Data-driven Algorithm Design

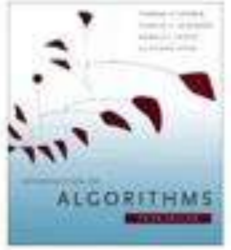
Maria-Florina (Nina) Balcan
Carnegie Mellon University

Analysis and Design of Algorithms

Classic algo design: solve a worst case instance.

- Easy domains, have optimal poly time algos.

  E.g., sorting, shortest paths



- Most domains are hard.

  E.g., clustering, partitioning, subset selection, auction design, …

Data driven algo design: use learning & data for algo design.

- Suited when repeatedly solve instances of the same algo problem.

Data Driven Algorithm Design

Data driven algo design: use learning & data for algo design.

- Different methods work better in different settings.
- Large family of methods – what's best in our application?

Prior work: largely empirical.

- Artificial Intelligence:

  [Horvitz-Ruan-Gomes-Kautz-Selman-Chickering, UAI 2001]

  [Xu-Hutter-Hoos-LeytonBrown, JAIR 2008]
- Computational Biology: E.g., [DeBlasio-Kececioglu, 2018]
- Game Theory: E.g., [Likhodedov and Sandholm, 2004]

Data Driven Algorithm Design

Data driven algo design: use learning & data for algo design.

- Different methods work better in different settings.
- Large family of methods – what's best in our application?

Prior work: largely empirical.

Our Work: Data driven algos with **formal guarantees**.

- Several cases studies of widely used algo families.
- General principles (for distributional & online learning): push boundaries of algorithm design and machine learning.

Related in spirit to Hyperparameter tuning, AutoML, MetaLearning.

Structure of the Talk

- Data driven algo design as batch learning.
    - A formal framework.
    - Case studies: clustering, partitioning pbs, auction pbs.
    - General sample complexity theorem.
- Data driven algo design as online learning.

Example: Clustering Problems

Clustering: Given a set objects organize then into natural groups.

- E.g., cluster news articles, or web pages, or search results by topic.

- Or, cluster customers according to purchase history.

- Or, cluster images by who is in them.

Often need do solve such problems repeatedly.

- E.g., clustering news articles (Google news).

Clustering Problems

Clustering: Given a set objects (news articles, customer surveys, web pages, …) organize then into natural groups.

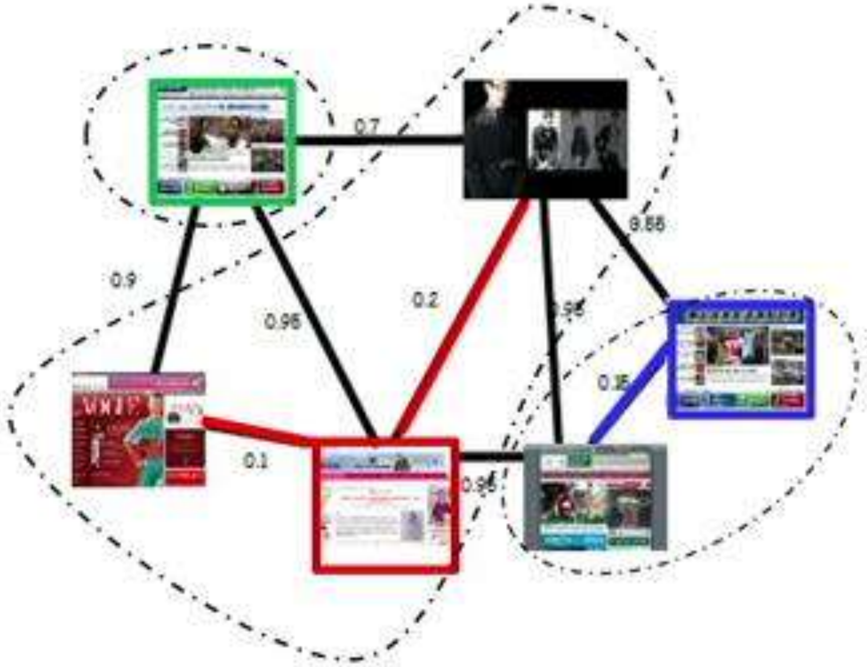
Objective based clustering

***k*-means**

Input: Set of objects S, d

Output: centers $\{c_1, c_2, \ldots, c_k\}$

To minimize $\sum_p \min_i d^2(p, c_i)$



Or minimize distance to ground-truth

Algorithm Design as Distributional Learning

Goal: *given large family of algos, sample of typical instances from domain, find an algo that performs well on new instances from same domain.*

[Gupta-Roughgarden, ITCS'16 &SICOMP'17]

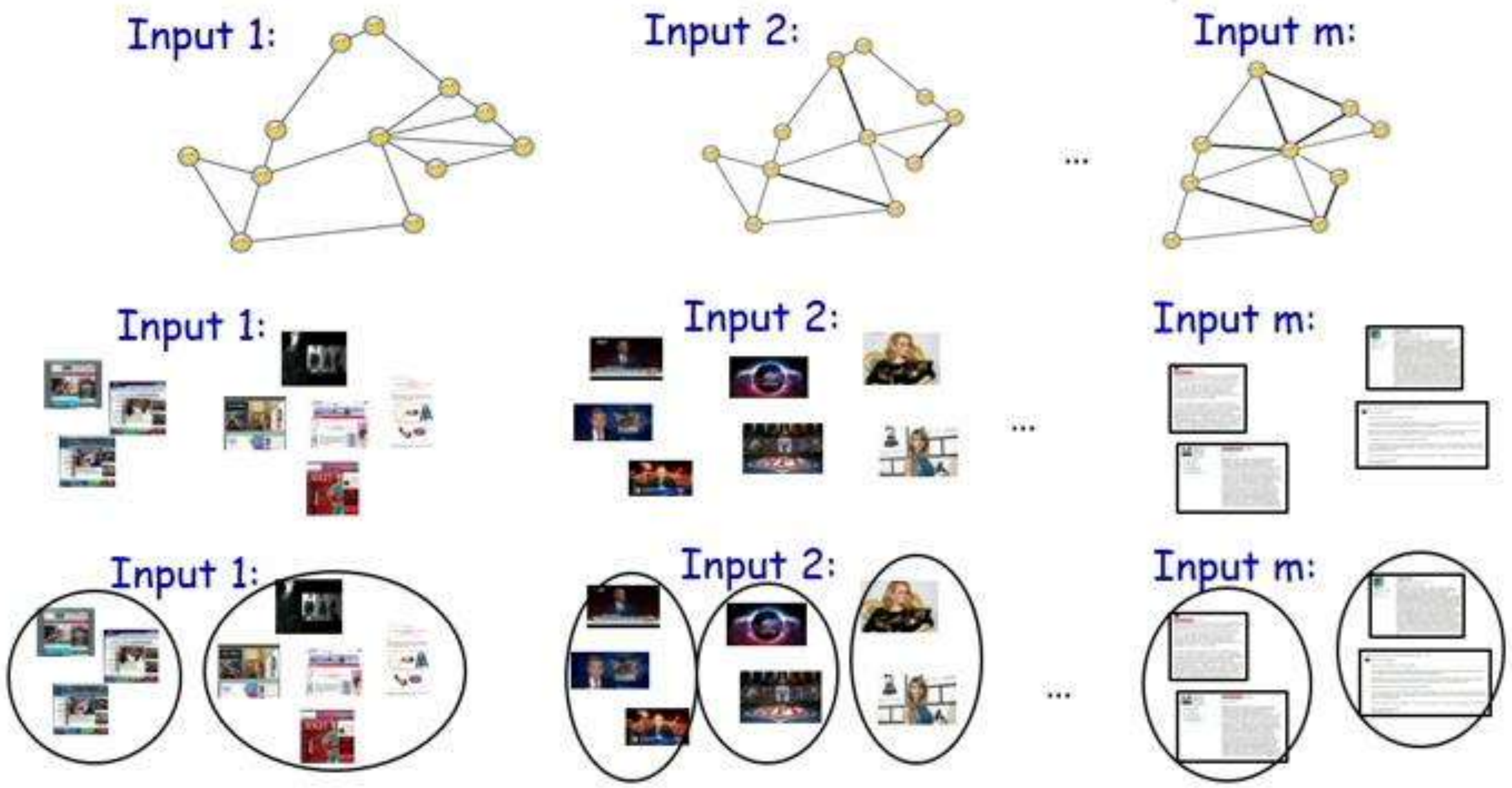
Large family F of algorithms

MST	+	Dynamic Programming
Greedy	+	Farthest Location
	...	

Sample of i.i.d. typical inputs

Facility location:

Clustering:

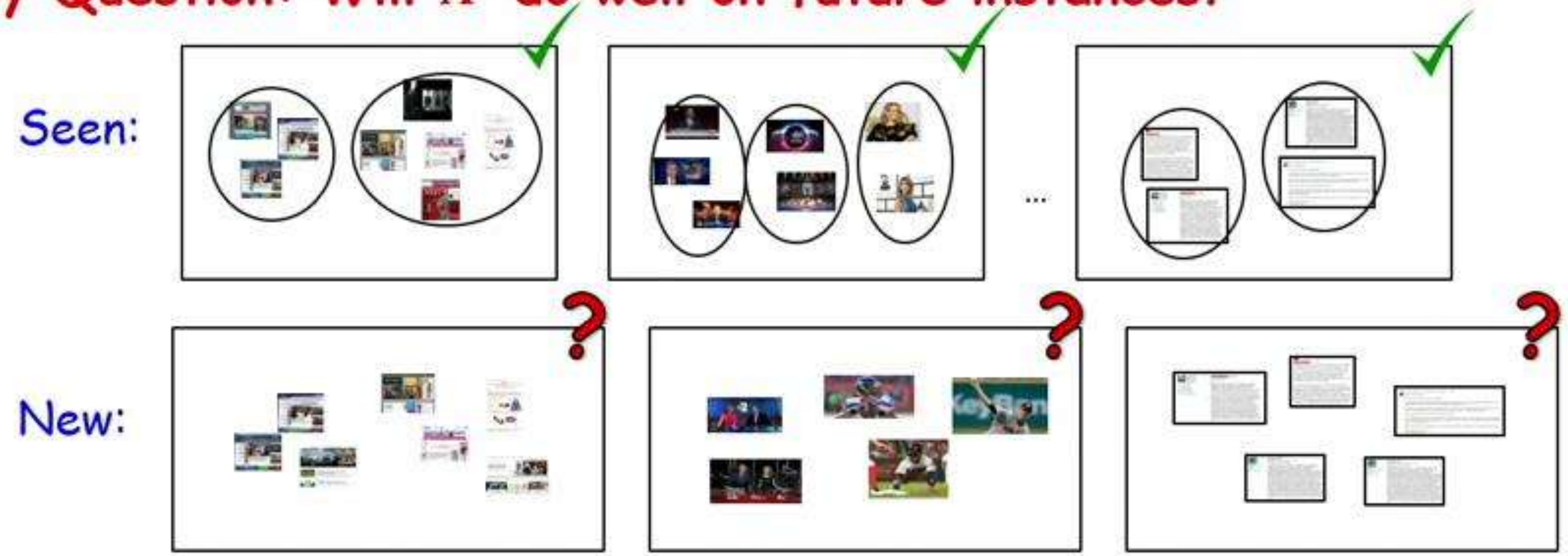


Sample Complexity of Algorithm Selection

Goal: given family of algos $\mathbf{F}$, sample of typical instances from domain (unknown distr. D), find algo that performs well on new instances from D.

Approach: **ERM**, find $\hat{\mathbf{A}}$ near optimal algorithm over the set of samples.

Key Question: Will $\hat{\mathbf{A}}$ do well on future instances?



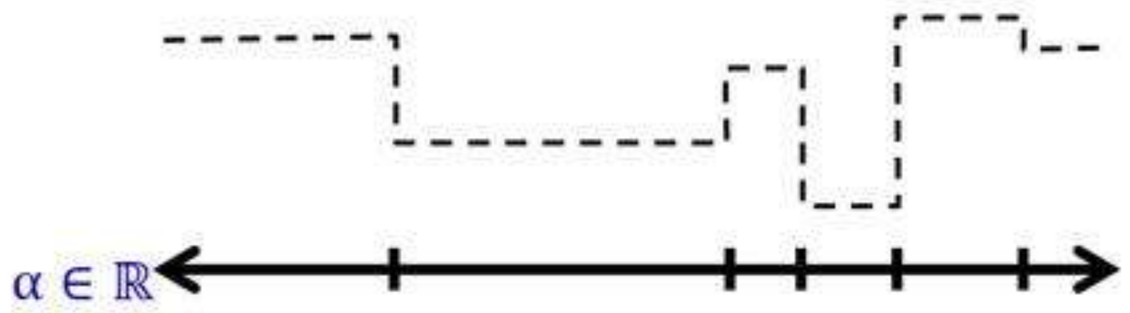
Sample Complexity: How large should our sample of typical instances be in order to guarantee good performance on new instances?

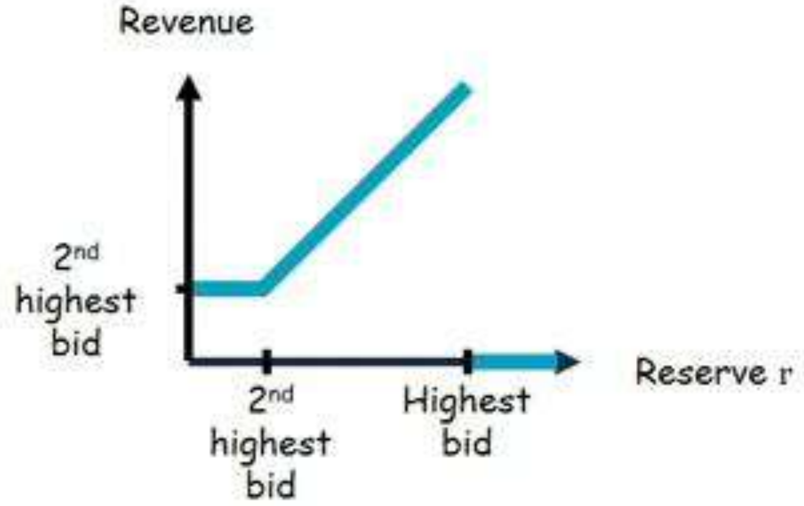
Statistical Learning Approach to AAD

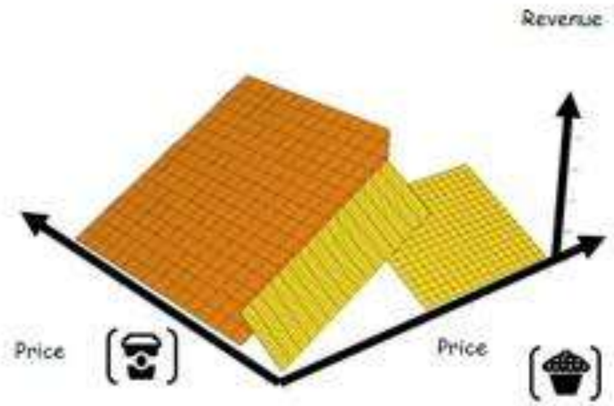
Sample Complexity: How large should our sample of typical instances be in order to guarantee good performance on new instances?

