma \sum_{k=1}^{d} |x_i|$$

BENEFITS OF REGULARIZATION

- **Cheap to compute**
  - For SGD and L2 regularization, there's just an extra scaling

$$x_{t+1} = (1 - 2\alpha_t \sigma^2) x_t - \alpha_t \nabla f(x_t; y_{i_t})$$

- **Makes the objective strongly convex**
  - This makes it easier to get and prove bounds on convergence

- **Helps with overfitting**

MOTIVATION FROM BAYES

- One way to think about regularization is as a **Bayesian prior**

- MLE interpretation of learning problem: $\mathbf{P}(y_i|x) = \frac{1}{Z}\exp(-f(x;y_i))$

$$\mathbf{P}(x|y) = \frac{\mathbf{P}(y|x)\,\mathbf{P}(x)}{\mathbf{P}(y)} = \frac{\mathbf{P}(y|x)\,\mathbf{P}(x)}{\mathbf{P}(y)}\prod_{i=1}^{N}\frac{1}{Z}\exp(-f(x;y_i))$$

  - Taking the logarithm:

$$\log\mathbf{P}(x|y) = -\log(Z) - \sum_{i=1}^{N} f(x;y_i) + \cancel{\log\mathbf{P}(x)} - \log\mathbf{P}(y)$$

MOTIVATION FROM BAYES

- So the MLE problem becomes

$$\max_x \log \mathbf{P}(x|y) = -\sum_{i=1}^{N} f(x; y_i) + \log \mathbf{P}(x) + (\text{constants})$$

- Now we need to pick a **prior probability distribution** for $\mathbf{x}$
  - Say we choose a Gaussian prior

$$\max_x \log \mathbf{P}(x|y) = -\sum_{i=1}^{N} f(x; y_i) + \log\left(\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{\sigma^2 \|x\|^2}{2}\right)\right) + (C)$$

$$= -\sum_{i=1}^{N} f(x; y_i) - \frac{\sigma^2}{2}\|x\|^2 + (C)$$

There's our regularization term!

MOTIVATION

- By setting a prior, we **limit the values we think the model can have**
- This prevents the model from doing bad things to try to fit the data
  - Like the polynomial-fitting example from the demo

[1] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Man, Rajat Monga, Sherry Moore, Derek Murray, Jon Shlens, Benoit Steiner, Ilya Sutskever, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Oriol Vinyals, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. 2015.

[2] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in Optimizing Recurrent Networks. 2012.

[3] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[4] C. Darken, J. Chang, and J. Moody. Learning rate schedules for faster stochastic gradient search. *Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop*, (September):1–11, 1992.

[5] Yann N. Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *arXiv*, pages 1–14, 2014.

[6] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. Large Scale Distributed Deep Networks. *NIPS 2012: Neural Information Processing Systems*, pages 1–11, 2012.

[7] Timothy Dozat. Incorporating Nesterov Momentum into Adam. *ICLR Workshop*, (1):2013–2016, 2016.

[8] John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[9] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv preprint arXiv:1502.03167v3*, 2015.

[10] Diederik P. Kingma and Jimmy Lei Ba. Adam: a Method for Stochastic Optimization. *International Conference on Learning Representations*, pages 1–13, 2015.

[11] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Müller. Efficient BackProp. *Neural Networks: Tricks of the Trade*, 1524:9–50, 1998.

[12] H. Brendan Mcmahan and Matthew Streeter. Delay-Tolerant Algorithms for Asynchronous Distributed Online Learning. *Advances in Neural Information Processing Systems (Proceedings of NIPS)*, pages 1–9, 2014.

[13] Arvind Neelakantan, Luke Vilnis, Quoc V. Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding Gradient Noise Improves Learning for Very Deep Networks. pages 1–11, 2015.

[14] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o(1/k2). *Doklady ANSSSR (translated as Soviet.Math.Docl.)*, 269:543–547.

[15] Feng Niu, Benjamin Recht, R Christopher, and Stephen J Wright. Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. pages 1–22, 2011.

[16] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, 2014.

[17] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks : the official journal of the International Neural Network Society*, 12(1):145–151, 1999.

[18] Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

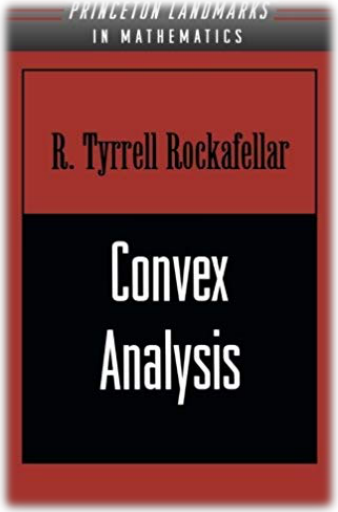
[19] Ilya Sutskever. Training Recurrent neural Networks. *PhD thesis*, page 101, 2013.

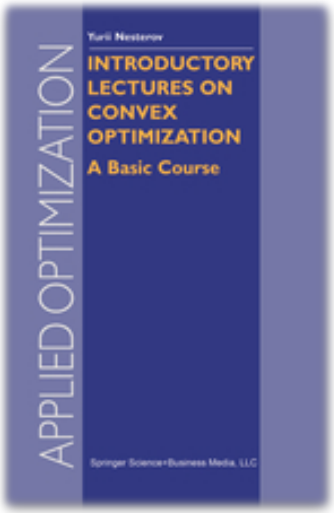
[20] Richard S. Sutton. Two problems with backpropagation and other steepest-descent learning procedures for networks, 1986.

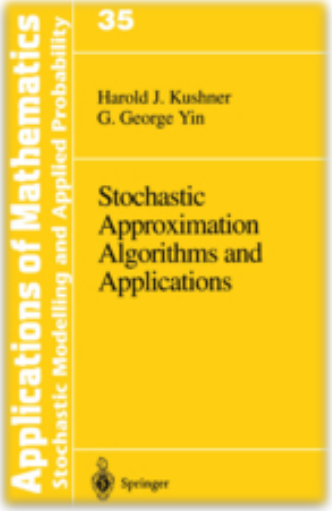
[21] Wojciech Zaremba and Ilya Sutskever. Learning to Execute. pages 1–25, 2014.

[22] Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv preprint arXiv:1212.5701*, 2012.

[23] Sixin Zhang, Anna Choromanska, and Yann LeCun. Deep learning with Elastic Averaging SGD. *Neural Information Processing Systems Conference (NIPS 2015)*, pages 1–24, 2015.







Many of the practical claims and tricks about SGD:

- Overfitting in Neural Nets : Backpropagation, Conjugate Gradient, and Early Stopping by Caruana, R. (online)
- Solving large scale linear prediction problems using stochastic gradient descent algorithms by T. Zhang (online).
- Practical Recommendations for Gradient-Based Training of Deep Architectures by Y. Bengio (online).
- Optimization Methods for Large-Scale Machine Learning by L. Bottou (online).