

Online Learning: Estimation and Inference

Linglong Kong

Department of Mathematical and Statistical Sciences
University of Alberta

Summer, 2026

Outline

- 1 Online Machine Learning
- 2 Basic Streaming Algorithms
- 3 Online Learning in Statistics
- 4 Recursive Algorithms
- 5 Online Updating Estimations
- 6 Renewable Estimation and Incremental Inference
- 7 Parallel-and-Stream Accelerator
- 8 Online Nonparametric Estimation
- 9 Advanced Gradient Descent
- 10 Deep Bootstrap Framework

What is Online Machine Learning?

Compared to "traditional" ML solutions, **online learning** is a fundamentally different approach, one that embraces the fact that **learning environments** can (and do) **change from second to second**.

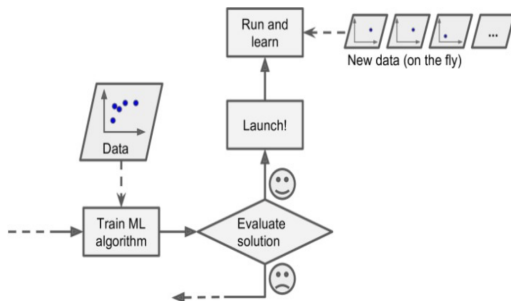


Figure: Online learning: Machine Learning System

What is Streaming Data?

- Streaming data is **data generated continuously** by thousands of data sources, which typically send in the data records **simultaneously, and in small sizes**.



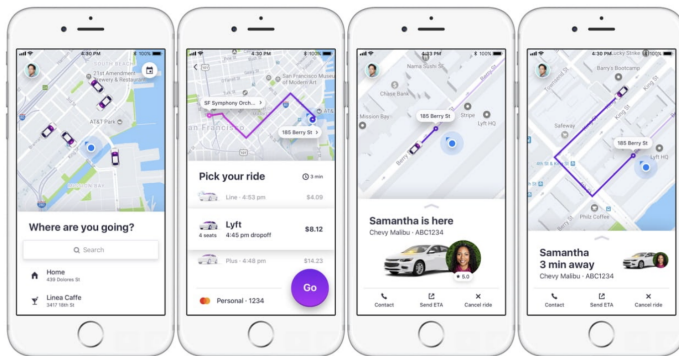
Benefits of Streaming Data

In a word: **real-time insight!**

More details:

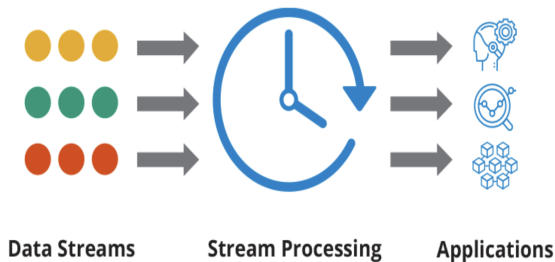
- Good for business: Real-time intelligence makes companies more responsive to market trends, customer needs, and business opportunities.
- Good for costumers: e.g., Netflix recommendations, a seamless shopping experience.

A Real Application: Lyft



Lyft is a ride-sharing app that requires up-to-the-second data to accurately match riders with drivers.

How to handle Streaming Data?



Basic idea: Streaming data needs to be **processed sequentially and incrementally** over sliding time windows.

From Offline Learning to Online Learning

Offline/batch learning:

- An entire dataset is available
- The examples can be processed several times
- Weak constrain on the computing time
- The distribution of data does not change

Online learning:

- Can be interrupted before its end
- Returns a valid result at any time
- Is expected to perform better as data come in continuously
- Relevant for time-critical application

From Offline Learning to Online Learning

In summary:

- Offline learning is an approach that ingests all the data at one time to train a model.
- In online learning, the training happens in **an incremental manner by continuously feeding data** as it arrives or in a small group.

Example

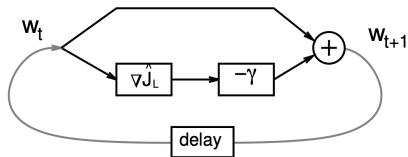


Figure: Batch Gradient Descent

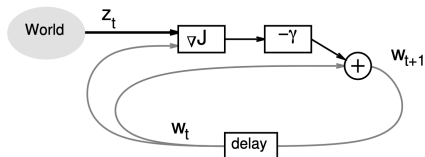


Figure: Stochastic Gradient Descent

Online Learning Versus Offline Learning

- Advantages of online learning:
 - Efficient
 - Adaptable
- When to consider online learning:
 - Data doesn't fit into memory
 - The distribution of your data may drift over time
 - Data is a function of time (e.g. stock prices)

Outline

- 1 Online Machine Learning
- 2 Basic Streaming Algorithms**
- 3 Online Learning in Statistics
- 4 Recursive Algorithms
- 5 Online Updating Estimations
- 6 Renewable Estimation and Incremental Inference
- 7 Parallel-and-Stream Accelerator
- 8 Online Nonparametric Estimation
- 9 Advanced Gradient Descent
- 10 Deep Bootstrap Framework

Requirements of Streaming Algorithms

- **Bounded storage:** only store a very limited amount of data to summarize the data stream
- **Single pass:** the incoming data points cannot be permanently stored
- **Real-time:** process data points on average at least as fast as the data is arriving
- **Concept drift:** can deal with a data generating process which evolves over time (e.g., distributions change or new structure in the data appears)

Finding Minimum

- Imagine that there is an extremely large list of numbers, too large for your RAM. You want to find out the minimum of this list.
- use “min(mylist)” ? No way...
- Alternatively, update the minimum:

$$\min\{a_1, a_2, \dots, a_n\} = \min\{\min\{a_1, \dots, a_{n-1}\}, a_n\}$$

Finding Mean

- $\text{mean} = \frac{a_1 + \dots + a_n}{n}$
- How to compute the mean of $n + 1$ elements when we already have the mean of the first n elements?
- Find

$$\text{updated mean} = \frac{a_1 + \dots + a_n + a_{n+1}}{n + 1} = \frac{\frac{a_1 + \dots + a_n}{n} \cdot n + a_{n+1}}{n + 1}.$$

In other words,

$$\text{updated mean} = \frac{\text{old mean} \cdot n + a_{n+1}}{n + 1}.$$

Reservoir Sampling: Background

- You have a large dataset and you want to **uniformly sample an object**.
- If you know the size n of the data set, you can uniformly draw a random number k between 1 and n , scan the data set and take the k -th element.
- Now imagine that you have a data stream...

Reservoir Sampling: Idea



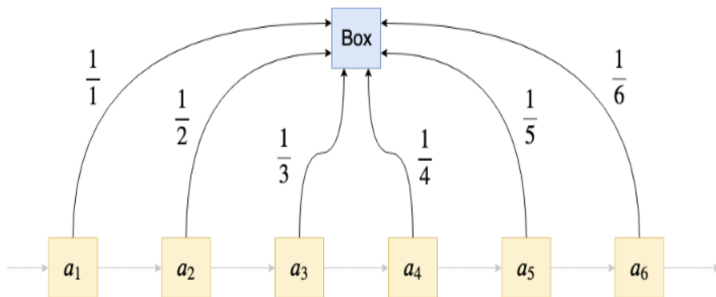
Figure: A prototypical reservoir

- You have a **single box (the reservoir)** for elements. When scanning the data stream, **replace the content of the box with the current element with a certain probability**.
- After passing over n elements **the goal** is to have each element in the box with probability $1/n$.

Reservoir Sampling: Steps

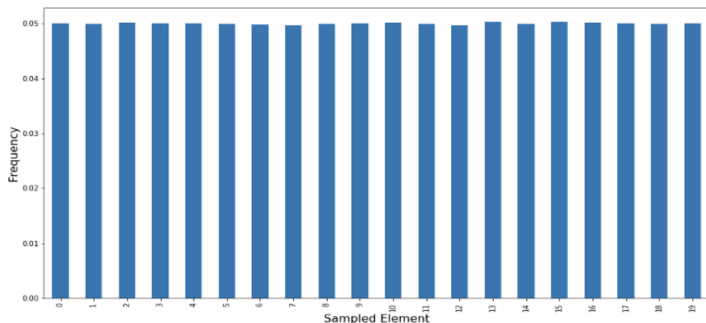
- Start with some constant probability p . Then the probability of the first element being in the box after $n - 1$ more elements is only $p(1 - p)^{(n-1)}$.
- **One idea** to fix this: We have to decrease the probability of a swap the longer we scan the sequence.
- What is a simple decay rate? How about $1/n$? Verify it on the next page.

Reservoir Sampling: Steps



Reservoir Sampling: Result

A quick check: repeatedly sample from a dataset of size 20. We expect to draw each element in about 5% of all cases.



[Source: <https://towardsdatascience.com/introduction-to-streaming-algorithms-b71808de6d29>]

Online Quantile Estimation: Randomized Algorithm

- **Target:** estimate the quantiles of a stream by the quantiles of a random input sample.
- **Key idea:** maintain a random sample of appropriate size, and when asked for a quantile of the input set, simply report the corresponding quantile of the random sample.
- If the size of the input stream, N is known, then just choose each element independently with probability k/N to include in the sample.
- If the size of the full data stream is unknown, then the reservoir sampling algorithm can be used instead.