$m = O(\dim(F)/\epsilon^2)$ instances suffice to ensure generalizability

Challenge: "nearby" algos can have drastically different behavior.







Algorithm Design as Distributional Learning

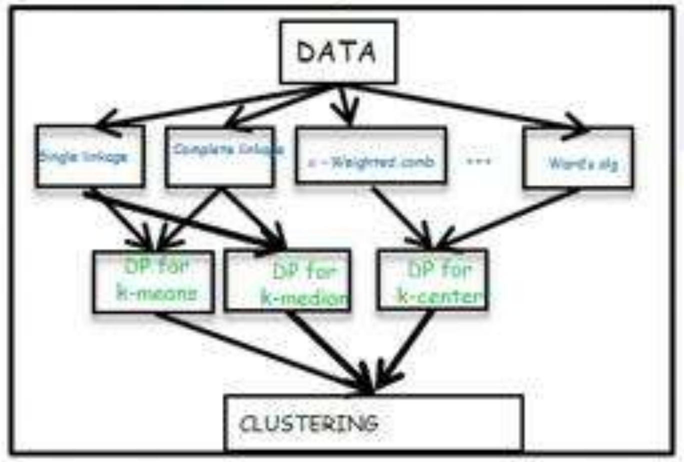
Prior Work: [Gupta-Roughgarden, ITCS'16 &SICOMP'17] proposed model; analyzed greedy algos for subset selection pbs (knapsack & independent set).

Our results: New algorithm classes for a wide range of problems.

Clustering: Parametrized Linkage

[Balcan-Nagarajan-Vitercik-White, COLT 2017]

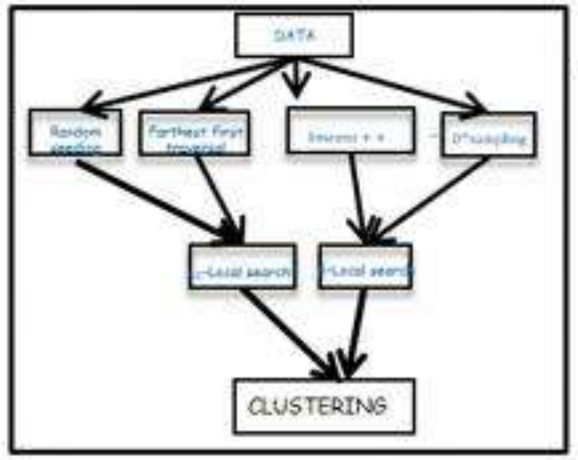
[Balcan-Dick-Lang, 2019]



$\dim(F) = O(\log n)$

Parametrized Lloyds

[Balcan-Dick-White, NeurIPS 2018]



$\dim(F) = O(k \log n)$

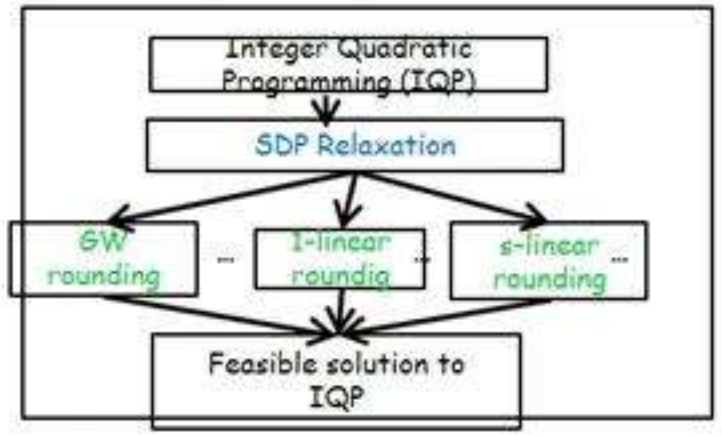
Alignment pbs (e.g., string alignment): parametrized dynamic prog.

[Balcan-DeBlasio-Dick-Kingsford-Sandholm-Vitercik, 2019]

Algorithm Design as Distributional Learning

Our results: New algorithm classes for a wide range of problems.

- Partitioning pbs via IQPs: SDP + Rounding

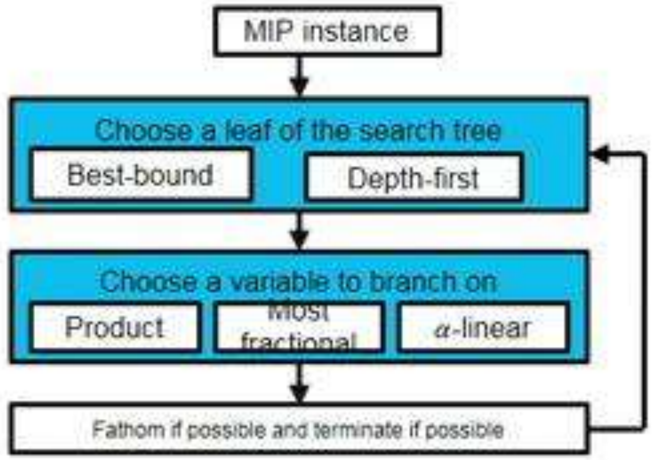


[Balcan-Nagarajan-Vitercik-White, COLT 2017]

E.g., Max-Cut,

$\dim(F) = O(\log n)$

Max-2SAT, Correlation Clustering



- MIPs: Branch and Bound Techniques

[Balcan-Dick-Sandholm-Vitercik, ICML'18]

$$
\begin{aligned}
&\text{Max } \boldsymbol{c} \cdot \boldsymbol{x} \\
&\text{s.t.} \quad A\boldsymbol{x} = \boldsymbol{b} \\
&\qquad\quad x_i \in \{0,1\}, \forall i \in I
\end{aligned}
$$

- Automated mechanism design for revenue maximization

Parametrized VCG auctions, posted prices, lotteries.

[Balcan-Sandholm-Vitercik, EC 2018]

Clustering: Linkage + Post-processing

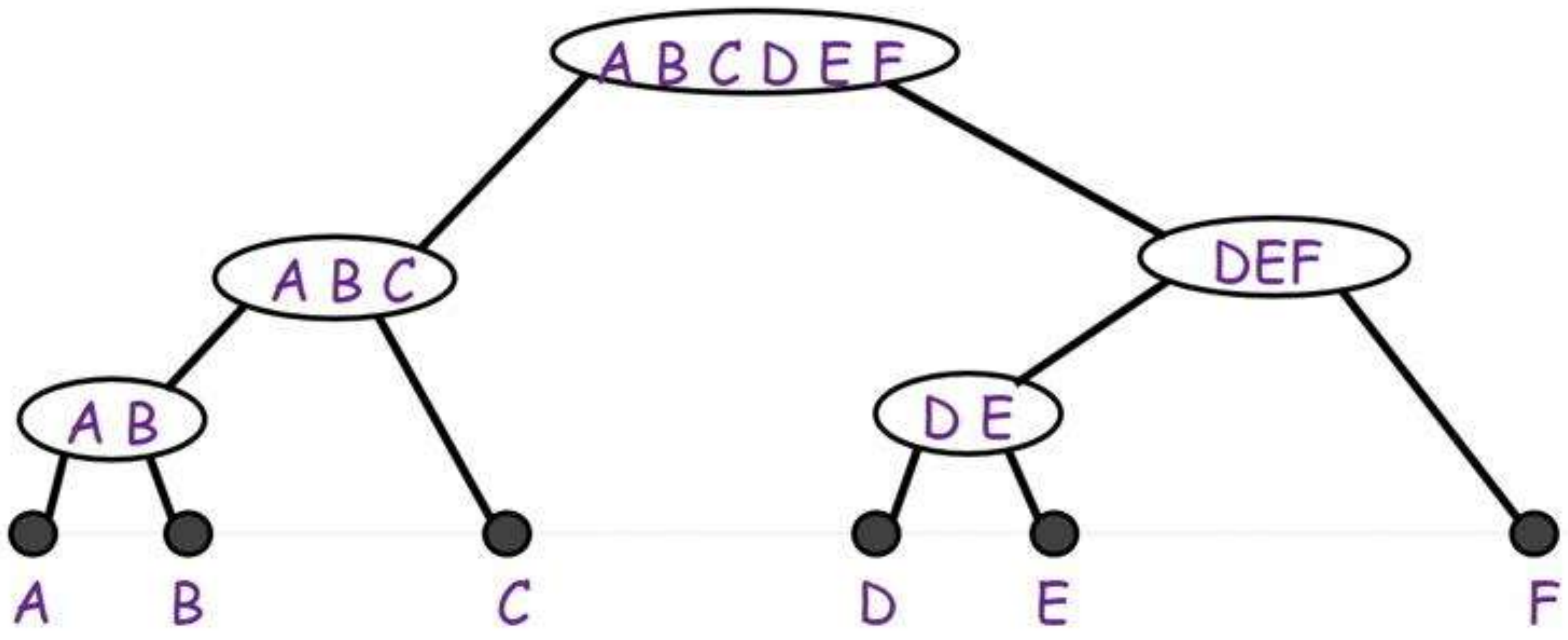
Family of poly time 2-stage algorithms:

1. Greedy linkage-based algo to get hierarchy (tree) of clusters.

Clustering: Linkage + Post-processing

Family of poly time 2-stage algorithms:

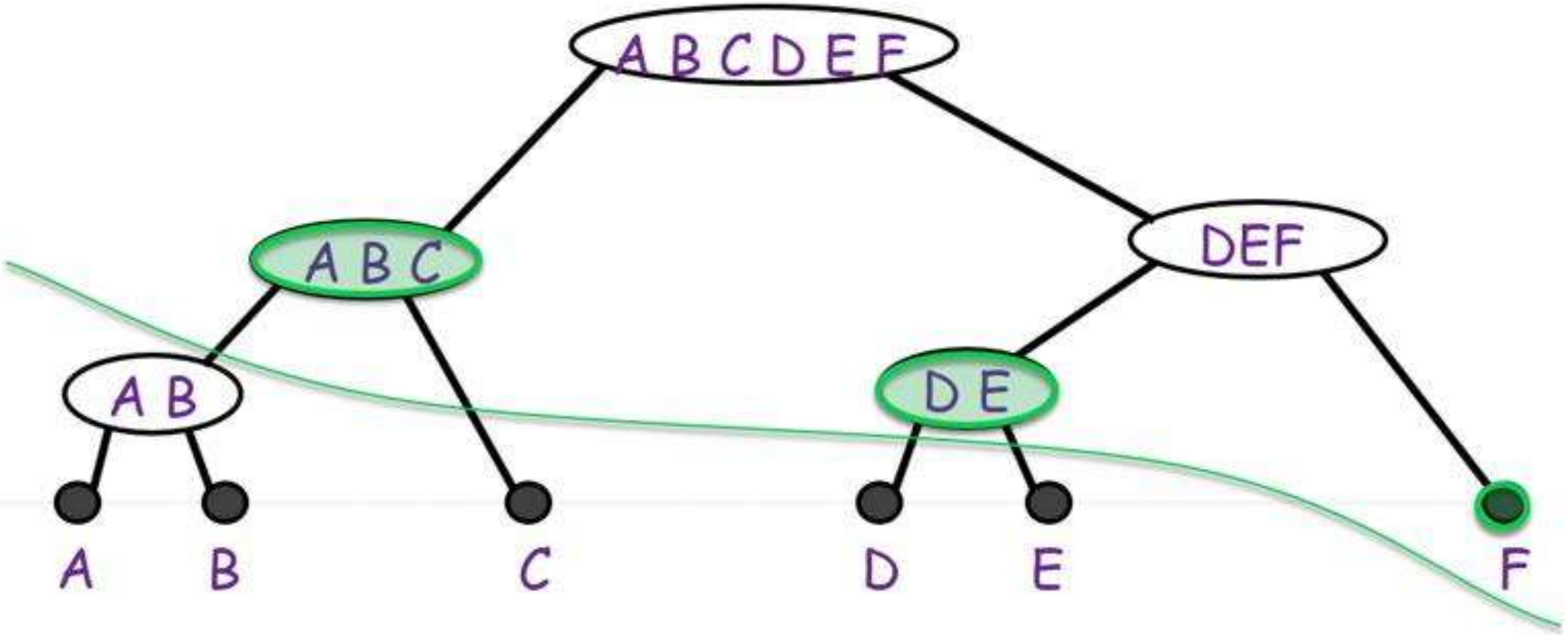
1. Greedy linkage-based algo to get hierarchy (tree) of clusters.
2. Fixed algo (e.g., DP or last k-merges) to select a good pruning.



Clustering: Linkage + Post-processing

Family of poly time 2-stage algorithms:

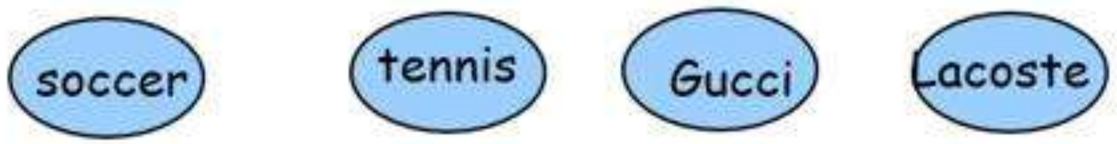
1. Greedy linkage-based algo to get hierarchy (tree) of clusters.
2. Fixed algo (e.g., DP or last k-merges) to select a good pruning.



Linkage Procedures for Hierarchical Clustering

Bottom-Up (agglomerative)

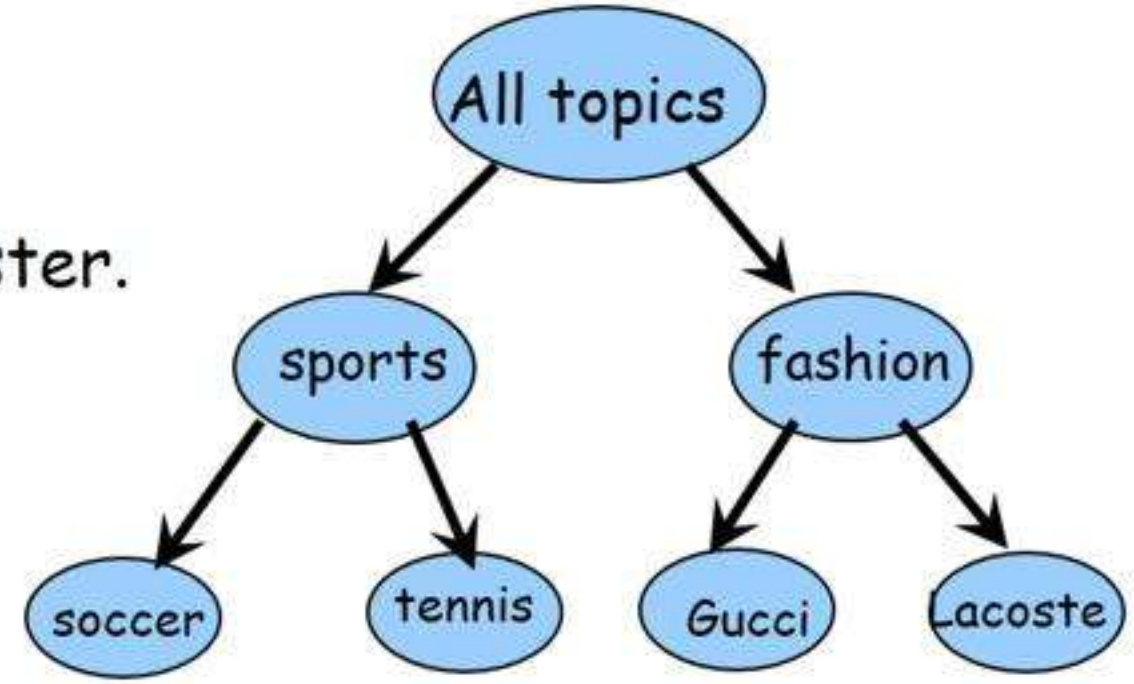
- Start with every point in its own cluster.
- Repeatedly merge the “closest” two clusters.



Linkage Procedures for Hierarchical Clustering

Bottom-Up (agglomerative)

- Start with every point in its own cluster.
- Repeatedly merge the "closest" two clusters.



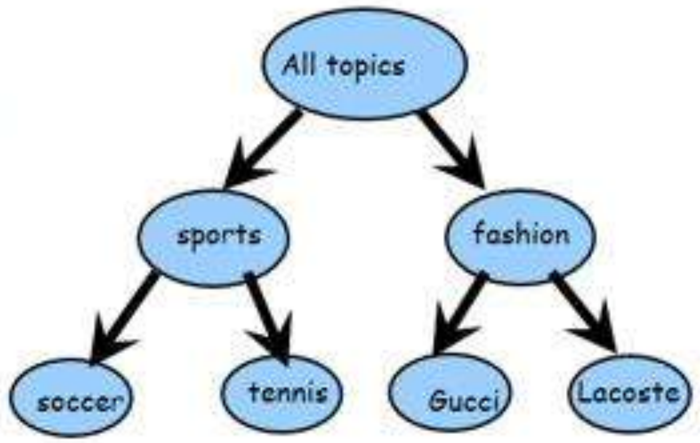
Different defs of "closest" give different algorithms.

Linkage Procedures for Hierarchical Clustering

Have a distance measure on pairs of objects.

$d(x,y)$ – distance between x and y

E.g., # keywords in common, edit distance, etc



- Single linkage: $\mathrm{dist}(A,B) = \min_{x \in A, x' \in B} \mathrm{dist}(x,x')$

- Complete linkage: $\mathrm{dist}(A,B) = \max_{x \in A, x' \in B} \mathrm{dist}(x,x')$

- Parametrized family, α-weighted linkage:

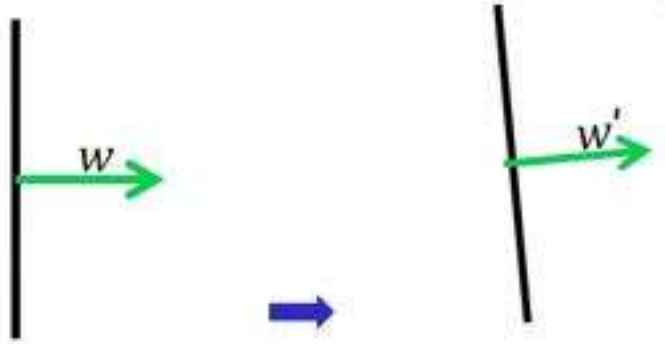
$$\mathrm{dist}_\alpha(A,B) = (1-\alpha) \min_{x \in A, x' \in B} d(x,x') + \alpha \max_{x \in A, x' \in B} d(x,x')$$

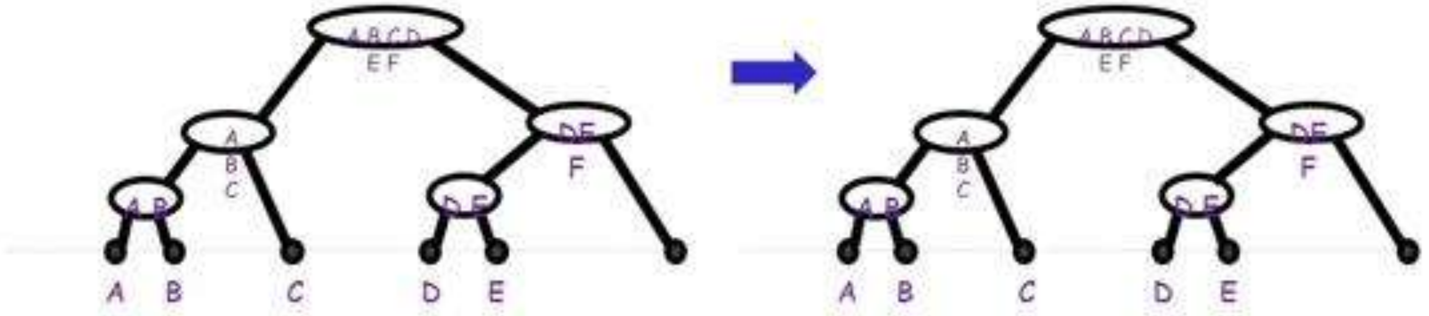
Clustering: Linkage + Dynamic Programming

Our Results: **α-weighted linkage** **+ Post-processing**

- Pseudo-dimension is O(log n), so small sample complexity.
- Given sample S, find best algo from this family in poly time.

Key Technical Challenge: small changes to the parameters of the algo can lead to radical changes in the tree or clustering produced.



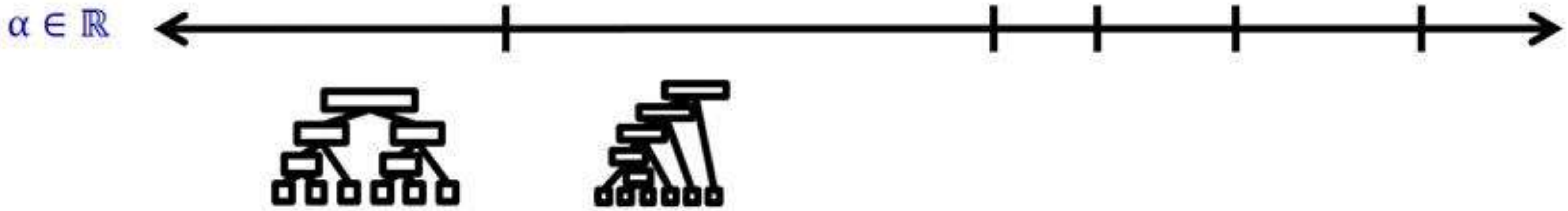


Problem: a single change to an early decision by the linkage algo, can snowball and produce large changes later on.

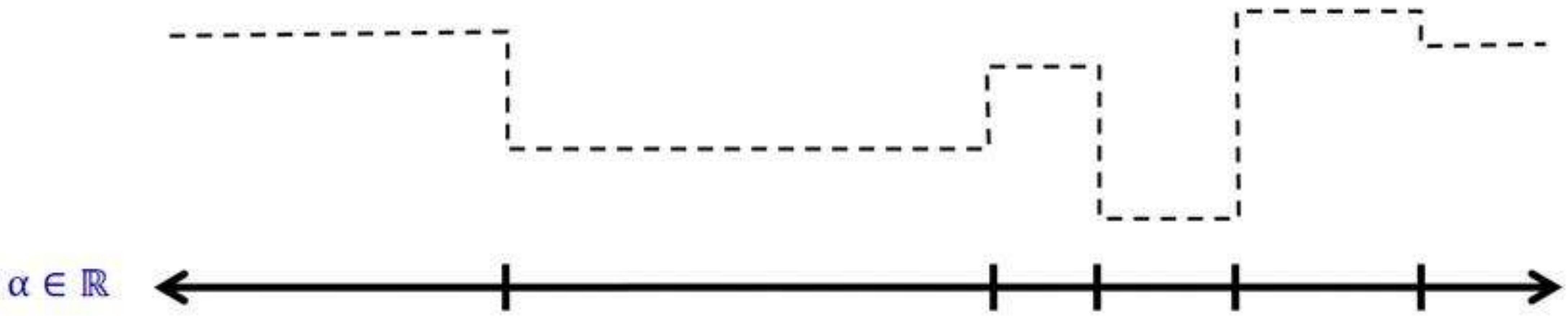
Clustering: Linkage + Dynamic Programming

Claim: Pseudo-dim of α-weighted linkage + Post-process is $O(\log n)$.

Key fact: If we fix a clustering instance of n pts and vary α, at most $O(n^8)$ switching points where behavior on that instance changes.



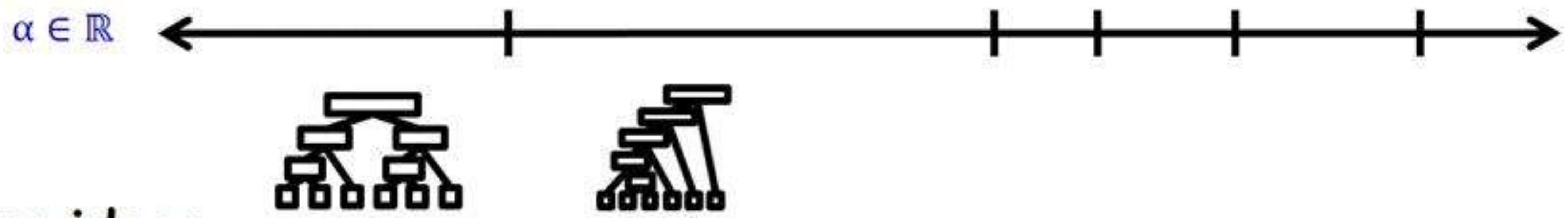
So, the cost function is piecewise-constant with at most $O(n^8)$ pieces.



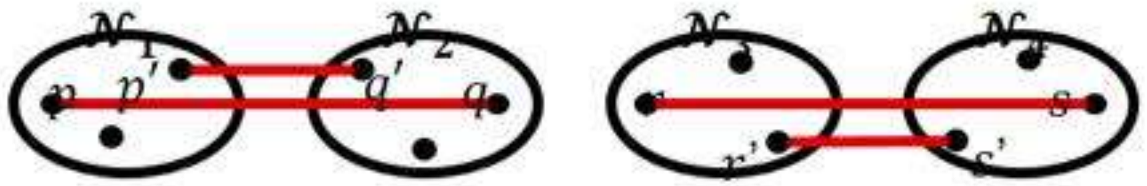
Clustering: Linkage + Dynamic Programming

Claim: Pseudo-dim of α-weighted linkage + Post-process is $O(\log n)$.

Key fact: If we fix a clustering instance of n pts and vary α, at most $O(n^8)$ switching points where behavior on that instance changes.



Key idea:

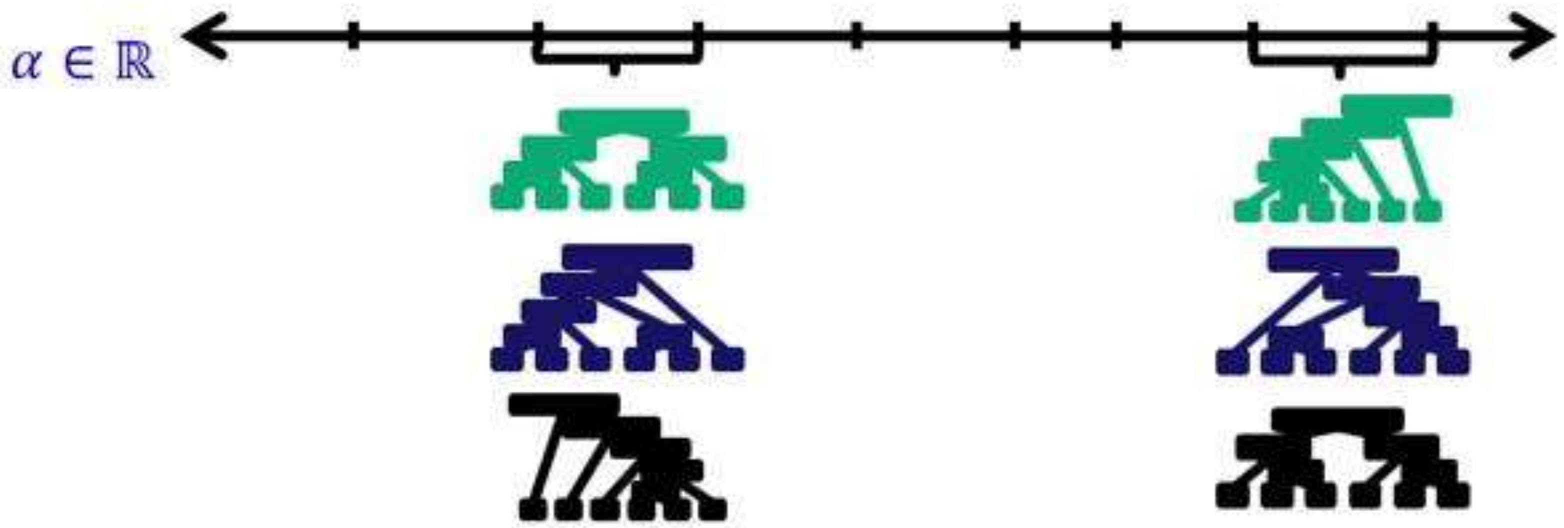


- For a given α, which will merge first, $\mathcal{N}_1$ and $\mathcal{N}_2$, or $\mathcal{N}_3$ and $\mathcal{N}_4$?
- Depends on which of $\alpha d(p,q) + (1-\alpha) d(p',q')$ or $\alpha d(r,s) + (1-\alpha) d(r',s')$ is smaller.
- An interval boundary an equality for 8 points, so $O(n^8)$ interval boundaries.