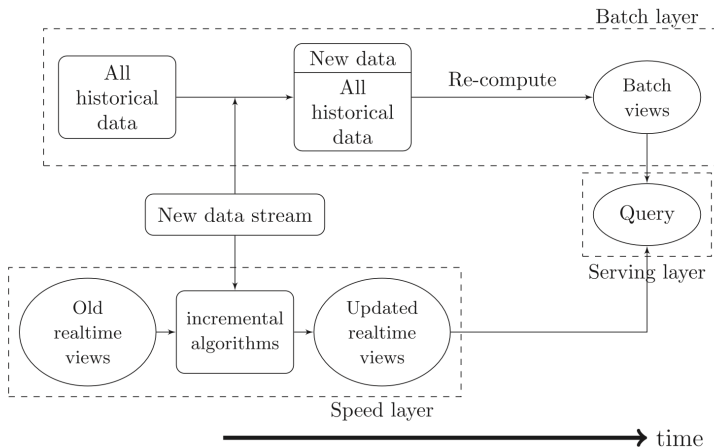
Outline

- 1 Online Machine Learning
- 2 Basic Streaming Algorithms
- 3 Online Learning in Statistics**
- 4 Recursive Algorithms
- 5 Online Updating Estimations
- 6 Renewable Estimation and Incremental Inference
- 7 Parallel-and-Stream Accelerator
- 8 Online Nonparametric Estimation
- 9 Advanced Gradient Descent
- 10 Deep Bootstrap Framework

Online Learning in Statistics

- Statistical analysis of streaming data sets has recently drawn considerable attention in the emerging field of “big data” analytics due to the availability of modern powerful computing platforms.
- The key methodology that is relevant to such data analysis pertains to algorithms that enable us to **update certain statistics of interest sequentially**.
- Online estimations target problems where **data arrive in streams or large chunks**, and address statistical problems in an updating framework **without storage requirements for previous data**.

Online Learning in Statistics



Online Learning in Statistics

Online data analytics:

- Recursive algorithms: SGD, (averaged) implicit SGD, online Newton step, online quasi-Newton methods, etc.
- Online updating estimations: online LSE, online estimating equations.
- Renewable estimation

Outline

- 1 Online Machine Learning
- 2 Basic Streaming Algorithms
- 3 Online Learning in Statistics
- 4 Recursive Algorithms**
- 5 Online Updating Estimations
- 6 Renewable Estimation and Incremental Inference
- 7 Parallel-and-Stream Accelerator
- 8 Online Nonparametric Estimation
- 9 Advanced Gradient Descent
- 10 Deep Bootstrap Framework

Recursive Algorithms

- SGD (Robbins and Monro, 1951):

$$\beta_i^{\text{sgd}} = \beta_{i-1}^{\text{sgd}} + \gamma_i \nabla_{\beta} \log \left\{ f \left(y_i; \mathbf{x}_i, \beta_{i-1}^{\text{sgd}} \right) \right\},$$

where $\gamma_i > 0$ is a pre-specified learning rate sequence.

- Implicit SGD:

$$\beta_i^{\text{im}} = \beta_{i-1}^{\text{im}} + \gamma_i \nabla_{\beta} \log \left\{ f \left(y_i; \mathbf{x}_i, \beta_i^{\text{im}} \right) \right\}.$$

- Toulis *et al.* (2014) concluded that implicit SGD appeared more robust to learning rate misspecification.

Recursive Algorithms

- Toulis *et al.* (2014) proposed an averaged implicit SGD (AISGD) algorithm,

$$\beta_i^{\text{aim}} = \frac{1}{i} \sum_{k=1}^i \beta_k^{\text{im}},$$

which was shown to be more statistically efficient.

- Asymptotic properties of both explicit and implicit SGDs have been well studied (Polyak and Juditsky, 1992).
- Traditional interval estimation procedures such as the plug-in procedure and the bootstrap are numerically difficult or infeasible in the presence of large-scale data or streaming data.

Recursive Algorithms

- Fang (2019) extended AISGD by adding a non-negative **random weight** $W_i^{(s)}$, to the gradient, resulting in the following implicit SGD procedure:

$$\beta_i^{(s)\text{im}} = \beta_{i-1}^{(s)\text{im}} + \gamma_i W_i^{(s)} \nabla_{\beta} \log \left\{ f \left(y_i; \mathbf{x}_i, \beta_i^{(s)\text{im}} \right) \right\},$$

$$\beta_i^{(s)\text{aim}} = \frac{1}{i} \sum_{k=1}^i \beta_k^{(s)\text{im}}.$$

- $\mathbb{E}(W_i) = \text{Var}(W_i) = 1$
- Set $W_i^{(s)} \equiv 1 \rightarrow$ AISGD estimate.

Recursive Algorithms

- Using samples drawn from, say, $W_i^{(s)} \sim \text{exponential}(1)$, $s = 1, \dots, S$, we obtain S copies of $\beta_i^{(s)\text{aim}}$.
- Further, using these replicates, we can assess the variability of $\beta_i^{(s)\text{aim}}$ and calculate the empirical standard error of the AISGD estimator for statistical inference.
- Called the **online bootstrap resampling procedure**, used to approximate the distribution of an AISGD estimator.

References I



Luo, L. and Song, P.X.

Renewable estimation and incremental inference in generalized linear models with streaming data sets

J. R. Statist. Soc. B, 82: 69-97, 2020.



Robbins, H. and Monro, S.

A stochastic approximation method

Ann. Math. Statist., 22(3): 400-407, 1951.



Toulis, P., Airoldi, E., and Rennie, J.

Statistical analysis of stochastic gradient methods for generalized linear models

International Conference on Machine Learning, 667-675, 2014.



Fang, Y.

Scalable statistical inference for averaged implicit stochastic gradient descent

Scandinavian Journal of Statistics, 46(4): 987-1002, 2019.



Polyak, B. and Juditsky, A.

Acceleration of stochastic approximation by averaging

SIAM journal on control and optimization, 30(4): 838-855, 1992.

Outline

- 1 Online Machine Learning
- 2 Basic Streaming Algorithms
- 3 Online Learning in Statistics
- 4 Recursive Algorithms
- 5 Online Updating Estimations**
- 6 Renewable Estimation and Incremental Inference
- 7 Parallel-and-Stream Accelerator
- 8 Online Nonparametric Estimation
- 9 Advanced Gradient Descent
- 10 Deep Bootstrap Framework

Online Mean and Variance Estimations

- Online mean estimation: $\bar{x}_n = \frac{(n-1)\bar{x}_{n-1} + x_n}{n} = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n}$
- Online variance estimation:

$$s_n^2 = \frac{(n-2)}{(n-1)} s_{n-1}^2 + \frac{(x_n - \bar{x}_{n-1})^2}{n}, \quad n > 1$$

- Keeping **summary statistics**, such as $\bar{x}_n$ and s_n^2 , rather than the whole historical data.

Online Least Squares Estimation

Setup:

- data arrive in blocks $k = 1, 2, \dots$
- in block k , n_k independent observations $\{(y_{ki}, \mathbf{x}_{ki}) : i = 1, \dots, n_k\}$
- y_{ki} is the response variable while $\mathbf{x}_{ki}$ is a $p \times 1$ covariate vector
- the regression model:

$$y_{ki} = \mathbf{x}_{ki}^\top \boldsymbol{\beta} + \epsilon_{ki}$$

Online Least Squares Estimation

- Let $\mathbf{y}_k = (y_{k1}, \dots, y_{kn_k})^\top$ and $\mathbf{X}_k = (\mathbf{x}_{k1}, \dots, \mathbf{x}_{kn_k})^\top$.
- The OLS estimator of $\boldsymbol{\beta}$ from block k is $\hat{\boldsymbol{\beta}}_k = (\mathbf{X}_k^\top \mathbf{X}_k)^{-1} \mathbf{X}_k^\top \mathbf{y}_k$.
- An appropriate estimator of σ_ϵ^2 from block k is

$$\hat{\sigma}_k^2 = (n_k - p)^{-1} \mathbf{y}_k^\top [I - \mathbf{X}_k (\mathbf{X}_k^\top \mathbf{X}_k)^{-1} \mathbf{X}_k^\top] \mathbf{y}_k.$$

- At the k th block, the cumulative coefficient estimator of $\boldsymbol{\beta}$ (based on all data from blocks 1 through k) is

$$\tilde{\boldsymbol{\beta}}_{1:k} = (\mathbf{X}_k^\top \mathbf{X}_k + \mathbf{V}_{k-1})^{-1} (\mathbf{X}_k^\top \mathbf{X}_k \hat{\boldsymbol{\beta}}_k + \mathbf{V}_{k-1} \tilde{\boldsymbol{\beta}}_{1:(k-1)}),$$

where $\tilde{\boldsymbol{\beta}}_0 = \mathbf{0}$, $\mathbf{V}_0 = \mathbf{0}_p$ is a $p \times p$ matrix of zeros, and $\mathbf{V}_k = \sum_{l=1}^k \mathbf{X}_l^\top \mathbf{X}_l$.

Online Least Squares Estimation

- The online-updated estimator of the residual sum of squares (SSE) based on cumulative data is

$$\begin{aligned} \text{SSE}_{1:k} = \text{SSE}_{1:(k-1)} + \text{SSE}_k + \tilde{\beta}_{1:(k-1)}^\top \mathbf{V}_{k-1} \tilde{\beta}_{1:(k-1)} \\ + \hat{\beta}_k^\top \mathbf{X}_k^\top \mathbf{X}_k \hat{\beta}_k - \tilde{\beta}_{1:k}^\top \mathbf{V}_k \tilde{\beta}_{1:k}. \end{aligned}$$

- Therefore, $\tilde{\sigma}_{1:k}^2 = \text{SSE}_{1:k} / (N_k - p)$, where $N_k = \sum_{l=1}^k n_l$ for $k = 1, 2, \dots$
- The online updating method is elegant because **it only uses quantities $(\tilde{\beta}_{1:k-1}, \text{SSE}_{1:(k-1)}, \mathbf{V}_{k-1}, N_{k-1})$, which are of low dimension** from the previous accumulation point, to compute $\tilde{\beta}_{1:k}$ and $\tilde{\sigma}_{1:k}^2$.