Clustering: Linkage + Dynamic Programming

Claim: Pseudo-dim of α-weighted linkage + Post-process is $O(\log n)$.

Key idea: For m clustering instances of n points, $O(mn^8)$ patterns.



- Pseudo-dim largest m for which 2^m patterns achievable.

- So, solve for $2^m \leq m\, n^8$. Pseudo-dimension is $O(\log n)$.

Clustering: Linkage + Dynamic Programming

Claim: Pseudo-dimension of α-weighted linkage + DP is $O(\log n)$, so small sample complexity.

For $N = O(\log n / \epsilon^2)$, w.h.p. expected performance cost of best α over the sample is ϵ-close to optimal over the distribution

Claim: Given sample S, can find best algo from this family in poly time.

Algorithm

- Solve for all α intervals over the sample

$\alpha \in \mathbb{R}$

- Find the α interval with the smallest empirical cost

Learning Both Distance and Linkage Criteria

[Balcan-Dick-Lang, 2019]

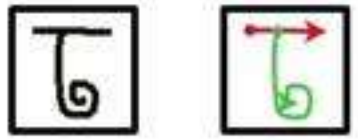
- Often different types of distance metrics.
  - Captioned images, d_0 image info, d_1 caption info.

"Black Cat"

"Bobcat"

  - Handwritten images: d_0 pixel info (CNN embeddings), d_1 stroke info.

Character Image Stroke Data



Family of Metrics: Given d_0 and d_1, define

$$d_\beta(x, x') = (1-\beta) \cdot d_0(x, x') + \beta \cdot d_1(x, x')$$

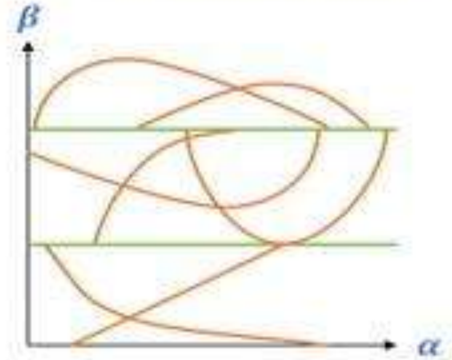
Parametrized (α, β)-weighted linkage (α interpolation between single and complete linkage and β interpolation between two metrics):

$$\mathrm{dist}_\alpha(A, B; d_\beta) = (1-\alpha) \min_{x \in A, x' \in B} d_\beta(x, x') + \alpha \max_{x \in A, x' \in B} d_\beta(x, x')$$

Learning Both Distance and Linkage Criteria

Claim: Pseudo-dim. of (α, β)-weighted linkage is $O(\log n)$.

Key fact: Fix instance of n pts; vary α, β, partition space with $O(n^8)$ linear, quadratic equations s.t. within each region, get same cluster tree.



Key Idea:

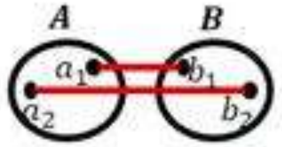
1. $O(n^4)$ linear sep. s.t. all β_1, β_2 in same region, d_{β_1} and d_{β_2} agree on order of distances between all n pts.

   Given β, decision whether $d_\beta(a, b)$ greater than $d_\beta(a', b')$ depends

   on which $(1-\beta)d_0(a, b) + \beta d_1(a, b)$ or $(1-\beta)d_0(a', b') + \beta d_1(a', b')$ is greater

2. Fix region, for sets A, B, all β agree on $a_1, b_1 = \underset{a\in A, b\in B}{\operatorname{argmin}} d_\beta(a, b)$, $a_2, b_2 = \underset{a\in A, b\in B}{\operatorname{argmax}} d_\beta(a, b)$.

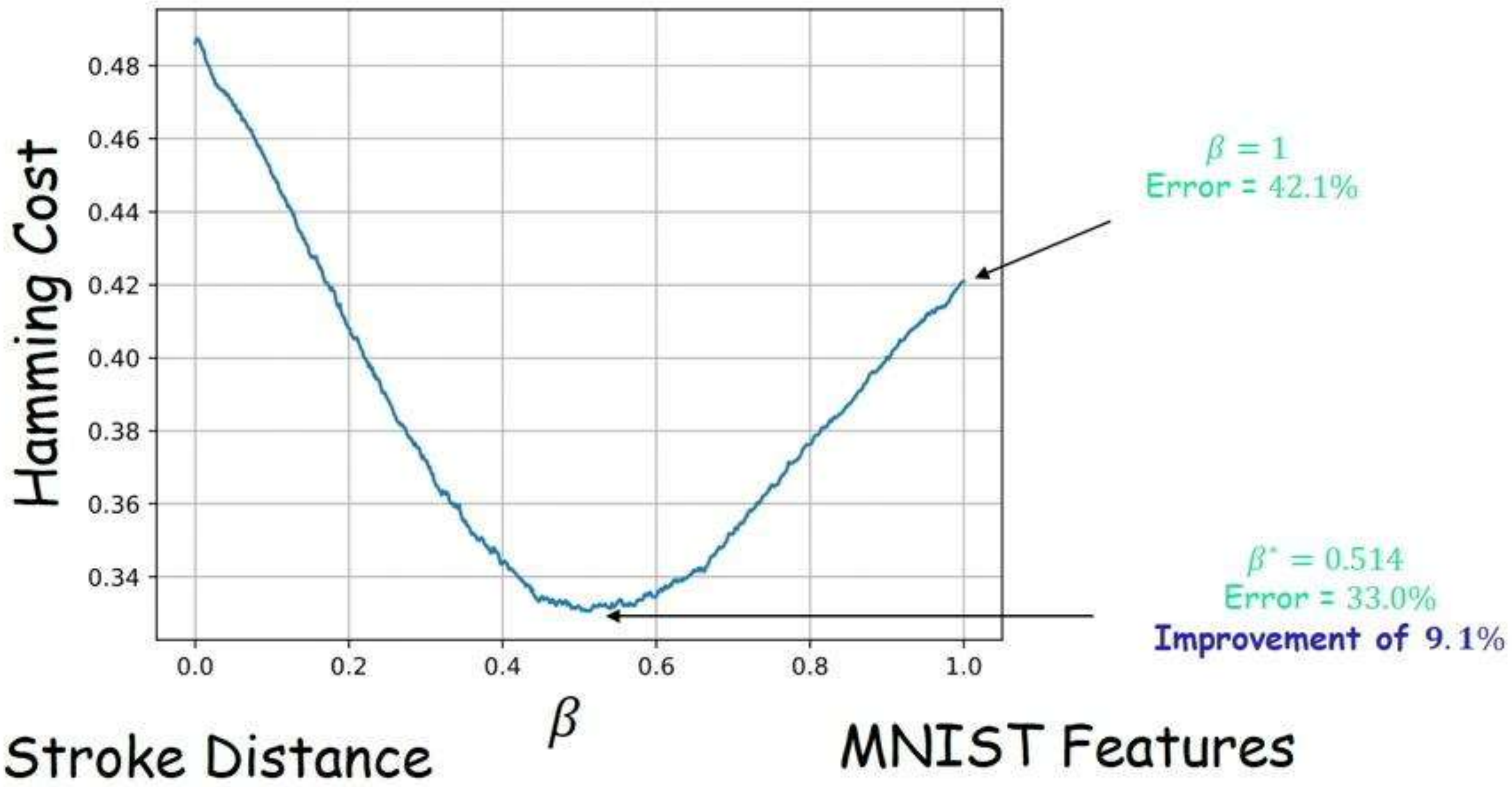
   So, $\mathrm{dist}_\alpha(A, B; d_\beta)$ is a quadratic fn of α, β:



$$\mathrm{dist}_\alpha(A, B; d_\beta) = (1-\alpha)[(1-\beta)d_0(a_1, b_1) + \beta d_1(a_1, b_1)] + \alpha[(1-\beta)d_0(a_2, b_2) + \beta d_1(a_2, b_2)]$$

Given α, decision to merge A, B or A', B' quadratic boundary, defined by 8 pts.

Clustering Subsets of Omniglot

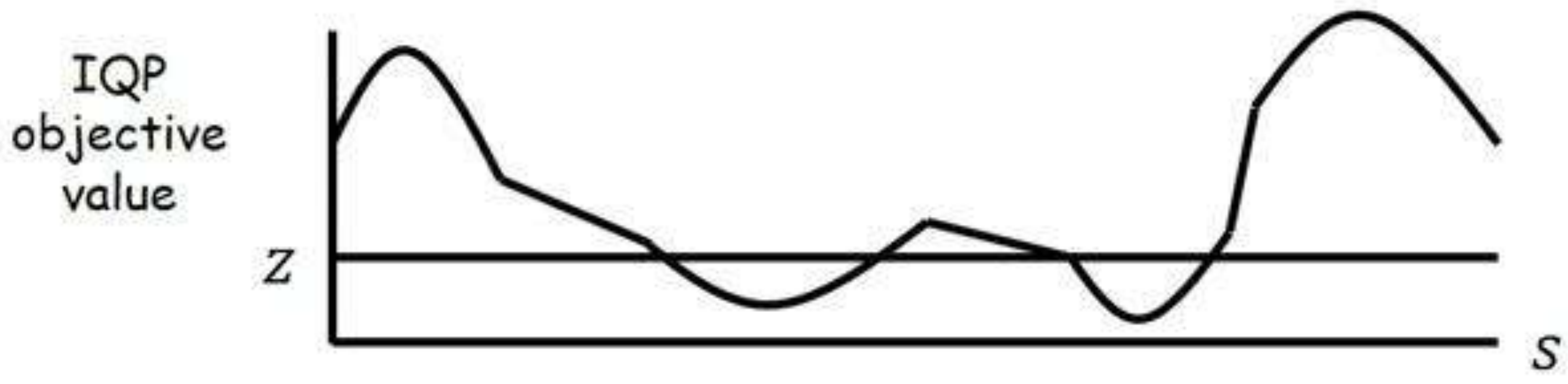


Partitioning Problems via IQPs

Our Results: SDP + s-linear rounding

Pseudo-dimension is O(log n), so small sample complexity.

Key idea: expected IQP objective value is piecewise quadratic in $\frac{1}{s}$ with n boundaries.



Data-driven Mechanism Design

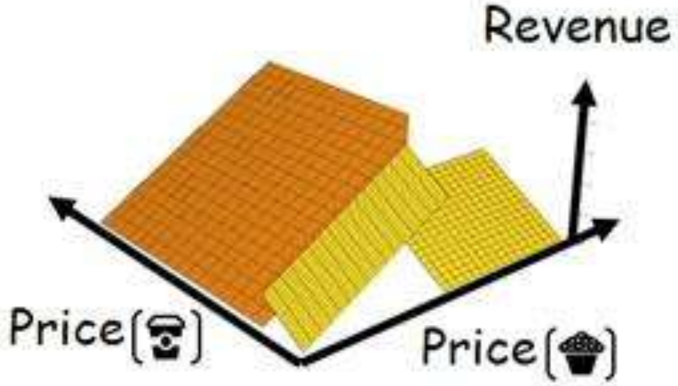
- **Similar ideas** for sample complex. of **data-driven mechanism design** for revenue maximization. [Balcan-Sandholm-Vitercik, EC'18]

- Pseudo-dim of $\{\text{revenue}_{\mathrm{M}} : \mathrm{M} \in \mathcal{M}\}$ for multi-item multi-buyer settings.
    - Many families: second-price auctions with reserves, posted pricing, two-part tariffs, parametrized VCG auctions, etc.

- Key insight: dual function sufficiently structured.
    - For a fixed set of bids, revenue is piecewise linear fnc of parameters.

<u>2nd-price auction with reserve</u>

<u>Posted price mechanisms</u>



General Sample Complexity via Dual Classes

[Balcan-DeBlasio-Kingsford-Dick-Sandholm-Vitercik, 2019]

- Want to prove that for all algorithm parameters $\boldsymbol{\alpha}$:

  $\frac{1}{|\mathcal{S}|}\sum_{\mathrm{I}\in\mathcal{S}} \mathrm{cost}_{\boldsymbol{\alpha}}(\mathrm{I})$ close to $\mathbb{E}[\mathrm{cost}_{\boldsymbol{\alpha}}(\mathbf{I})]$.

- Function class whose complexity want to control: $\{\mathrm{cost}_{\boldsymbol{\alpha}}: \text{parameter } \boldsymbol{\alpha}\}$.
- Proof takes advantage of structure of **dual class** $\{\mathrm{cost}_{\mathrm{I}}: \text{instances } \mathbf{I}\}$.

Theorem: Suppose for each $\mathrm{cost}_{\mathrm{I}}(\alpha)$ there are $\leq \mathrm{N}$ boundary fns $\mathrm{f}_1, \mathrm{f}_2, \ldots \in \mathrm{F}$ s. t within each region defined by them, $\exists\, \mathrm{g} \in \mathrm{G}$ s.t. $\mathrm{cost}_{\mathrm{I}}(\alpha) = \mathrm{g}(\alpha)$.

$$\mathrm{Pdim}(\{\mathrm{cost}_{\alpha}(\mathrm{I})\}) = O\big((\mathrm{d}_{\mathrm{F}^*} + \mathrm{d}_{\mathrm{G}^*}) \log(\mathrm{d}_{\mathrm{F}^*} + \mathrm{d}_{\mathrm{G}^*}) + \mathrm{d}_{\mathrm{F}^*} \log \mathrm{N}\big)$$

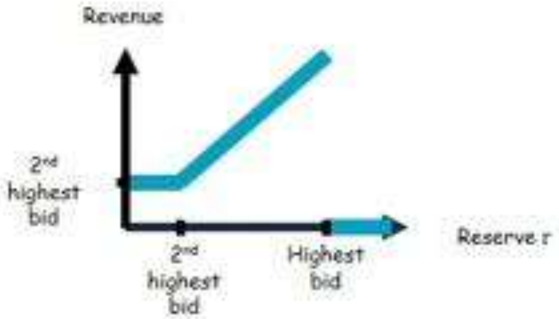
$\mathrm{d}_{\mathrm{F}^*}$ =VCdim of dual of F, $\mathrm{d}_{\mathrm{G}^*}$ =Pdim of dual of G.