Outline

- 1 Online Machine Learning
- 2 Basic Streaming Algorithms
- 3 Online Learning in Statistics
- 4 Recursive Algorithms
- 5 Online Updating Estimations
- 6 Renewable Estimation and Incremental Inference**
- 7 Parallel-and-Stream Accelerator
- 8 Online Nonparametric Estimation
- 9 Advanced Gradient Descent
- 10 Deep Bootstrap Framework

Problem Setup

- **Goal:** fit a GLM $\mathbb{E}(y_i | \mathbf{x}_i) = g(\mathbf{x}_i^\top \boldsymbol{\beta})$, $i = 1, \dots, N_b$, where N_b is the sample size of aggregated streaming data up to batch b , $N_b = \sum_{j=1}^b n_j$.
- For a GLM, write out the associated log-likelihood function:

$$l_{N_b}(\boldsymbol{\beta}, \phi; D_b^*) = \sum_{i \in D_b^*} \log \{a(y_i; \phi)\} - \frac{1}{2\phi} \sum_{i \in D_b^*} d(y_i; \mu_i)$$

- $D_b^* = \{D_1, \dots, D_b\}$: cumulative data
- $d(y_i; \mu_i)$: unit deviance function
- $\mu_i = \mathbb{E}(y_i | \mathbf{x}_i)$: mean parameter
- $a(\cdot)$: normalizing factor
- $\phi > 0$: dispersion parameter
- Denote the (unit) score function by $\mathbf{U}(y_i; \mathbf{x}_i, \boldsymbol{\beta}) := \nabla_{\boldsymbol{\beta}} d(y_i; \mu_i)$.
- Denote the single-batch negative Hessian by

$$\mathbf{J}_b(D_b; \boldsymbol{\beta}) := -\nabla_{\boldsymbol{\beta}} \mathbf{U}_b(D_b; \boldsymbol{\beta}).$$

Renewable Estimation

- Consider a simple scenario of two batches of data D_1 and D_2 , where D_2 arrives after D_1 .
- **Target:** update the initial MLE $\hat{\beta}_1$ to a renewed MLE $\hat{\beta}_2^*$ without using any subject level data but only some summary statistics from D_1 .
- $\hat{\beta}_2^*$ satisfies the following aggregated score equation:

$$\mathbf{U}_1(D_1; \hat{\beta}_2^*) + \mathbf{U}_2(D_2; \hat{\beta}_2^*) = \mathbf{0}. \quad (6.1)$$

Renewable Estimation

- To derive a **renewable estimate**, take the first-order Taylor series expansion of the first term in equation (6.1) around the MLE $\hat{\beta}_1$:

$$\mathbf{U}_1(D_1; \hat{\beta}_1) + \mathbf{J}_1(D_1; \hat{\beta}_1)(\hat{\beta}_1 - \tilde{\beta}_2) + \mathbf{U}_2(D_2; \tilde{\beta}_2) = \mathbf{0},$$

where $\tilde{\beta}_2$ is an approximation of the oracle MLE $\hat{\beta}_2^*$.

- The proposed estimator $\tilde{\beta}_2$ satisfies

$$\mathbf{J}_1(D_1; \hat{\beta}_1)(\hat{\beta}_1 - \tilde{\beta}_2) + \mathbf{U}_2(D_2; \tilde{\beta}_2) = \mathbf{0}.$$

- Numerically**, we can find $\tilde{\beta}_2$ by, for example, the Newton-Raphson algorithm, i.e., at the $(r + 1)$ th iteration,

$$\tilde{\beta}_2^{(r+1)} = \tilde{\beta}_2^{(r)} + \{\mathbf{J}_1(D_1; \hat{\beta}_1) + \mathbf{J}_2(D_2; \tilde{\beta}_2^{(r)})\}^{-1} \{\mathbf{J}_1(D_1; \hat{\beta}_1)(\hat{\beta}_1 - \tilde{\beta}_2^{(r)}) + \mathbf{U}_2(D_2; \tilde{\beta}_2^{(r)})\}.$$

Renewable Estimation

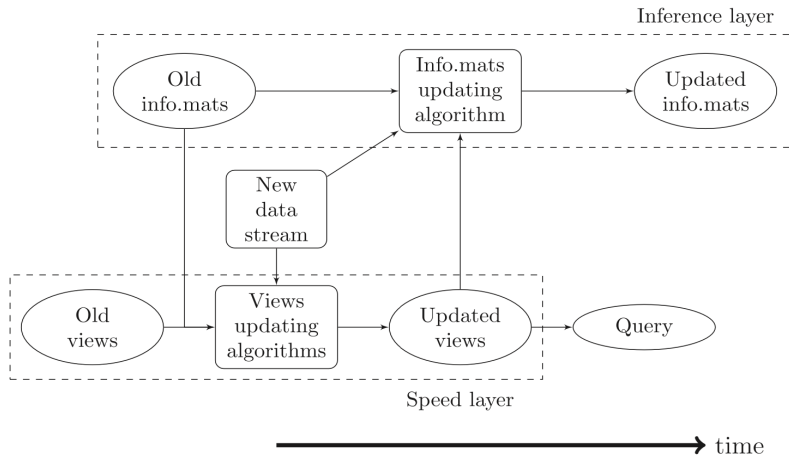
- To speed up the calculations, we may avoid updating $\mathbf{J}_2(D_2, \tilde{\beta}_2^{(r)})$ at each iteration:

$$\tilde{\beta}_2^{(r+1)} = \tilde{\beta}_2^{(r)} + \left\{ \mathbf{J}_1(\hat{\beta}_1) + \mathbf{J}_2(\hat{\beta}_1) \right\}^{-1} \tilde{\mathbf{U}}_2^{(r)},$$

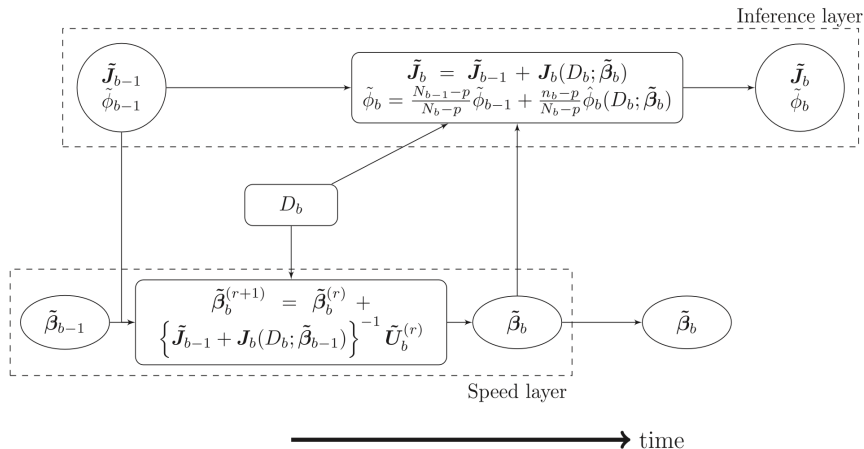
where $\tilde{\mathbf{U}}_2^{(r)} = \mathbf{J}_1(D_1; \hat{\beta}_1)(\hat{\beta}_1 - \tilde{\beta}_2^{(r)}) + \mathbf{U}_2(D_2; \tilde{\beta}_2^{(r)})$.

- $\tilde{\beta}_2$ is iteratively solved by using the adjusted score function $\tilde{\mathbf{U}}_2^{(r)}$ and the aggregated negative Hessian $\{\mathbf{J}_1(\hat{\beta}_1) + \mathbf{J}_2(\hat{\beta}_1)\}$ evaluated at the previous estimate $\hat{\beta}_1$.

Renewable Estimation and Incremental Inference



Renewable Estimation and Incremental Inference



Simulation Experiments

- Generate the full data set D_B^* independently from the respective GLMs with the mean model $\mathbb{E}(y_i | \mathbf{x}_i) = g(\mathbf{x}_i^\top \boldsymbol{\beta}_0)$, $i = 1, \dots, N_B$.
- Set $\boldsymbol{\beta}_0 = (0.2, -0.2, 0.2, -0.2, 0.2)^\top$.
- Compare the renewable method with several leading methods in the current literature:
 - the oracle MLE obtained by processing the entire data once
 - AISGD
 - the sequential estimation method of the OLSE in the linear model
 - the sequential estimation method of the CEE or CUEE for non-linear GLMs

Simulation Experiments

	<i>Results for $B = 10$ and $N_B = 10^3$</i>					<i>Results for $B = 100$ and $N_B = 10^4$</i>				
	<i>MLE</i>	<i>AISGD</i>	<i>CEE</i>	<i>CUEE</i>	<i>Renew</i>	<i>MLE</i>	<i>AISGD</i>	<i>CEE</i>	<i>CUEE</i>	<i>Renew</i>
Abias $\times 10^{-3}$	61.59	63.18	58.71	60.78	60.97	19.59	24.14	20.80	19.93	19.55
ASE $\times 10^{-3}$	78.70	58.34	81.07	79.38	79.15	24.73	28.40	25.53	24.93	24.76
ESE $\times 10^{-3}$	77.32	78.63	73.05	76.30	76.56	24.50	30.23	22.99	24.81	24.44
CP	0.96	0.83	0.97	0.96	0.96	0.95	0.92	0.95	0.95	0.95
CTime (s)	0.01	—	0.03	0.06	0.01	0.08	—	0.34	0.63	0.07
RTime (s)	0.007	0.008	0.028	0.056	0.006	0.045	0.064	0.311	0.599	0.047
	<i>Results for $B = 10^3$ and $N_B = 10^5$</i>					<i>Results for $B = 10^4$ and $N_B = 10^6$</i>				
Abias $\times 10^{-3}$	6.23	23.44	12.63	7.66	6.22	1.92	23.44	12.43	4.67	1.92
ASE $\times 10^{-3}$	7.82	27.94	8.07	7.88	7.82	2.47	27.94	2.55	2.49	2.47
ESE $\times 10^{-3}$	7.78	29.39	7.31	9.42	7.78	2.42	29.39	2.28	5.98	2.42
CP	0.95	0.94	0.68	0.90	0.95	0.95	0.94	0	0.67	0.95
CTime (s)	2.88	—	3.056	5.74	0.64	343.5	—	32.60	56.51	6.46
RTime (s)	0.51	0.19	2.84	5.50	0.47	7.04	0.98	28.85	54.04	4.66

† N_B increases from 10^3 to 10^6 . Results are summarized from 500 replications.