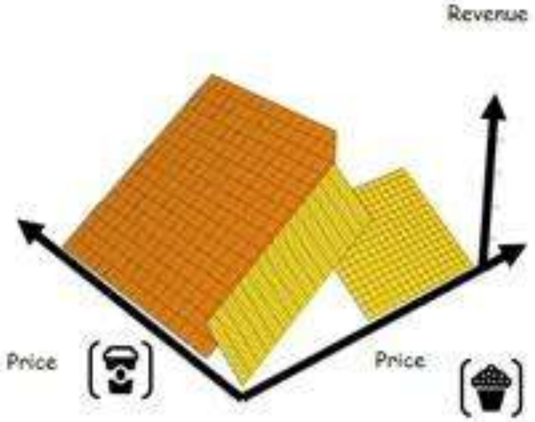
General Sample Complexity via Dual Classes

Theorem: Suppose for each $\text{cost}_I(\alpha)$ there are $\leq N$ boundary fns $f_1, f_2, \ldots \in F$ s. t within each region defined by them, $\exists\, g \in G$ s.t. $\text{cost}_I(\alpha) = g(\alpha)$.

$$\text{Pdim}(\{\text{cost}_\alpha(I)\}) = O\big((d_{F^*} + d_{G^*}) \log(d_{F^*} + d_{G^*}) + d_{F^*} \log N\big)$$

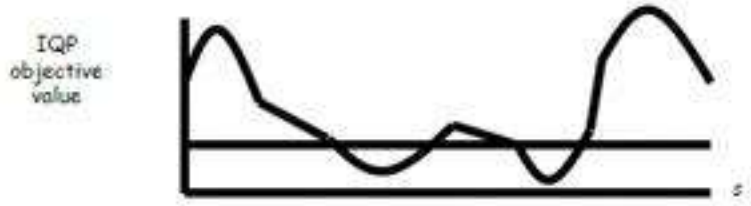
d_{F^*} =VCdim of dual of F, d_{G^*} =Pdim of dual of G.

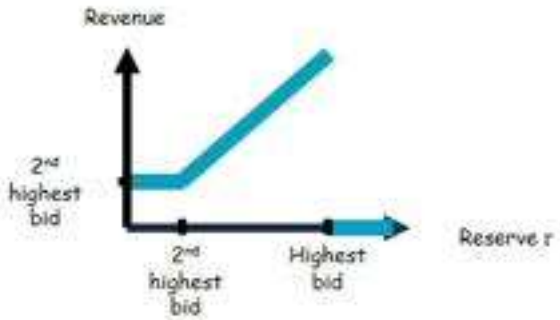


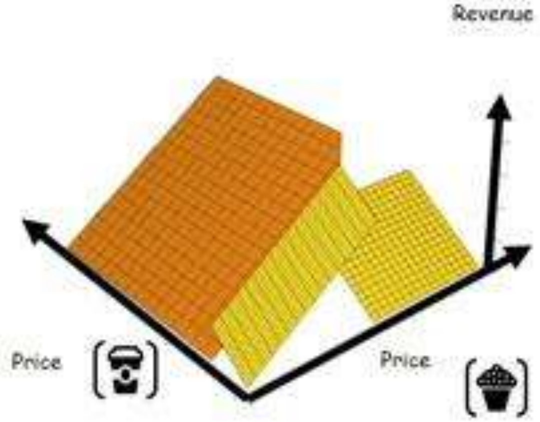


Online Algorithm Selection

- So far, batch setting: collection of typical instances given upfront.
- [Balcan-Dick-Vitercik, FOCS 2018], [Balcan-Dick-Pedgen, 2019] online alg. selection.
- **Challenge**: scoring fns non-convex, with lots of discontinuities.





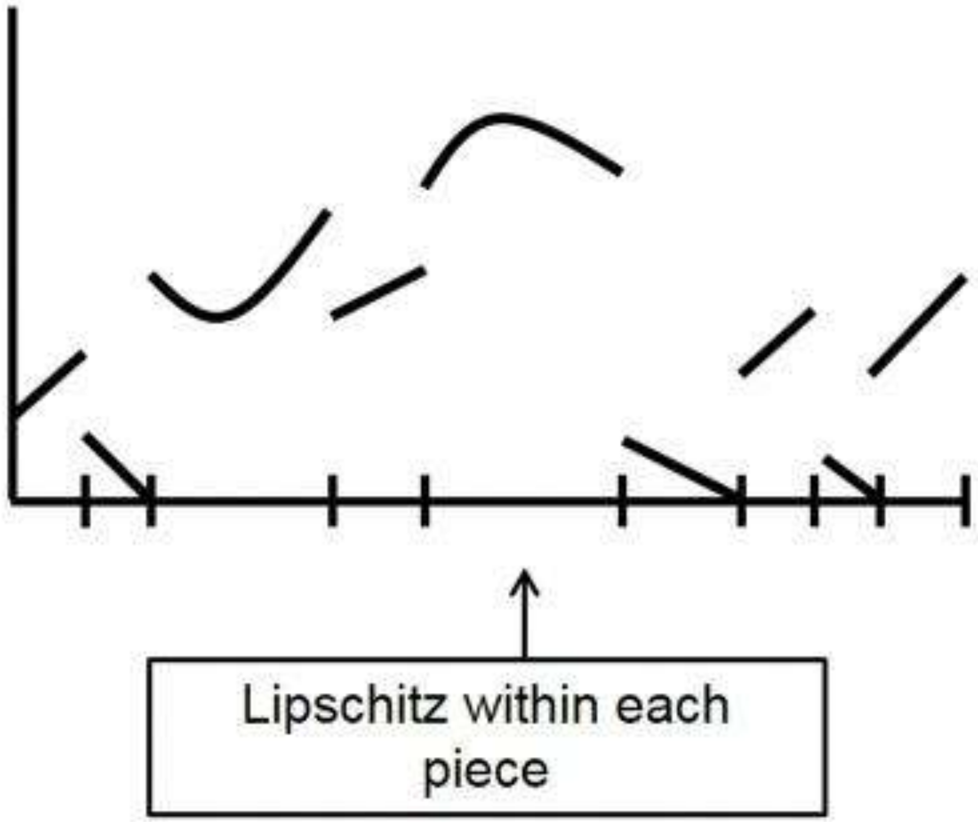


Cannot use known techniques.

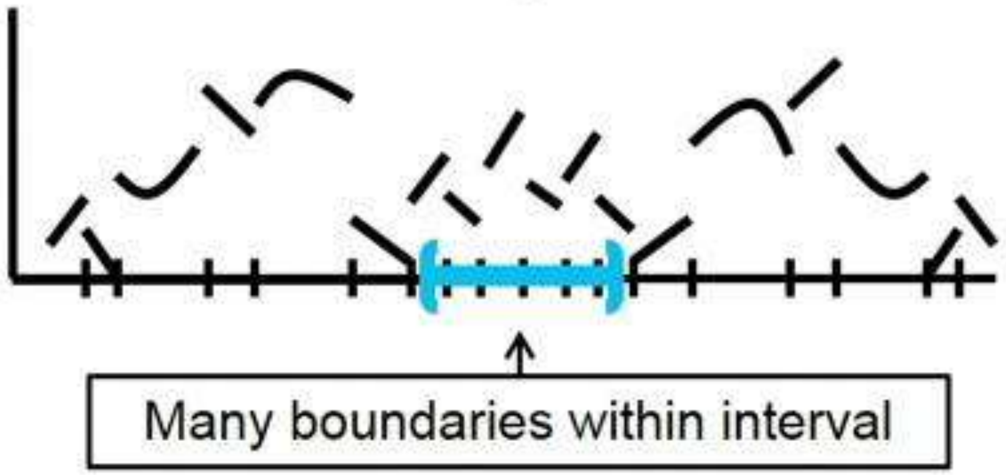
- Identify general properties (piecewise Lipschitz fns with dispersed discontinuities) sufficient for strong bounds.
  - Show these properties hold for many alg. selection pbs.

Dispersion, Sufficient Condition for No-Regret

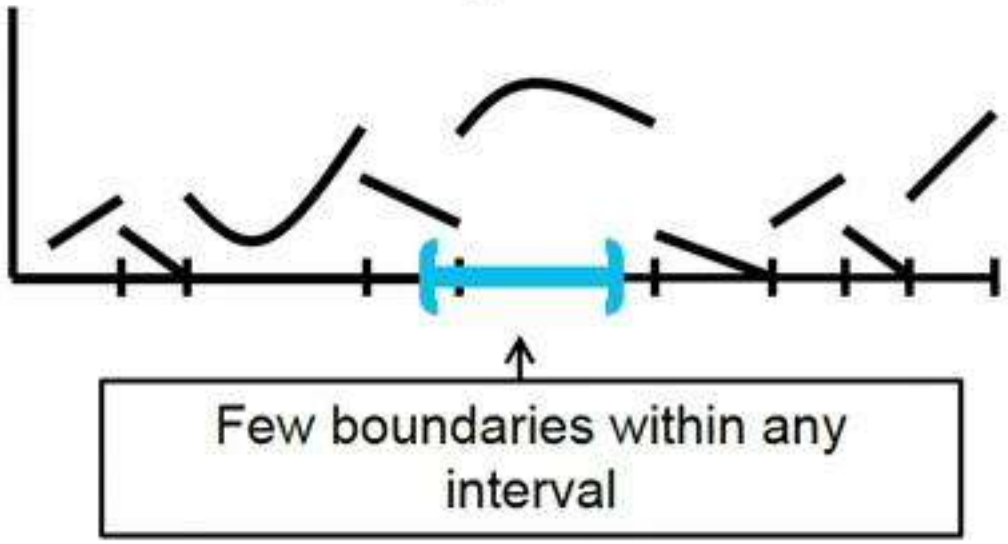
Piecewise Lipschitz function



Not disperse



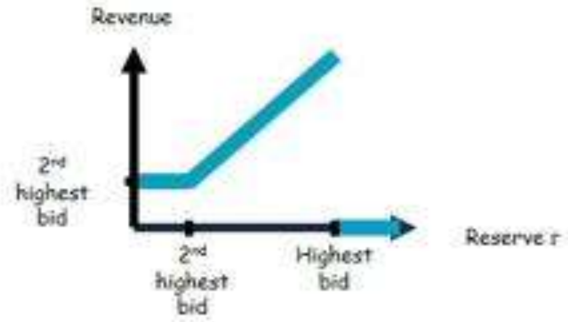
Disperse

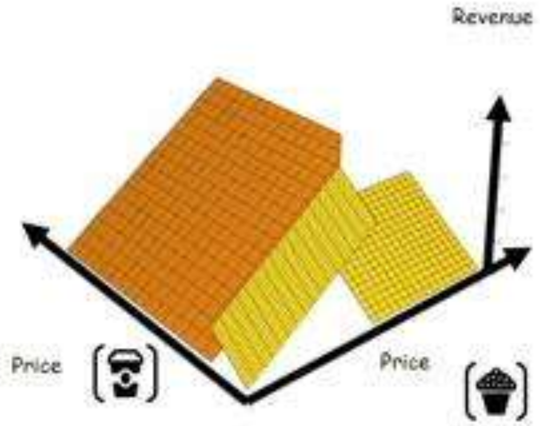


$\{u_1(\cdot), \ldots, u_T(\cdot)\}$ is $(\mathbf{w}, \mathbf{k})$-dispersed if any ball of radius $\mathbf{w}$ contains boundaries for at most $\mathbf{k}$ of the u_i.

Summary and Discussion

- Strong performance guarantees for data driven algorithm selection for combinatorial problems.
- Provide and exploit structural properties of dual class for good sample complexity and regret bounds.





- Learning theory: techniques of independent interest beyond algorithm selection.

Reducing ML Bias using Truncated Statistics

Constantinos Daskalakis
EECS and CSAIL, MIT

High-Level Goals

- ❑ **Selection bias in data collection**

  ⇒ **train distribution ≠ test distribution**

  ⇒ **prediction bias (a.k.a. "ML bias")**

- ❑ **Our Work:** decrease bias, by developing machine learning methods more robust to **censored and truncated samples**

High-Level Goals

❑ **Selection bias in data collection**

⇒ **train distribution ≠ test distribution**

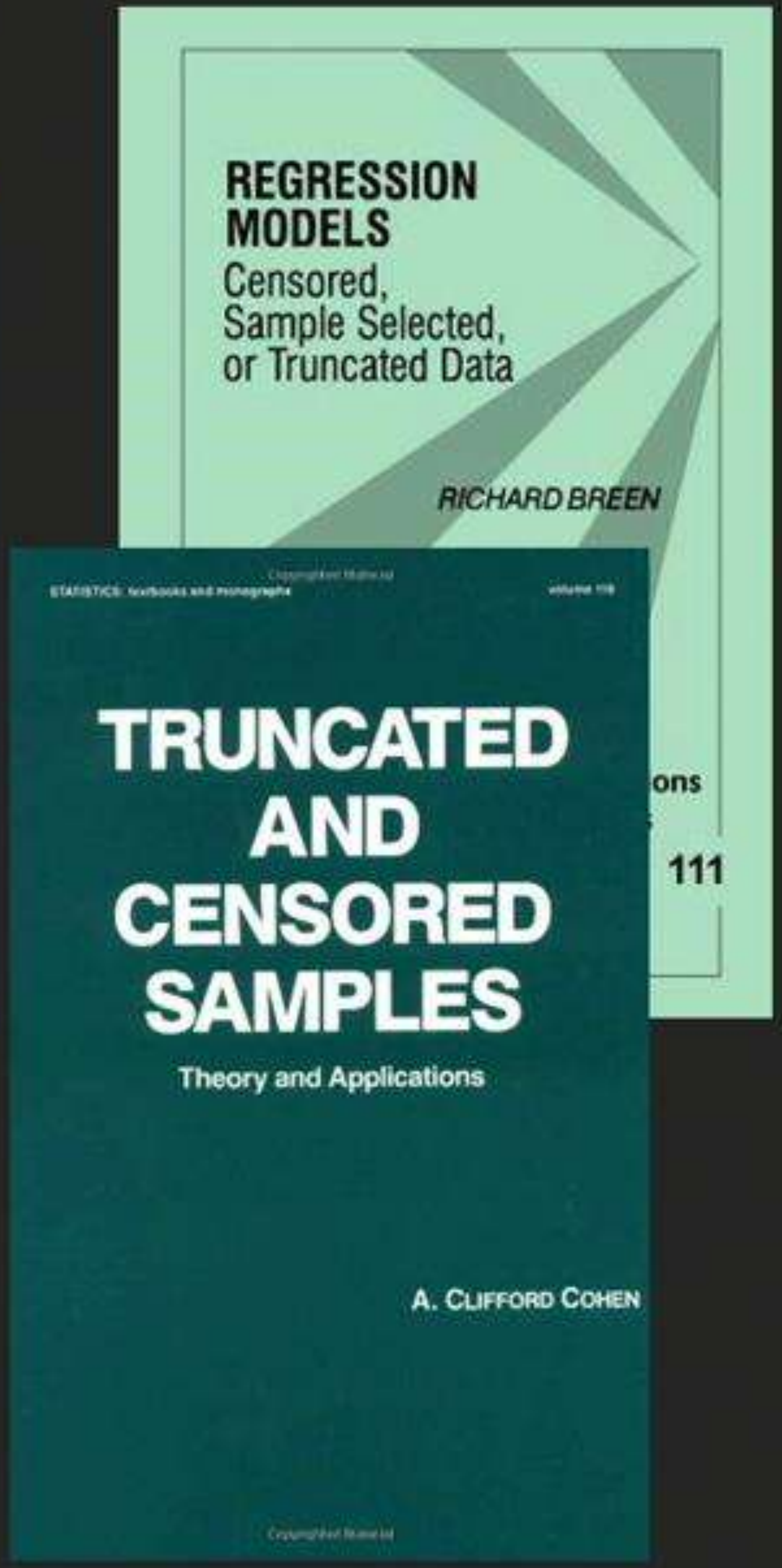
⇒ **prediction bias (a.k.a. "ML bias")**