Simulation Experiments: Lager p

<i>Results for the following methods</i>					
	<i>AISGD</i>	<i>MLE</i>	<i>CEE</i>	<i>CUEE</i>	<i>Renew</i>
<i>p</i> = 1000					
Abias	25.799	2.176	3.880	2.242	2.152
ASE	1.70×10^{-3}	2.705	2.904	2.668	2.707
ESE	1.72×10^{-3}	2.715	2.358	2.616	2.673
CP	0	0.948	0.757	0.937	0.951
CTime (min)	—	17.959	17.288	20.470	4.207
RTime (min)	1.609	16.686	17.093	20.258	4.014
<i>p</i> = 2500					
Abias	16.386	2.212	6.994	2.581	2.192
ASE	1.71×10^{-3}	2.728	3.475	2.523	2.789
ESE	1.72×10^{-3}	2.745	1.804	2.442	2.715
CP	0	0.946	0.561	0.874	0.954
CTime (min)	—	126.407	122.528	149.411	31.451
RTime (min)	4.737	123.904	122.037	148.924	30.917

†The number of covariates, p , varies from 1000 to 2500.

Outline

- 1 Online Machine Learning
- 2 Basic Streaming Algorithms
- 3 Online Learning in Statistics
- 4 Recursive Algorithms
- 5 Online Updating Estimations
- 6 Renewable Estimation and Incremental Inference
- 7 Parallel-and-Stream Accelerator**
- 8 Online Nonparametric Estimation
- 9 Advanced Gradient Descent
- 10 Deep Bootstrap Framework

Serial and Parallel Computing

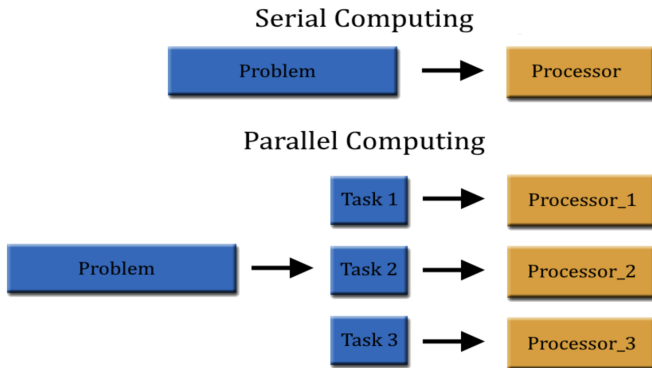


Figure: Serial versus parallel computing.

Trade-Offs

Trade-offs between **serial and parallel computing**:

- Serial computing is more computationally expensive since it does not leverage parallel computing, but has no loss of statistical power.
- Divide-and-combine is computationally fast but results in a loss of statistical efficiency.

Parallel-and-Stream Accelerator (PASA)

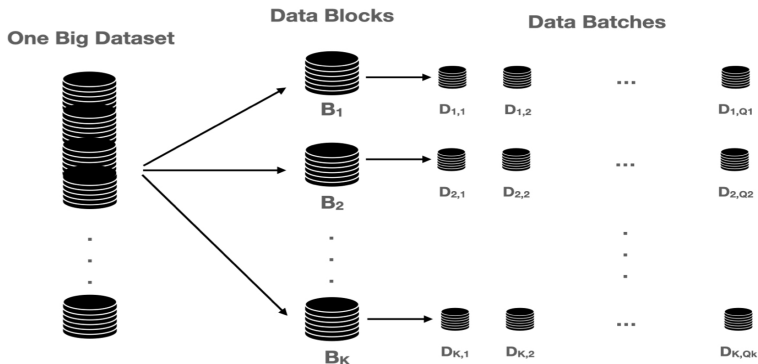


Figure: Flowchart of PASA: A big dataset is first randomly partitioned into K parallel data blocks, and each data block is then processed sequentially via renewable learning.

Setup

- Consider the regression setting with outcome y drawn from a common parametric distribution conditional on p explanatory variables $\mathbf{x} = (x_1, \dots, x_p)^\top : (y_i; \mathbf{x}_i) \sim f(y; \mathbf{x}, \boldsymbol{\beta}_0, \phi_0), i = 1, \dots, N$.
- Consider the GLM setting in which the distribution $f(y; \mathbf{x}, \boldsymbol{\beta}, \phi)$ belongs to the family of exponential dispersion models.
- Suppose the sample size N is so big that a centralized analysis of the data is computationally prohibitive on one computing node.

PASA

- PASA randomly splits the N independent observations to form K disjoint blocks.
- The primary task is to conduct both estimation and inference for β over the entire data.
- Given the data split, this task becomes a divide-and-combine procedure:
 - ① estimate β and related inferential quantities in block k (renewable estimation, covered before)
 - ② integrate estimators from all blocks into one unified estimator and inferential procedure (combining algorithm)

Combining Algorithm

Once block estimators $\hat{\beta}_k, k = 1, \dots, K$ obtained using the streaming algorithm, these block-level estimations can be combined using a closed form **Best Linear Unbiased Estimation (BLUE)** update:

$$\hat{\beta}_{PASA} = \left(\sum_{k=1}^K n_k \hat{\phi}_k^{-1} \hat{\mathbf{J}}_k \right)^{-1} \sum_{k=1}^K n_k \hat{\phi}_k^{-1} \hat{\mathbf{J}}_k \hat{\beta}_k.$$

Numerical Performance

- The full dataset is randomly partitioned into K data blocks, and data within each block are processed in a sequence of $Q_k \equiv Q, k = 1, \dots, K$, data batches.
- Compare PASA with:
 - the offline MLE
 - the MapReduce estimator by parallelizing K data blocks.
- Study the effect of **different number of blocks K** and **different number of batches Q** on point estimation and computation efficiency of the three methods, respectively.

Numerical Performance

Linear Model													
$N = 10^5$	Offline MLE	MapReduce				PASA, fixed Q				PASA, fixed K			
K	1	1	10	100	1000	1	10	100	1000	10			
Q	1	1				10				1	10	100	1000
A.bias $\times 10^{-3}$	3.067	3.067	3.067	3.070	3.101	3.067	3.067	3.070	3.101	3.067	3.067	3.067	3.067
ASE $\times 10^{-3}$	3.833	3.833	3.832	3.829	3.792	3.833	3.832	3.829	3.792	3.832	3.832	3.832	3.832
ESE $\times 10^{-3}$	3.822	3.822	3.823	3.827	3.859	3.822	3.823	3.827	3.859	3.823	3.823	3.823	3.823
CP	0.950	0.950	0.950	0.948	0.947	0.950	0.950	0.948	0.947	0.950	0.950	0.950	0.950
C.Time(s)	0.278	0.205	0.038	0.012	0.010	0.082	0.019	0.016	0.024	0.010	0.021	0.060	0.408
R.Time(s)	0.236	0.166	0.030	0.011	0.010	0.041	0.012	0.011	0.012	0.005	0.014	0.033	0.184

Figure: Simulation results under the linear model summarized over 500 replications, with fixed $N = 10^5, p = 5$.

Numerical Performance

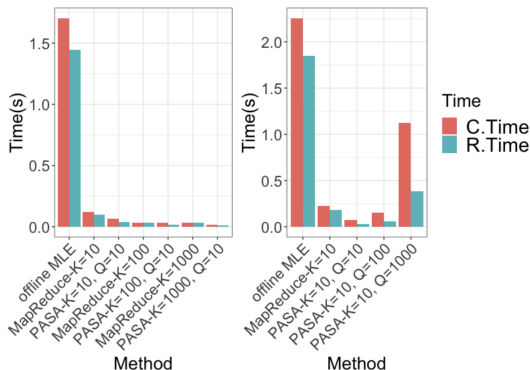


Figure: Computation time for the linear regression model with $N = 10^6$.

Outline

- 1 Online Machine Learning
- 2 Basic Streaming Algorithms
- 3 Online Learning in Statistics
- 4 Recursive Algorithms
- 5 Online Updating Estimations
- 6 Renewable Estimation and Incremental Inference
- 7 Parallel-and-Stream Accelerator
- 8 Online Nonparametric Estimation**
- 9 Advanced Gradient Descent
- 10 Deep Bootstrap Framework

Online Nonparametric Regression

- Motivation example:

$$Y_i = r(X_i) + \varepsilon_i$$

- Datasets are available sequentially as $\mathbf{d}_j = \{(Y_i, X_i) : i \in \mathbf{i}_j\}$ with index sets $\mathbf{i}_j$ and $j = 1, 2, \dots$.
- Denote $\mathcal{D}_k = \bigcup_{j=1}^k \mathbf{d}_j$ and $\mathcal{I}_k = \bigcup_{j=1}^k \mathbf{i}_j$.
- The **Nadaraya-Watson** estimator $r(x)$ computed on $\mathcal{D}_k$ is defined as

$$\hat{r}_k(x) = \frac{\sum_{i \in \mathcal{I}_k} Y_i K_h(X_i - x)}{\sum_{i \in \mathcal{I}_k} K_h(X_i - x)}.$$

- $K_h(\cdot) = \frac{1}{h} K\left(\frac{\cdot}{h}\right)$ with $K(\cdot)$ and h be kernel function and bandwidth, respectively.

Online Nonparametric Regression

- $\hat{r}_k(x)$ satisfies

$$\sum_{i \in \mathcal{I}_{k-1}} K_h(X_i - x)(\hat{r}_k(x) - \hat{r}_{k-1}(x)) - \sum_{i \in \mathcal{I}_k} (Y_i - \hat{r}_k(x))K_h(X_i - x) = 0$$

with initial estimator $\hat{r}_0(x) \equiv 0$.

- The estimator $\hat{r}_k(x)$ could be thought of as **the online updating version** from the prior estimator $\hat{r}_{k-1}(x)$ in the sense of estimating equation.
- In other words, $\hat{r}_k(x)$ is the solution to the **incremental equation**:

$$\sum_{i \in \mathcal{I}_{k-1}} K_h(X_i - x)(r(x) - \hat{r}_{k-1}(x)) - \sum_{i \in \mathcal{I}_k} (Y_i - r(x))K_h(X_i - x) = 0,$$

- or equivalently, it is the solution to the **incremental optimization problem**:

$$\min_{r(x)} \sum_{i \in \mathcal{I}_{k-1}} K_h(X_i - x)(r(x) - \hat{r}_{k-1}(x))^2 + \sum_{i \in \mathcal{I}_k} (Y_i - r(x))^2 K_h(X_i - x).$$

Online Nonparametric Regression

- The last two equations are **online updating estimating equation** and **online updating loss function**, respectively.
- only involve the current data set $\mathbf{d}_k$ and the previous estimator $\hat{r}_{k-1}(x)$ together with the accumulative quantity $\sum_{i \in \mathcal{I}_{k-1}} K_h(X_i - x)$.
- The above incremental updating procedures can be extended into general cases, for example, smoothing spline estimation and local polynomial estimation in nonparametric models.

Another Perspective

- Updating weighted sum:

$$\hat{\theta}_2(\mathbf{d}_1, \mathbf{d}_2) = \arg \max_{\theta} \{l(\theta; \mathbf{d}_2) - (\hat{\theta}_1(\mathbf{d}_1) - \theta)^\top A (\hat{\theta}_1(\mathbf{d}_1) - \theta)\}$$

- θ is an unknown parameter vector, $l(\theta; \mathbf{d}_2)$ is the likelihood of θ computed on the current dataset $\mathbf{d}_2$.
- $\hat{\theta}_1(\mathbf{d}_1)$ is a previous estimator of θ computed on the historical dataset $\mathbf{d}_1$, and A is a weight matrix proportional to the inverse of the asymptotic covariance of $\hat{\theta}_1(\mathbf{d}_1)$.
- aggregates the **weighted least squares** centralized at the previous estimator $\hat{\theta}_1(\mathbf{d}_1)$ and **the current likelihood** $l(\theta; \mathbf{d}_2)$ respectively from two samples $\mathbf{d}_1$ and $\mathbf{d}_2$ to construct an **updating likelihood** and then get an updating estimator $\hat{\theta}_2(\mathbf{d}_1, \mathbf{d}_2)$.

Some Interesting Questions

- How to do real-time analysis on streaming and **confidential data**?
- In real-time data streaming, how to deal with the negative effect of **poisoned data batches**?
- How to address the problem of **distribution shift** that is common in real life when we have streaming data?

References I



Luo, L. and Song, P.X.

Renewable estimation and incremental inference in generalized linear models with streaming data sets

J. R. Statist. Soc. B, 82: 69-97, 2020.



Wang, C., Chen, M., Wu, J., Yan, J., Zhang, Y., and Schifano, E.

Online updating method with new variables for big data streams

Canadian Journal of Statistics, 46(1): 123-146, 2018.



Lin, L., Li, W., and Lu, J.

Unified rules of renewable weighted sums for various online updating estimations

arXiv preprint arXiv:2008.08824, 2020.



Hector, E.C., Luo, L., and Song, P.X.

Parallel-and-stream accelerator for computationally fast supervised learning

Computational Statistics & Data Analysis, 177, 2023.

Outline

- 1 Online Machine Learning
- 2 Basic Streaming Algorithms
- 3 Online Learning in Statistics
- 4 Recursive Algorithms
- 5 Online Updating Estimations
- 6 Renewable Estimation and Incremental Inference
- 7 Parallel-and-Stream Accelerator
- 8 Online Nonparametric Estimation
- 9 Advanced Gradient Descent**
- 10 Deep Bootstrap Framework

Overview

- While gradient descent is simple and intuitive, it has many problems as well.
 - Saddle-point problem
 - Not applicable to non-differentiable objectives
 - Could be slow
 - **How to scale to big data problems**
- In this lecture, we will discuss some advanced techniques that can alleviate these problems

Momentum Method

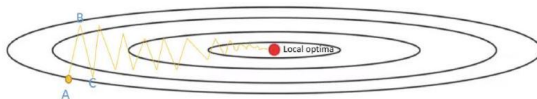
- Introduced in 1964 by Polyak, momentum method is a technique that can accelerate gradient descent by taking accounts of previous gradients in the update rule at each iteration.

$$\begin{aligned}m^{(k)} &= \mu m^{(k-1)} + (1 - \mu) \nabla f(x^{(k)}) \\x^{(k+1)} &= x^{(k)} - \alpha m^{(k)}\end{aligned}$$

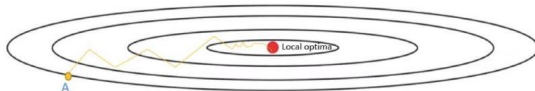
- where $0 \leq \mu < 1$
- When $\mu = 0$, gradient descent is recovered.

How Does Momentum Work?

- The vanilla gradient descent may suffer from oscillations when the magnitudes of gradient varies a lot across different directions.



- Using the exponential weighted gradient (momentum), those oscillations are more likely to be damped out, resulting in faster rate of convergence.



Nesterov's Acceleration

- Choose any initial $x^{(0)} = x^{(-1)}, \forall k = 1, 2, 3, \dots$

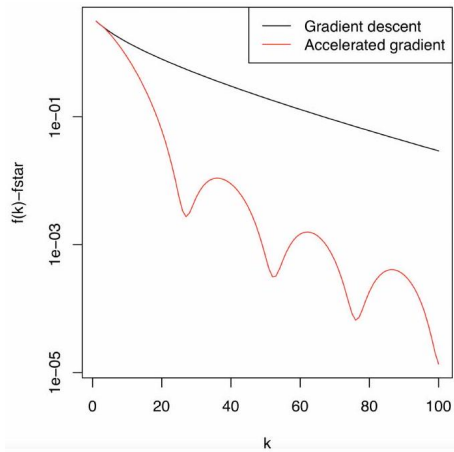
$$y = x^{(k-1)} + \frac{k-2}{k+1}(x^{(k-1)} - x^{(k-2)})$$

$$x^{(k)} = y - t_k \nabla f(y)$$

- The first two steps are the usually gradient updates
- After that, $y = x^{(k-1)} + \frac{k-2}{k+1}(x^{(k-1)} - x^{(k-2)})$ carries some “momentum” from previous iterations, and
- $x^{(k)} = y - t_k \nabla f(y)$ uses **lookahead gradient** at y .

Example

Logistic regression



Convergence Rate of Gradient Methods

Assumptions

- f is convex and continuously differentiable on $\mathbb{R}^n$
- $\nabla f(x)$ is L -Lipschitz continuous w.r.t Euclidean norm: for any $x, y \in \mathbb{R}^n$

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

- optimal value $f^* = \inf_x f(x)$ is finite and attained at x^*

Theorem: Gradient descent with $0 < t \leq 1/L$ satisfies

$$f(x^{(k)}) - f^* \leq \frac{1}{2kt} \|x^{(0)} - x^*\|^2$$

Some Useful Lemma and Strong Convexity

- If f is L -Lipschitz, then for any $x, y \in \mathbb{R}^n$

$$f(y) \leq f(x) + \nabla f(x)^T (y - x) + \frac{L}{2} \|y - x\|^2$$

- If f is differentiable and m -strongly convex, then

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{m}{2} \|y - x\|^2$$

If $m = 0$, we cover the standard(weak) convexity

In other words, f is **sandwiched** between two quadratic functions

Proof

- If $x^+ = x - t\nabla f(x)$ and $0 < t \leq 1/L$

$$\begin{aligned} f(x^+) &\leq f(x) - t\|\nabla f(x)\|^2 + \frac{t^2 L}{2} \|\nabla f(x)\|^2 \\ &\leq f(x) - \frac{t}{2} \|\nabla f(x)\|^2 \end{aligned}$$

- From convexity

$$f(x) \leq f^* + \nabla f(x)^T (x - x^*) - \frac{m}{2} \|x - x^*\|^2$$

- Add the above two inequalities

$$f(x^+) - f^* \leq \nabla f(x)^T (x - x^*) - \frac{t}{2} \|\nabla f(x)\|^2 - \frac{m}{2} \|x - x^*\|^2$$

Proof

- Continue ...

$$\begin{aligned} &\leq \frac{1}{2t} (\|x - x^*\|^2 - \|x^+ - x^*\|^2) - \frac{m}{2} \|x - x^*\|^2 \\ &= \frac{1}{2t} ((1 - mt) \|x - x^*\|^2 - \|x^+ - x^*\|^2) \end{aligned} \quad (1)$$

$$\leq \frac{1}{2t} (\|x - x^*\|^2 - \|x^+ - x^*\|^2) \quad (2)$$

- For gradient descent updates

$$\begin{aligned} \sum_{i=1}^k (f(x^{(i)}) - f^*) &\leq \frac{1}{2t} \sum_{i=1}^k (\|x^{(i-1)} - x^*\|^2 - \|x^{(i)} - x^*\|^2) \\ &= \frac{1}{2t} (\|x^{(0)} - x^*\|^2 - \|x^{(k)} - x^*\|^2) \end{aligned}$$

Proof

- Since $f(x^{(i)})$ is non-increasing

$$f(x^{(k)}) - f^* \leq \frac{1}{2kt} \|x^{(0)} - x^*\|^2$$

- If f is m -strongly convex, and $m > 0$, from (1)

$$\|x^{(i)} - x^*\|^2 \leq (1 - mt) \|x^{(i-1)} - x^*\|^2, \quad \forall i = 1, 2, \dots$$

- Therefore

$$\|x^{(k)} - x^*\|^2 \leq (1 - mt)^k \|x^{(0)} - x^*\|^2$$

i.e., linear convergence if f is strongly convex ($m > 0$)

Oracle Lower Bound of First-order Methods

- First order method: any iterative algorithm that selects $x^{(k+1)}$ in the set

$$x^{(0)} + \text{span}\{\nabla f(x^{(0)}), \nabla f(x^{(1)}), \dots, \nabla f(x^{(k)})\}$$

- **Theorem (Nesterov):** for every integer $k \leq (n - 1)/2$ and every $x^{(0)}$, there exist functions that satisfy the assumptions such that for any first-order method

$$f(x^{(k)}) - f^* \geq \frac{3}{32} \cdot \frac{L\|x^{(0)} - x^*\|^2}{(k + 1)^2}$$

- Therefore, $1/k^2$ is the best convergence rate for all first-order methods.