❑ **Our Work:** decrease bias, by developing machine learning methods more robust to **censored and truncated samples**

Truncated Statistics: samples falling outside of observation "window" are hidden and their count is also hidden

Censored Statistics: ditto, but count of hidden data is provided

- limitations of measurement devices
- limitations of data collection
  - experimental design, ethical or privacy considerations,…



Motivating Example: IQ vs Income

Goal: Regress (IQ, Training, Education) vs Earnings [Wolfle&Smith'56, Hause'71,…]

Data Collection: survey families whose income is smaller than 1.5 times the poverty line; collect data $(x_i, y_i)_i$ where

- x_i: (IQ, Training, Education,…) of individual i
- y_i: earnings of individual i

Regression: fit some model $y = h_\theta(x) + \varepsilon$, e.g. $y = \theta^T x + \varepsilon$

Obvious Issue: **thresholding incomes may introduce bias**

It does, as pointed out by [Hausman-Wise, Econometrica'76] debunking prior results "showing" effects of education are strong, while of IQ and training are not

Motivating Example 2: Height vs Basketball

Goal: Regress **Height vs Basketball Performance**

Data Collection: use **NBA data** $(x_i, y_i)_i$ where

- x_i: height of individual i
- y_i: average number of points per game scored by individual i

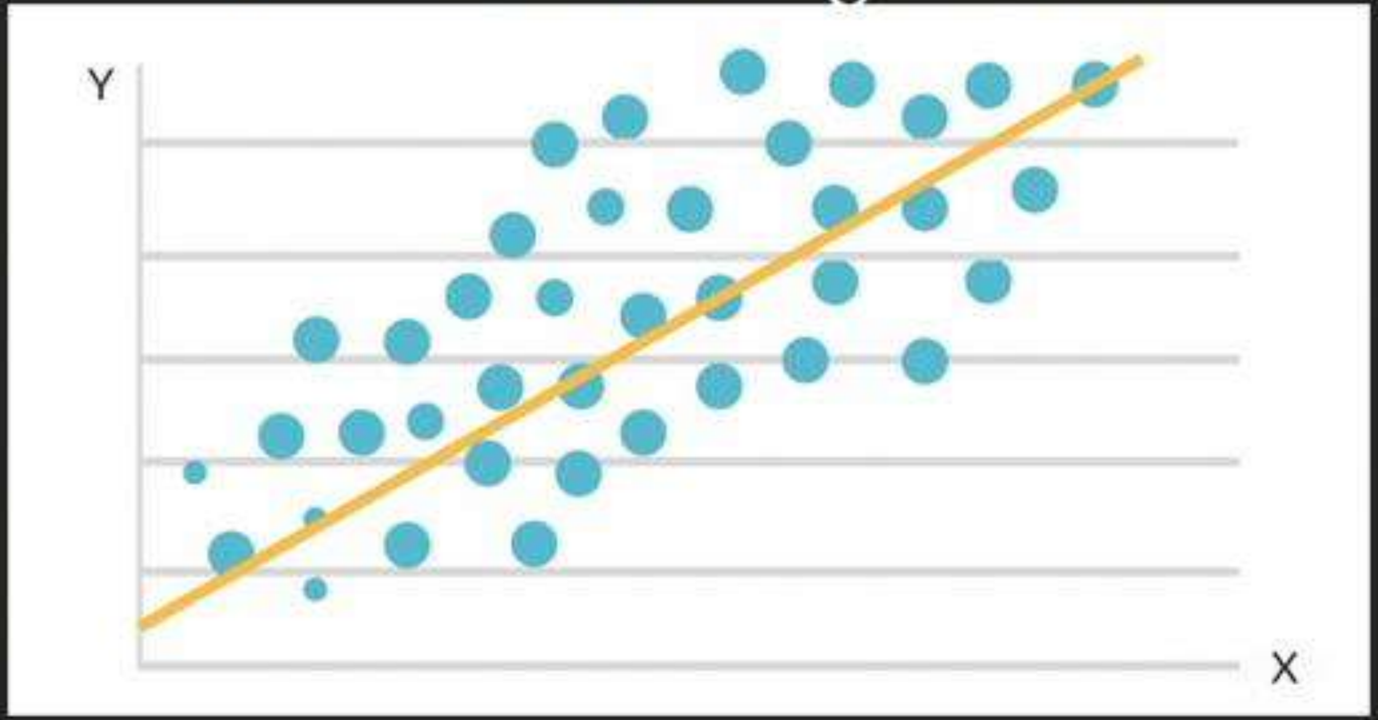
Regression: fit some model $y = h_\theta(x) + \varepsilon$

Obvious Issue: by using NBA data, we might infer that height is neutral or even negatively correlated with performance

What Happened?

Mental Picture:

Vanilla Linear Regression

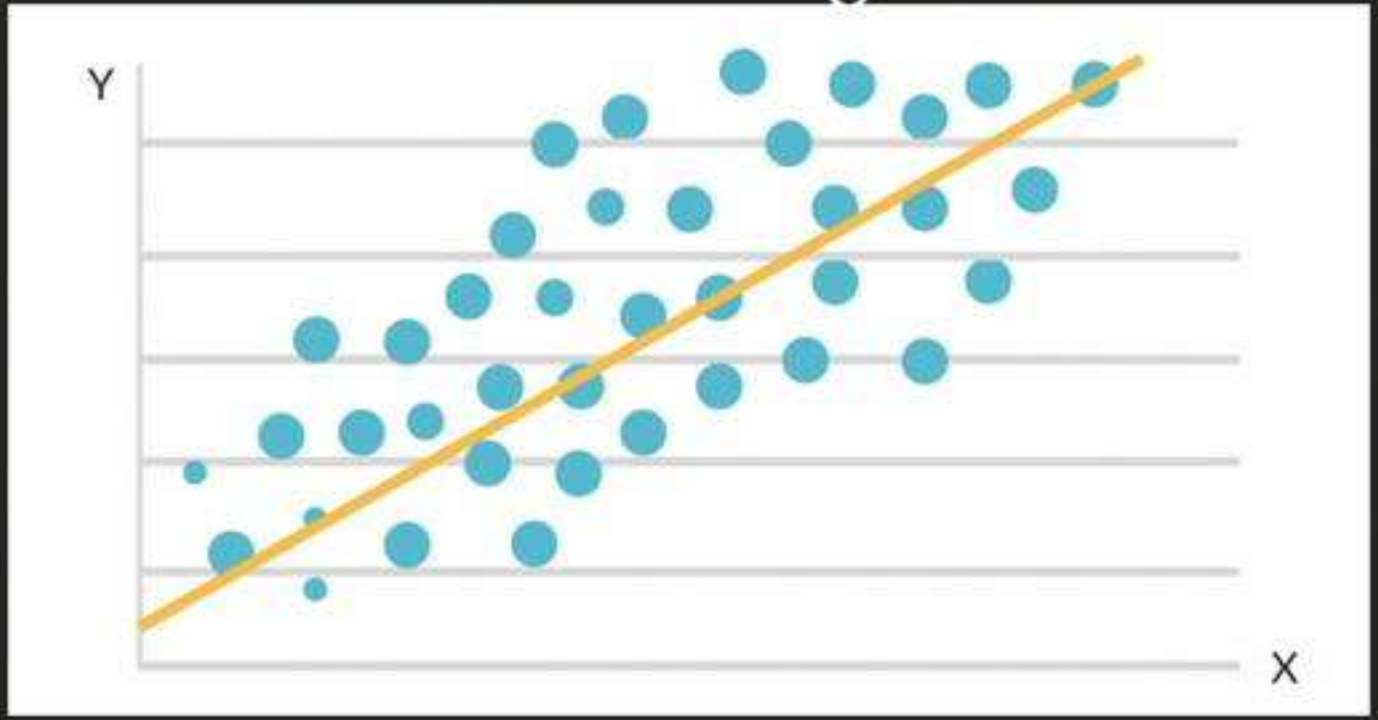


Truth: $y_i = \theta \cdot x_i + \varepsilon_i$, for all i

What Happened?

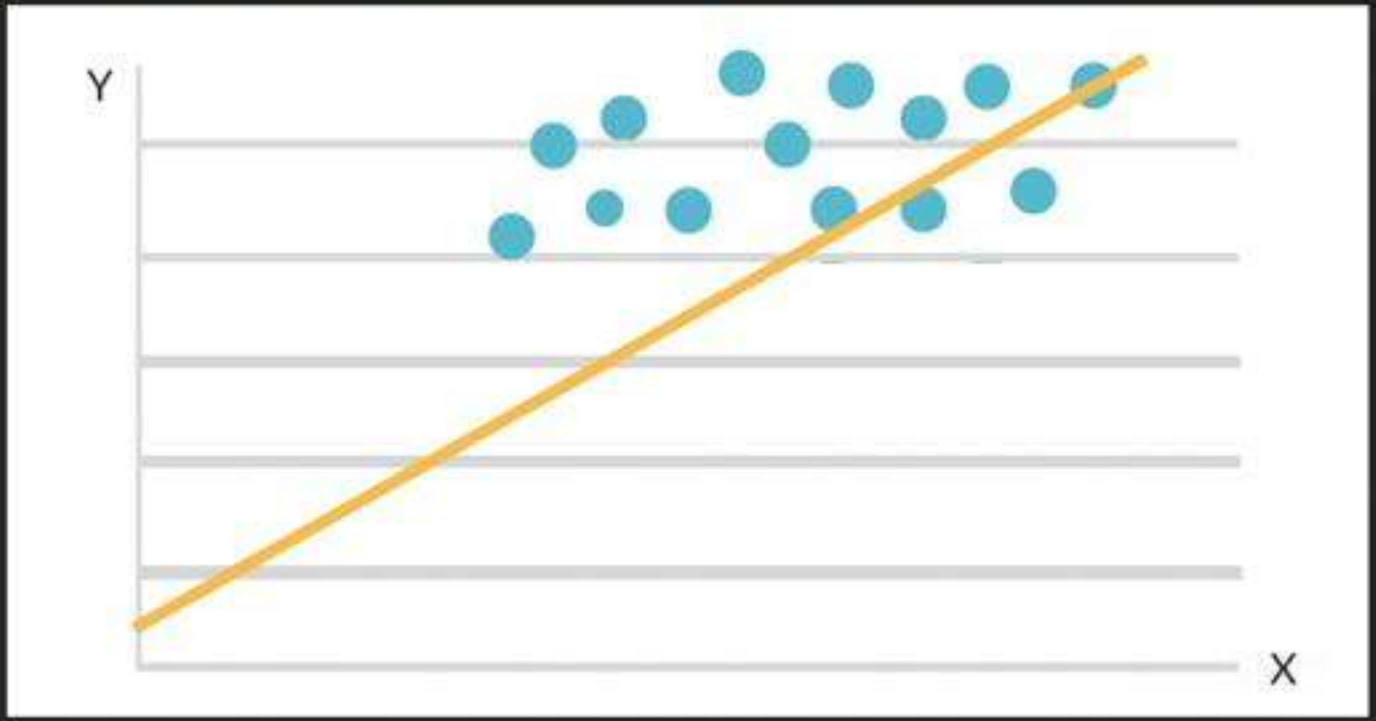
Mental Picture:

Vanilla Linear Regression



Truth: $y_i = \theta \cdot x_i + \varepsilon_i$, for all i

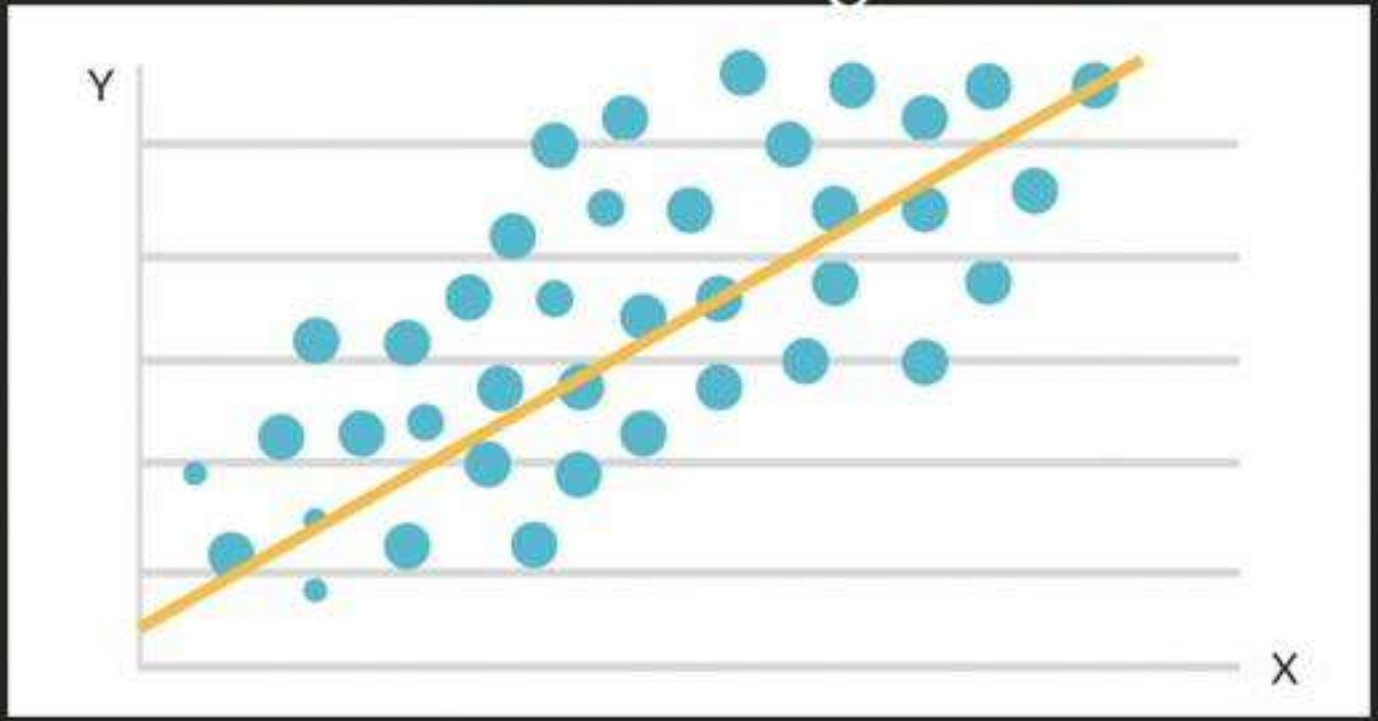
Data truncated on the Y-axis



What Happened?