Convergence Rate of Nesterov's Acceleration

- Accelerated gradient descent with fixed step size $t \leq 1/L$ satisfies

$$f(x^{(k)}) - f^* \leq \frac{2\|x^{(0)} - x^*\|^2}{t(k+1)^2}$$

- Nesterov's accelerated gradient (NAG) descent achieve the oracle convergence rate of first-order methods!

Reformulation of NAG

- Initialize $x^{(0)} = u^{(0)}$, and for $k = 1, 2, \dots$

$$\begin{aligned}y &= (1 - \theta_k)x^{(k-1)} + \theta_k u^{(k-1)} \\x^{(k)} &= y - t_k \nabla f(y) \\u^{(k)} &= x^{(k)} + \frac{1}{\theta_k}(x^{(k)} - x^{(k-1)})\end{aligned}$$

with $\theta_k = 2/(k+1)$.

- This is equivalent to the formulation of NAG presented earlier (slide 5), and makes convergence analysis easier

Proof

- If $y = (1 - \theta)x + \theta u$, $x^+ = y - t\nabla f(y)$, and $0 < t \leq 1/L$

$$f(x^+) \leq f(y) + \nabla f(y)^T(x^+ - y) + \frac{1}{2t}\|x^+ - y\|^2$$

- From convexity, $\forall z \in \mathbb{R}^n$

$$f(y) \leq f(z) + \nabla f(y)^T(y - z)$$

- Add these together

$$f(x^+) \leq f(z) + \frac{1}{t}(x^+ - y)(z - x^+) + \frac{1}{2t}\|x^+ - y\|^2 \quad (3)$$

Proof

- Let $u^+ = x + \frac{1}{\theta}(x^+ - x)$, using bound (3) at $z = x$ and $z = x^*$

$$\begin{aligned}
 f(x^+) - f^* &= (1 - \theta)(f(x) - f^*) \\
 &\leq \frac{1}{t}(x^+ - y)^T(\theta x^+ + (1 - \theta)x - x^*) + \frac{1}{2t}\|x^+ - y\|^2 \\
 &= \frac{\theta^2}{2t}(\|u - x^*\|^2 - \|u^+ - x^*\|^2)
 \end{aligned}$$

- i.e.*, at iteration k

$$\begin{aligned}
 &\frac{t}{\theta_k^2}(f(x^{(k)}) - f^*) + \frac{1}{2}\|u^{(k)} - x^*\|^2 \\
 &\leq \frac{(1 - \theta_k)t}{\theta_k^2}(f(x^{(k-1)}) - f^*) + \frac{1}{2}\|u^{(k-1)} - x^*\|^2
 \end{aligned}$$

Proof

- Using $(1 - \theta_i)/\theta_i^2 \leq 1/\theta_{i-1}^2$, and iterating this inequality

$$\begin{aligned}
 & \frac{t}{\theta_k^2} (f(x^{(k)}) - f^*) + \frac{1}{2} \|u^{(k)} - x^*\|^2 \\
 & \leq \frac{(1 - \theta_1)t}{\theta_1^2} (f(x^{(0)}) - f^*) + \frac{1}{2} \|u^{(0)} - x^*\|^2 \\
 & = \frac{1}{2} \|x^{(0)} - x^*\|^2
 \end{aligned}$$

- Therefore

$$f(x^{(k)}) - f^* \leq \frac{\theta_k^2}{2t} \|x^{(0)} - x^*\|^2 = \frac{2}{t(k+1)^2} \|x^{(0)} - x^*\|^2$$

Why NAG works?

- Although the algebraic manipulations of the proof is beautiful, the acceleration effect in NAG has been mysterious and hard to understand
- Recent works reinterpreted the NAG algorithm from different point of views, including Zhu et al (2017) and Su et al (2014)
- Here we introduce the ODE explanation from Su et al (2014)

An ODE Explanation

- Su et al (2014) proposed an ODE based explanation where NAG can be viewed as a discretization of the following ordinary differential equation

$$\ddot{X} + \frac{3}{t}\dot{X} + \nabla f(X) = 0, \quad t > 0 \quad (4)$$

with initial conditions $X(0) = x^{(0)}, \dot{X}(0) = 0$.

- **Theorem** (Su et al): For any $f \in \mathcal{F}_\infty \triangleq \bigcap_{L>0} \mathcal{F}_L$ and any $x^{(0)} \in \mathbb{R}^n$, the ODE (4) with initial conditions $X(0) = x^{(0)}, \dot{X}(0) = 0$ has a unique global solution $X \in C^2((0, \infty); \mathbb{R}^n) \cap C^1([0, \infty); \mathbb{R}^n)$.

Convergence Rate of The ODE Solution

- **Theorem** (Su et al): For any $f \in \mathcal{F}_\infty$, let $X(t)$ be the unique global solution to (4) with initial conditions $X(0) = x^{(0)}$, $\dot{X}(0) = 0$. For any $t > 0$,

$$f(X(t)) - f^* \leq \frac{2\|x^{(0)} - x^*\|^2}{t^2}$$

- Consider the energy functional defined as

$$\mathcal{E}(t) \triangleq t^2(f(X(t)) - f^*) + 2\|X + \frac{t}{2}\dot{X} - x^*\|^2$$

- The derivative of the energy function is

$$\dot{\mathcal{E}} = 2t(f(X) - f^*) + t^2\langle \nabla f, \dot{X} \rangle + 4\langle X + \frac{t}{2}\dot{X} - x^*, \frac{3}{2}\dot{X} + \frac{t}{2}\ddot{X} \rangle$$

Convergence Rate of The ODE Solution

- Substituting $3\dot{X}/2 + t\ddot{X}/2$ with $-t\nabla f(X)/2$

$$\begin{aligned}\dot{\mathcal{E}} &= 2t(f(X) - f^*) + 4\langle X - x^*, -\frac{t}{2}\nabla f(X)\rangle \\ &= 2t(f(X) - f^*) - 2t\langle X - x^*, \nabla f(X)\rangle \\ &\leq 0\end{aligned}$$

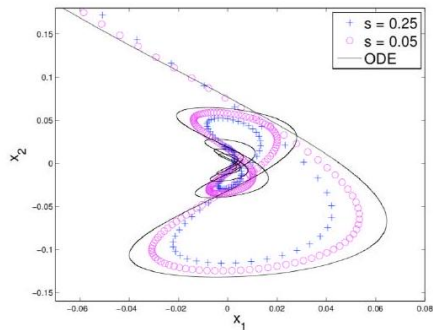
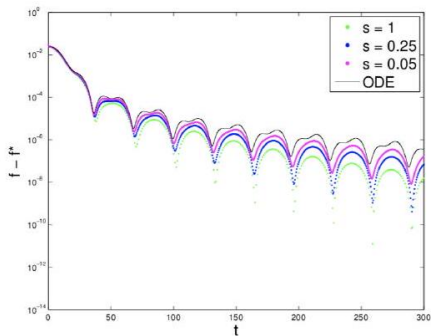
where the last inequality follows from the convexity of f .

- Therefore,

$$f(X(t)) - f^* \leq \mathcal{E}(t)/t^2 \leq \mathcal{E}(0)/t^2 = \frac{2\|x^{(0)} - x^*\|^2}{t^2}$$

Example

$$f(x) = 0.02x_1^2 + 0.005x_2^2, \quad x^{(0)} = (1, 1)$$



Proximal Gradient Descent: Motivation

The objective in many unconstrained optimization problems can be split in two components

$$\text{minimize } f(x) = g(x) + h(x)$$

- g is convex and differentiable on $\mathbb{R}^n$
- h is convex and simple, but may be non-differentiable

Examples

- Indicator function of closed convex set C

$$h(x) = 1_C(x) = \begin{cases} 0, & x \in C \\ +\infty, & x \notin C \end{cases}$$

- L_1 regularization (LASSO): $h(x) = \|x\|_1$

Proximal Mapping

The **proximal mapping** (or **proximal-operator**) of a convex function h is defined as

$$\text{prox}_h(x) = \arg \min_u \left(h(u) + \frac{1}{2} \|u - x\|_2^2 \right)$$

Examples

- $h(x) = 0$: $\text{prox}_h(x) = x$
- $h(x) = 1_C(x)$: prox_h is projection on C

$$\text{prox}_h(x) = \arg \min_{u \in C} \|u - x\|_2^2 = P_C(x)$$

- $h(x) = \|x\|_1$: prox_h is the "soft-threshold" (shrinkage) operation

$$\text{prox}_h(x_i) = \begin{cases} x_i - 1 & x_i \geq 1 \\ 0 & |x_i| \leq 1 \\ x_i + 1 & x_i \leq -1 \end{cases}$$

Proximal Gradient Descent

- Proximal gradient algorithm**

$$x^{(k+1)} = \text{prox}_{t_k h}(x^{(k)} - t_k \nabla g(x^{(k)})), \quad k = 0, 1, \dots$$

- Interpretation. If $x^+ = \text{prox}_{th}(x - t \nabla g(x))$, from the definition of proximal mapping

$$\begin{aligned} x^+ &= \arg \min_u \left(h(u) + \frac{1}{2t} \|u - x + t \nabla g(x)\|_2^2 \right) \\ &= \arg \min_u \left(h(u) + g(x) + \nabla g(x)^T (u - x) + \frac{1}{2t} \|u - x\|_2^2 \right) \end{aligned}$$

- x^+ minimizes $h(u)$ plus a simple quadratic local approximation of $g(u)$ around x

Examples

- **Gradient Descent:** special case with $h(x) = 0$

$$x^+ = x - t \nabla g(x)$$

- **Projected Gradient Descent:** special case with $h(x) = 1_C(x)$

$$x^+ = P_C(x - t \nabla g(x))$$

- **ISTA (Iterative Shrinkage-Thresholding Algorithm):** special case with $h(x) = \|x\|_1$

$$x^+ = \mathcal{S}_t(x - t \nabla g(x))$$

where

$$\mathcal{S}_t(u) = (|u| - t)_+ \cdot \text{sign}(u)$$

Convergence Rate of Proximal Gradient Descent

- If h is convex and closed,

$$\text{prox}_h(x) = \arg \min_u \left(h(u) + \frac{1}{2} \|u - x\|_2^2 \right)$$

exists and is unique for all x . Moreover, it has the following useful properties

$$u = \text{prox}_h(x) \iff x - u \in \partial h(u) \iff h(z) \geq h(u) + (x - u)^T (z - u), \forall z$$

- Proximal gradient descent has the same convergence rate as gradient descent when $0 < t \leq 1/L$

$$f(x^{(k)}) - f^* \leq \frac{1}{2kt} \|x^{(0)} - x^*\|_2^2$$

Accelerated Proximal Gradient Descent

- Similarly, we can apply Nesterov's acceleration for proximal gradient descent. Choose any initial $x^{(0)} = x^{(-1)}, \forall k = 1, \dots$

$$y = x^{(k-1)} + \frac{k-2}{k+1}(x^{(k-1)} - x^{(k-2)})$$

$$x^{(k)} = \text{prox}_{t_k h}(y - t_k \nabla g(y))$$

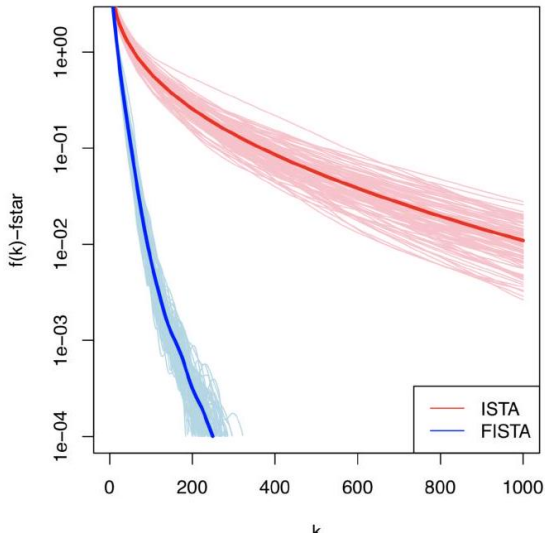
- Convergence rate is the same with NAG if $0 < t \leq 1/L$

$$f(x^{(k)}) - f^* \leq \frac{2\|x^{(0)} - x^*\|^2}{t(k+1)^2}$$

- When applied to LASSO, this is called **FISTA** (**F**ast **I**terative **S**hrinkage-**T**hresholding **A**lgorithm)

Example: ISTA vs FISTA

LASSO Logistic regression: 100 instances



Stochastic Optimization

Consider the following stochastic optimization problem

$$\min_x f(x) = \mathbb{E}_\xi(F(x, \xi)) = \int F(x, \xi)p(\xi)d\xi$$

- ξ is a random variable
- The challenge: evaluation of the expectation/integration

Example

- Supervised Learning

$$\min_w f(w) = \mathbb{E}_{(x,y) \sim D(x,y)}(\ell(h_w(x), y))$$

where $D(x, y)$ is the data distribution, $\ell(\cdot, \cdot)$ is certain loss, w is the model parameter

Stochastic Gradient Descent

- Gradient descent with stochastic approximation (SA)

$$x^{(k+1)} = x^{(k)} - t_k g(x^{(k)})$$

where $\mathbb{E}(g(x)) = \nabla f(x), \forall x$

- Example. Consider supervised learning with observations $\mathcal{D} = \{x_i, y_i\}_{i=1}^N$

$$\min_w f(w) = \frac{1}{N} \sum_{i=1}^N \ell(h_w(x^{(i)}, y^{(i)}))$$

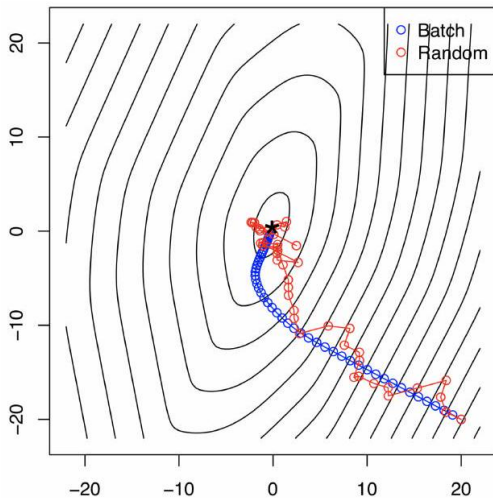
SGD

$$w^{(k+1)} = w^{(k)} - t_k \nabla \ell(h_w(x^{(i_k)}, y^{(i_k)}))$$

where $i_k \in \{1, \dots, m\}$ is some chosen index at iteration k .

Example

Stochastic logistic regression



Convergence Rate of SGD

- Assume that $\mathbb{E}(\|g(x)\|^2) \leq M^2$ and $f(x)$ is convex

$$\mathbb{E}(f(\bar{x}^{[0:k]}) - f^*) \leq \frac{\|x^{(0)} - x^*\|^2 + M^2 \sum_{j=0}^k t_j^2}{2 \sum_{j=0}^k t_j}$$

where $\bar{x}^{[0:k]} = \sum_{j=1}^k t_j x^{(j)} / \sum_{j=1}^k t_j$

- Fix the number of iterations K and constant step sizes

$t_j = \frac{\|x^{(0)} - x^*\|}{M\sqrt{K}}$, $j = 0, 1, \dots, K$, we have

$$\mathbb{E}(f(\bar{x}^K) - f^*) \leq \frac{\|x^{(0)} - x^*\| M}{\sqrt{K}}$$

where $\bar{x}^K = \frac{1}{K+1} \sum_{j=0}^K x^{(j)}$

Proof

By convexity, we have $f(x^{(k)}) - f^* \leq \nabla f(x^{(k)})^T (x^{(k)} - x^*)$

$$\begin{aligned} t_k \mathbb{E}(f(x^{(k)})) - t_k f^* &\leq t_k \mathbb{E}(g(x^{(k)})^T (x^{(k)} - x^*)) \\ &= \frac{1}{2} (\mathbb{E} \|x^{(k)} - x^*\|^2 - \mathbb{E} \|x^{(k+1)} - x^*\|^2) + \frac{1}{2} t_k^2 \mathbb{E} \|g(x^{(k)})\|^2 \\ &\leq \frac{1}{2} (\mathbb{E} \|x^{(k)} - x^*\|^2 - \mathbb{E} \|x^{(k+1)} - x^*\|^2) + \frac{1}{2} t_k^2 M^2 \end{aligned}$$

$\forall k \geq 0$. Therefore

$$\sum_{j=0}^k t_j \mathbb{E}(f(x^{(j)})) - \sum_{j=0}^k t_j f^* \leq \frac{1}{2} \|x^{(0)} - x^*\|^2 + \frac{M^2}{2} \sum_{j=0}^k t_j^2$$

Dividing both side with $\sum_{j=0}^k t_j$ together with convexity complete the proof

Pros and Cons of Vanilla SGD

What We Love About SGD

- Efficient in computation and memory usage, naturally scalable for big data problems
- Less likely to be trapped at local modes

What Needs to Be Improved

- In general, vanilla SGD is slow to converge (only $1/k$ even with strong convexity). Variance reduction seems to be a good remedy, see algorithms like SVRG, SAGA, etc.
- Choosing a proper learning rate can be difficult, require much effort in hyperparameter tuning to get good results
- The same learning rate applies to all parameter updates

Inspiration From Fisher Scoring

- Assume that f can be related to a probabilistic model, *i.e.*

$$f(\theta) = -\mathbb{E}_{y \sim P_{data}} L(y|\theta) = -\mathbb{E}_{y \sim P_{data}} \log p(y|\theta)$$

- Recall that Fisher information is defined as

$$\mathcal{I}(\theta) = \mathbb{E}_{y \sim p(y|\theta)} (\nabla L(y|\theta) (\nabla L(y|\theta))^T) \quad (5)$$

- We can use Fisher information to adapt the learning rate according to the local curvature. (5) inspire us to use some average of $g(\theta^{(t)})(g(\theta^{(t)}))^T$

Adaptive Stochastic Gradient Descent

- Previously, we performed an update for all parameters using the same learning rate
- Duchi et al (2011) proposed an improved version of SGD, AdaGrad, that adapts the learning rate to the parameters, according to the frequencies of their associated features
- Denote the vector of parameters as θ and the gradient at iteration t as g_t . Let η be the usual learning rate for SGD. AdaGrad's update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t} + \epsilon} \odot g_t$$

where G_t is a diagonal matrix where each diagonal element is the sum of the squares of the corresponding gradients up to time step t

RMSprop

- A potential weakness about AdaGrad is its accumulation of the squared gradients in G_t , which in turn cause the learning rate to shrink and eventually become very small
- RMSprop (Geoff Hinton): resolve AdaGrad's diminishing learning rate via the exponentially decaying average

$$\mathbb{E}(g^2)_t = 0.9\mathbb{E}(g^2)_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\mathbb{E}(g^2)_t} + \epsilon} g_t$$

Adam

- Presumably the most popular stochastic gradient methods in machine learning, proposed by D.P. Kingma et al (2014).
- In addition to the squared gradients, Adam also keeps an exponentially decaying average of the past gradients

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \quad v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

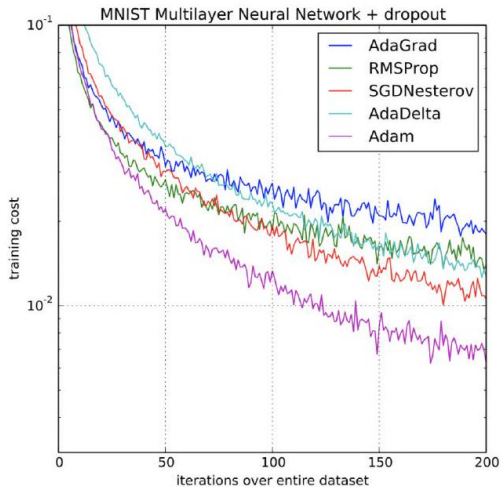
- Bias correction for zero initialization

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

- Adam uses the same update rule

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

Test on MNIST Images



Pros and Cons for Adaptive Methods

Pros

- Faster training speed and smoother learning curve
- Easier to choose hyperparameters
- Better when data are very sparse

Cons

- Worse performance on unseen data (Wilson et al., 2017)
- Convergence issue: non-decreasing learning rates, extreme learning rates

Some recent proposals for improvement: AMSGrad (Reddi et al., 2018), AdaBound (Luo et al., 2019), AdaBelief (Zhuang et al., 2020), etc.

Outline

- 1 Online Machine Learning
- 2 Basic Streaming Algorithms
- 3 Online Learning in Statistics
- 4 Recursive Algorithms
- 5 Online Updating Estimations
- 6 Renewable Estimation and Incremental Inference
- 7 Parallel-and-Stream Accelerator
- 8 Online Nonparametric Estimation
- 9 Advanced Gradient Descent
- 10 Deep Bootstrap Framework**

Deep Bootstrap Framework

Generalization theory

Goal: Understand when and why trained models have small test error

To study

Empirically measure

Analyze

$$\text{TestError}(f_t) = \text{TrainError}(f_t) + \underbrace{[\text{TestError}(f_t) - \text{TrainError}(f_t)]}_{\text{Generalization gap}}$$

Problem 1:

Modern methods can often achieve zero training error. The resulting equation becomes trivial

Problem 2:

Since our current bounds for this are all expressed in terms of *function capacity*, they turn vacuous when applied to *overparameterized* neural networks

Deep Bootstrap Framework

Deep Bootstrap Framework

Key idea: Consider alternative test error decomposition

$$\text{TestError}(f_t) = \underbrace{\text{TestError}(f_t^{\text{iid}})}_{\text{A: Online Learning}} + \underbrace{[\text{TestError}(f_t) - \text{TestError}(f_t^{\text{iid}})]}_{\text{B: Bootstrap error}}$$

Deep Bootstrap Framework

$$\text{TestError}(f_t) = \underbrace{\text{TestError}(f_t^{\text{iid}})}_{\text{A: Online Learning}} + \underbrace{[\text{TestError}(f_t) - \text{TestError}(f_t^{\text{iid}})]}_{\text{B: Bootstrap error}}$$

Given some data generating distribution D and a learning algorithm f
(a fixed architecture + optimizer + hyperparameters)

f_t Model after training for t steps on a dataset size of n
(**Real World**: Training with reused/bootstrapped data)

f_t^{iid} Model after training for t steps on a dataset size of ∞
(**Ideal World**: Training always with fresh data)

Deep Bootstrap Framework

$$\text{TestError}(f_t) = \underbrace{\text{TestError}(f_t^{\text{iid}})}_{\text{A: Online Learning}} + \underbrace{[\text{TestError}(f_t) - \text{TestError}(f_t^{\text{iid}})]}_{\text{B: Bootstrap error}}$$

Online learning: What is the test error after training for t steps on completely fresh data?

How fast does the model optimize the population loss in the ideal world?

Deep Bootstrap Framework

$$\text{TestError}(f_t) = \underbrace{\text{TestError}(f_t^{\text{iid}})}_{\text{A: Online Learning}} + \underbrace{[\text{TestError}(f_t) - \text{TestError}(f_t^{\text{iid}})]}_{\text{B: Bootstrap error}}$$

Online learning: What is the test error after training for t steps on completely fresh data?

How fast does the model optimize the population loss in the ideal world?

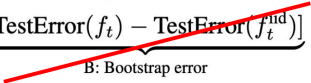
Bootstrap Error: How much does the real world differ from the ideal world?

What effect does reusing data instead of having access to unlimited fresh data have on test error?

Deep Bootstrap Framework

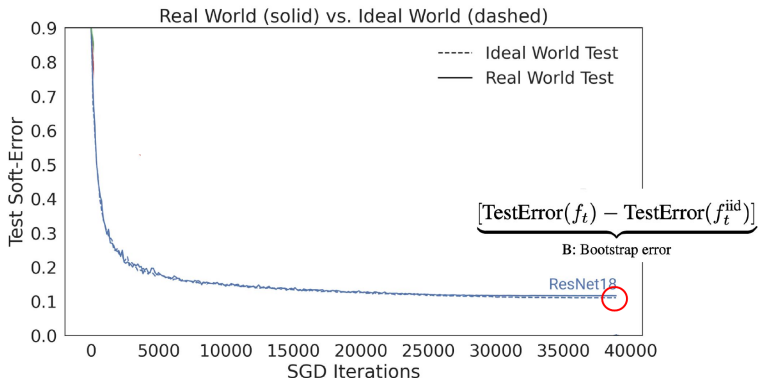
Main result of the paper

In typical deep learning setups, the bootstrap error is seen to be very small!

$$\text{TestError}(f_t) = \underbrace{\text{TestError}(f_t^{\text{iid}})}_{\text{A: Online Learning}} + \underbrace{[\text{TestError}(f_t) - \text{TestError}(f_t^{\text{iid}})]}_{\text{B: Bootstrap error}}$$


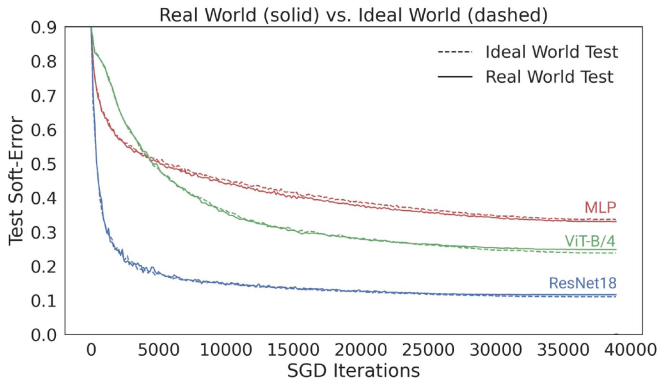
Deep Bootstrap Framework

What small bootstrap error looks like...



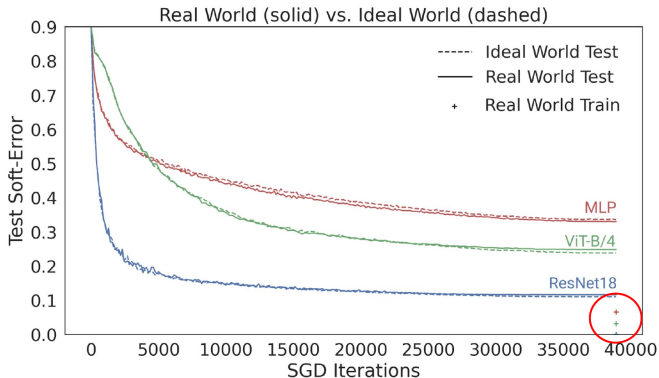
Deep Bootstrap Framework

For different architectures as well...



Deep Bootstrap Framework

Even when overfitting ... !



Deep Bootstrap Framework

What does this imply

$$\text{TestError}(f_t) = \underbrace{\text{TestError}(f_t^{\text{iid}})}_{\text{A: Online Learning}} + \underbrace{[\text{TestError}(f_t) - \text{TestError}(f_t^{\text{iid}})]}_{\text{B: Bootstrap error}}$$

To try to understand performance in

- Offline
- Real world, finite-data
- Overparameterized

settings ...

Deep Bootstrap Framework

What does this imply

$$\text{TestError}(f_t) = \underbrace{\text{TestError}(f_t^{\text{iid}})}_{\text{A: Online Learning}} + \underbrace{[\text{TestError}(f_t) - \text{TestError}(f_t^{\text{iid}})]}_{\text{B: Bootstrap error}}$$

To try to understand performance in

- Offline
- Real world, finite-data
- Overparameterized

settings ...

We can study the

- Online
- ideal world, infinite-data
- Underparameterized

setting (borrowing literature from online learning community, etc.)

Deep Bootstrap Framework

What does this imply

$$\text{TestError}(f_t) = \underbrace{\text{TestError}(f_t^{\text{iid}})}_{\text{A: Online Learning}} + \underbrace{[\text{TestError}(f_t) - \text{TestError}(f_t^{\text{iid}})]}_{\text{B: Bootstrap error}}$$

Summary: Good online learners are good offline generalizers