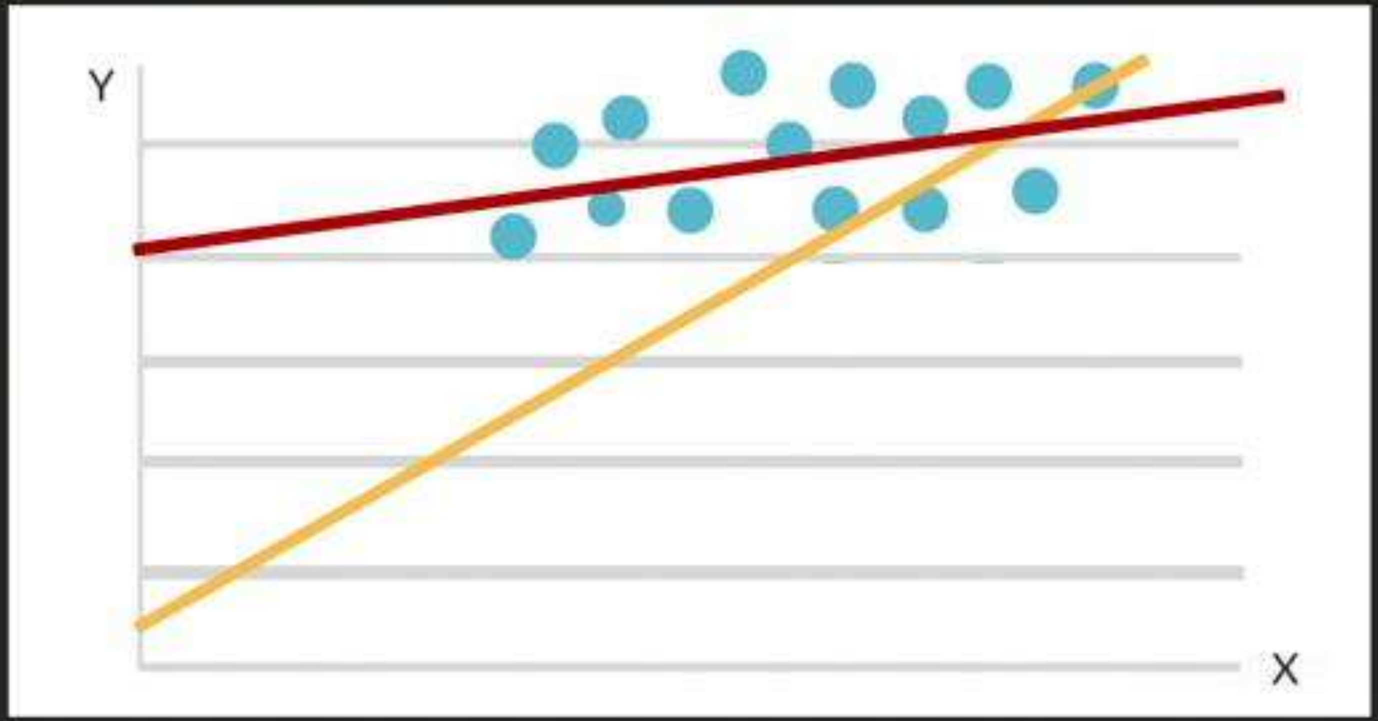
Mental Picture:

Vanilla Linear Regression



Truth: $y_i = \theta \cdot x_i + \varepsilon_i$, for all i

Data truncated on the Y-axis



Motivating Eg 3: Truncation on the X-axis

Gender Classifier	Darker Male	Darker Female	Lighter Male	Lighter Female	Largest Gap
Microsoft	94.0%	79.2%	100%	98.3%	20.8%
FACE++	99.3%	65.5%	99.2%	94.0%	33.8%
IBM	88.0%	65.3%	99.7%	92.9%	34.4%

Buolamwini, Gebru, FAT 2018

Explanation: Training data contains more faces that are of lighter skin tone, male gender, Caucasian

⇒ Training loss of gender classifier pays less attention to faces that are of darker skin tone, female gender, non-Caucasian

⇒ Test loss on faces that are of darker skin tone, female gender, non-Caucasian is worse

Classical example of bias in ML systems

Menu

- **Motivation**
- Flavor of Models, Techniques, Results

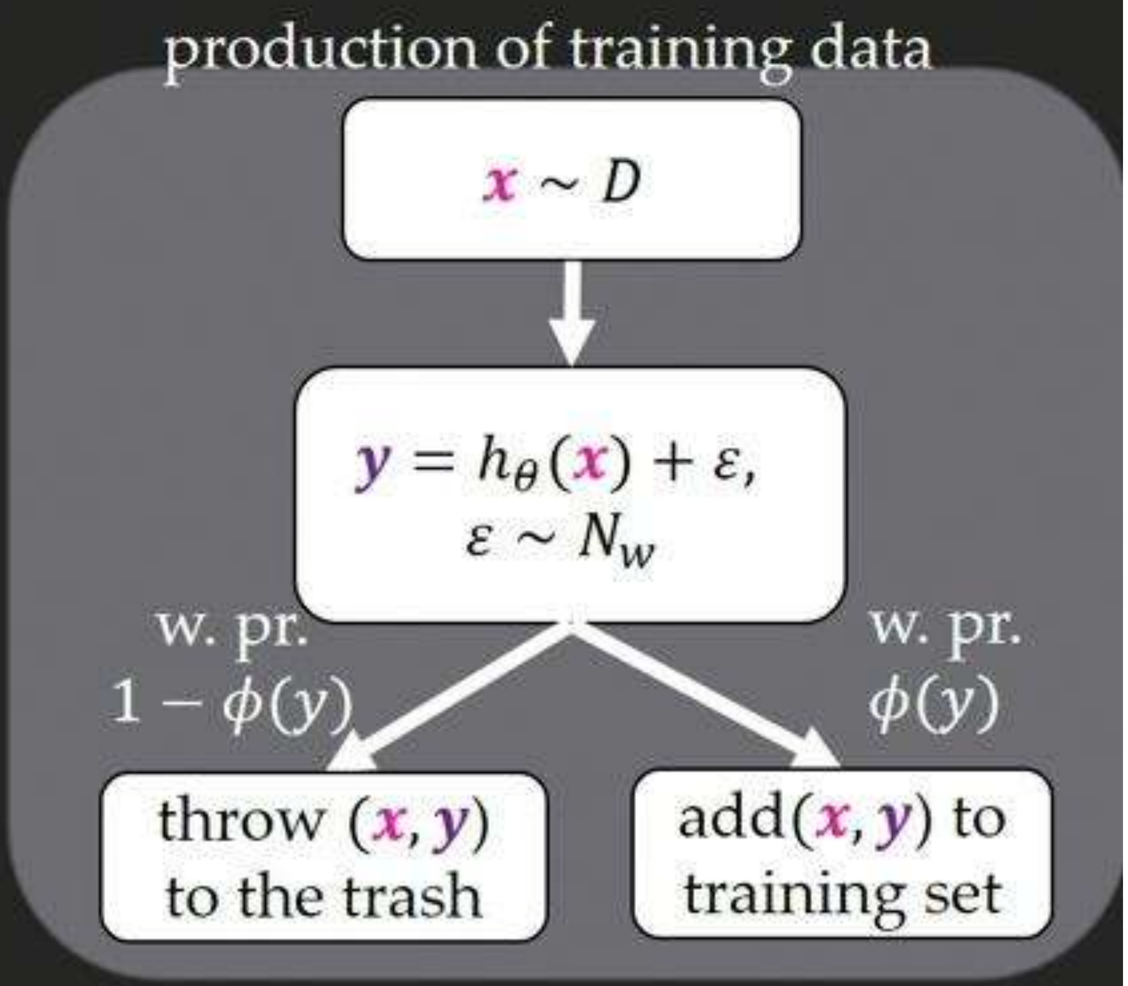
Menu

- **Motivation**
- **Flavor of Models, Techniques, Results**

Problem 1: Truncation on the Y-Axis

(recall: IQ vs Earnings, Height vs Basketball)

Truncated Regression Model:



Motivating Eg 3: Truncation on the X-axis

Gender Classifier	Darker Male	Darker Female	Lighter Male	Lighter Female	Largest Gap
Microsoft	94.0%	79.2%	100%	98.3%	20.8%
FACE++	99.3%	65.5%	99.2%	94.0%	33.8%
IBM	88.0%	65.3%	99.7%	92.9%	34.4%

Buolamwini, Gebru, FAT 2018

Explanation: Training data contains more faces that are of lighter skin tone, male gender, Caucasian

⇒ Training loss of gender classifier pays less attention to faces that are of darker skin tone, female gender, non-Caucasian

⇒ Test loss on faces that are of darker skin tone, female gender, non-Caucasian is worse

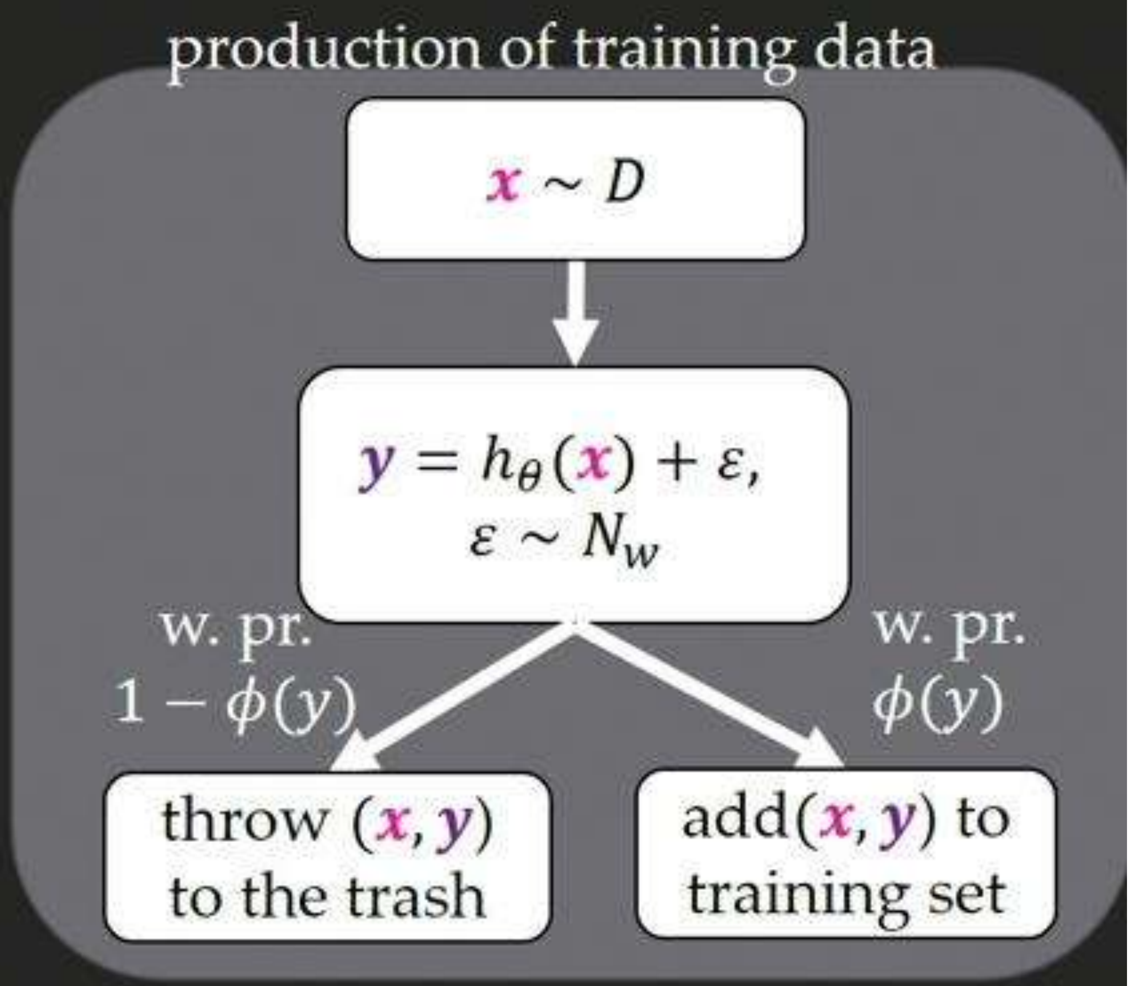
Classical example of bias in ML systems

Problem 1: Truncation on the Y-Axis

(recall: IQ vs Earnings, Height vs Basketball)

Truncated Regression Model:

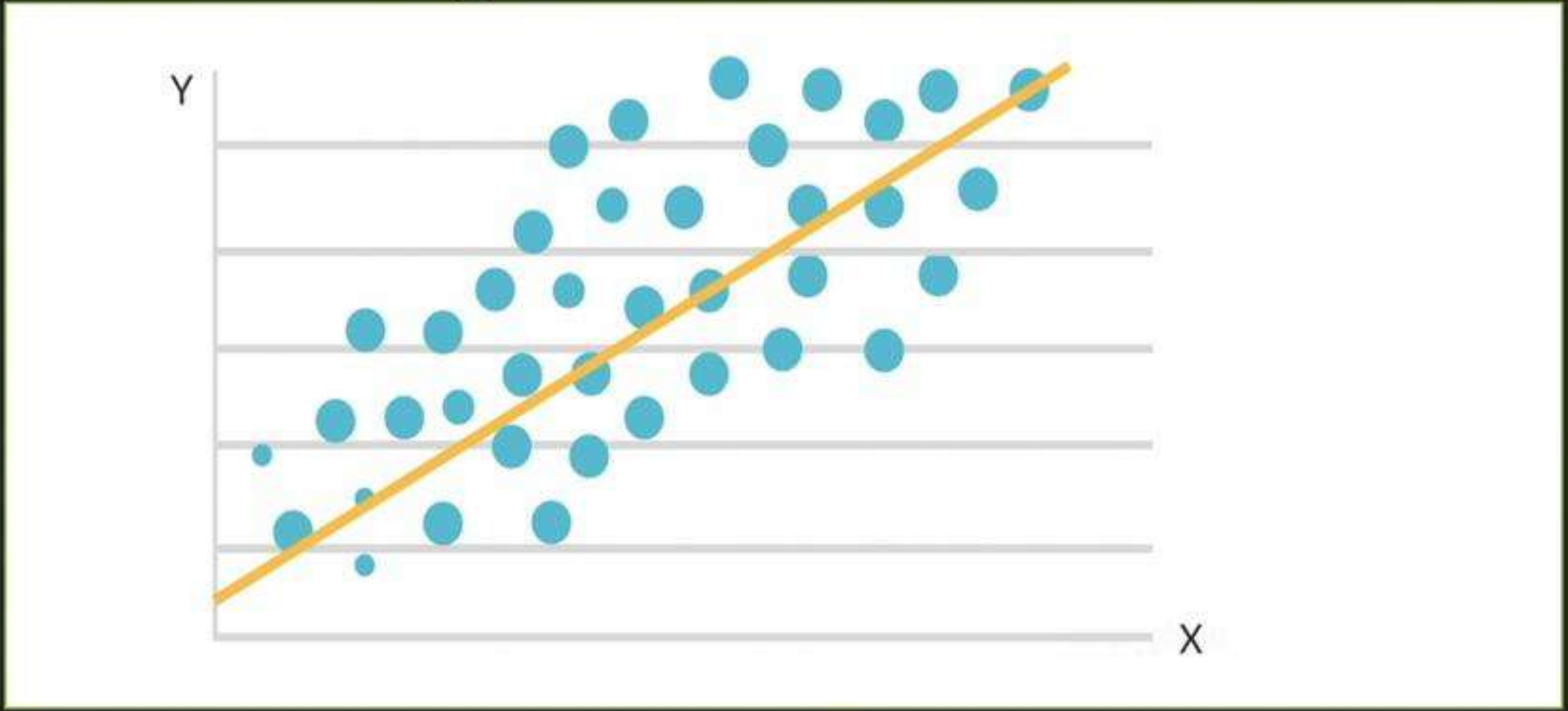
- (*unknown*) distribution D over covariates x
- (*unknown*) response mechanism $h_\theta: x \mapsto y$, $\theta \in \Theta$
- (*unknown*) noise distribution N_w, $w \in \Omega$
- (*known*) filtering mechanism $\phi: y \mapsto p \in [0,1]$

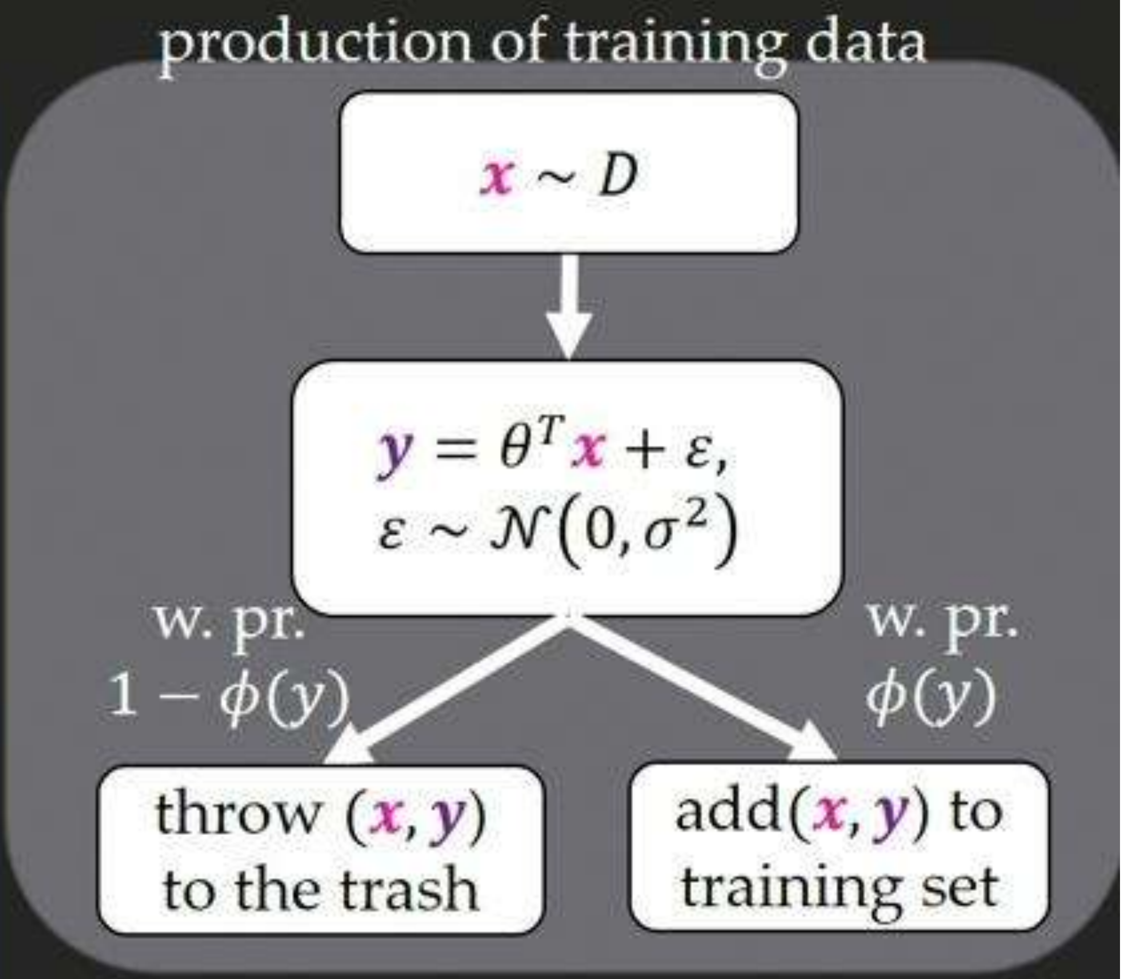


Problem 1: Truncation on the Y-Axis

(recall: IQ vs Earnings, Height vs Basketball)

Truncated Regression Model:

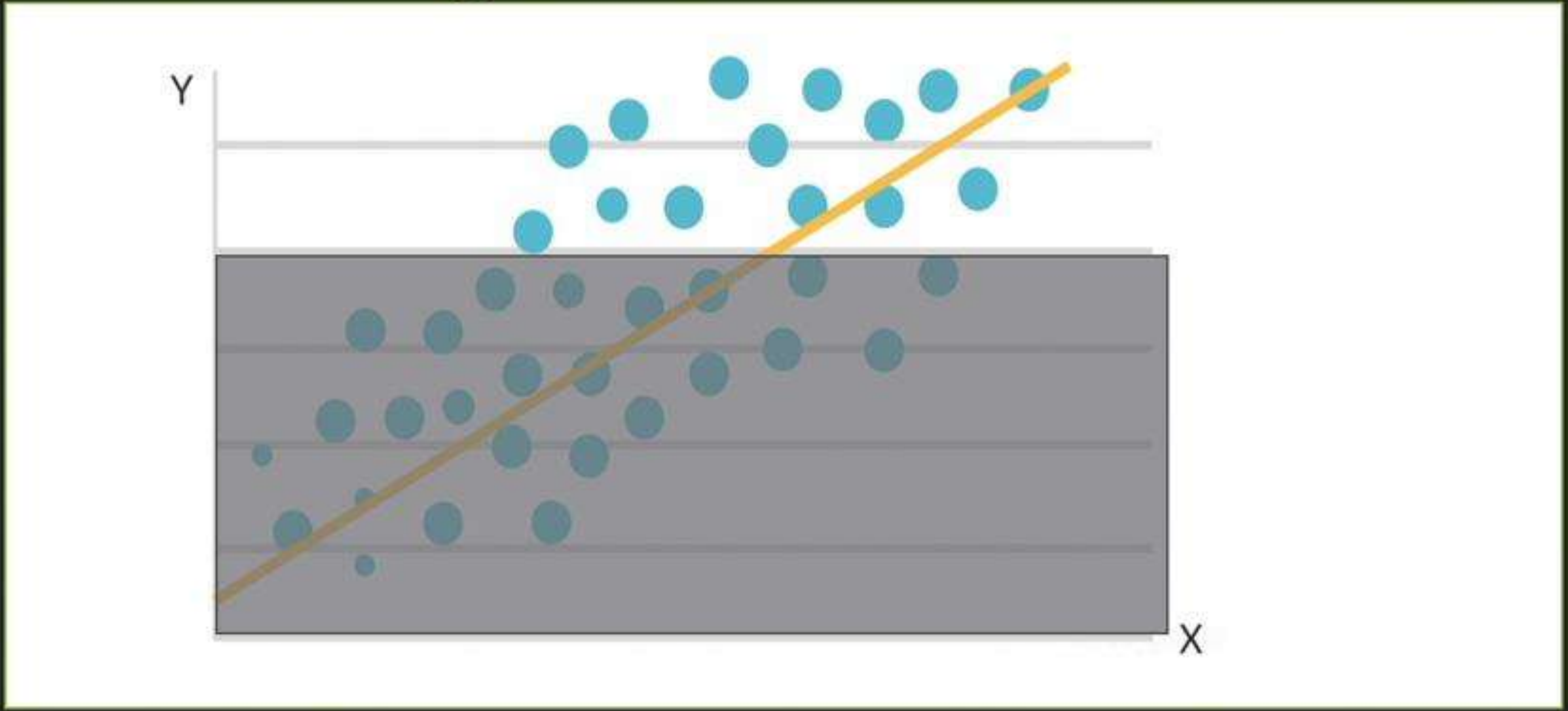




Problem 1: Truncation on the Y-Axis

(recall: IQ vs Earnings, Height vs Basketball)

Truncated Regression Model:



production of training data

$\boldsymbol{x} \sim D$

$\boldsymbol{y} = \theta^T \boldsymbol{x} + \varepsilon,$
$\varepsilon \sim \mathcal{N}(0, \sigma^2)$

w. pr. $1 - \phi(y)$: throw $(\boldsymbol{x}, \boldsymbol{y})$ to the trash

w. pr. $\phi(y)$: add $(\boldsymbol{x}, \boldsymbol{y})$ to training set

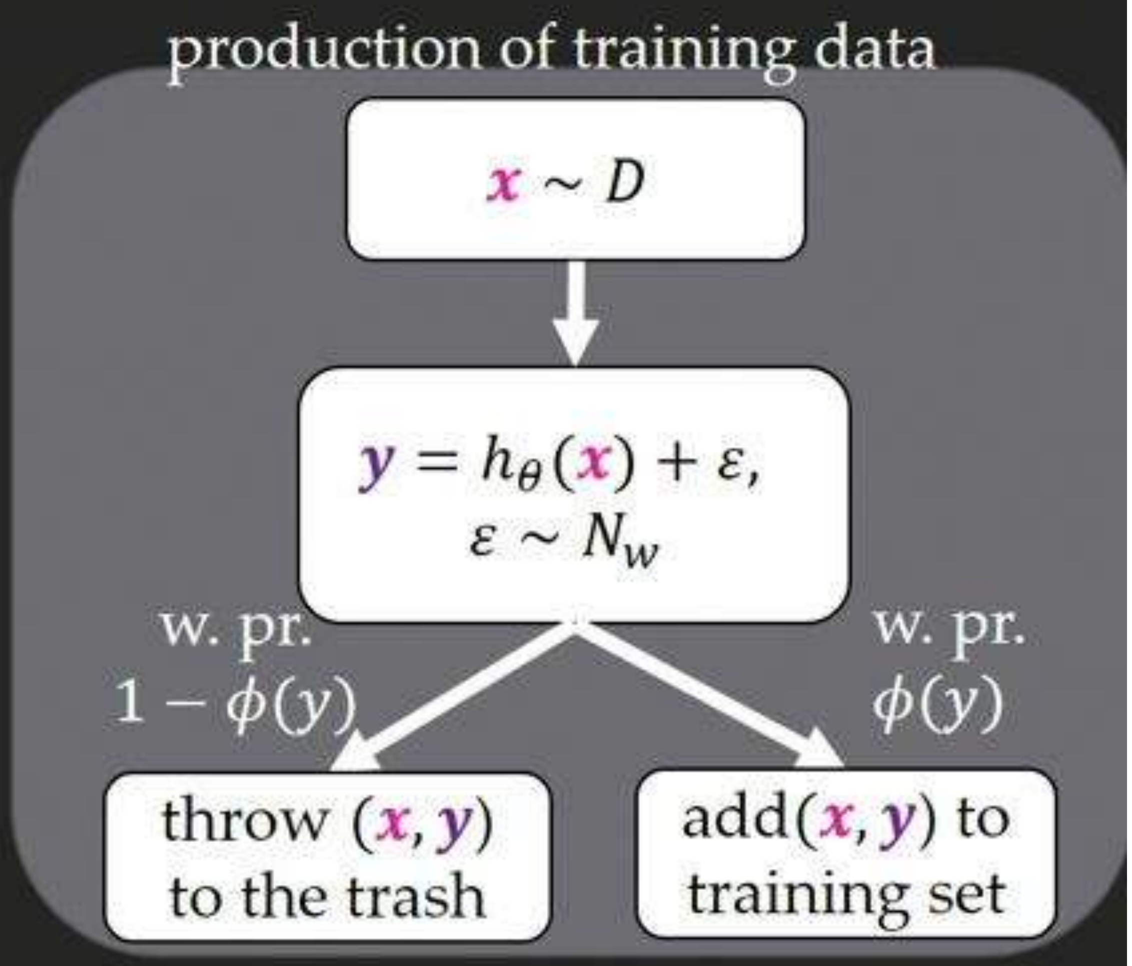
Problem 1: Truncation on the Y-Axis

(recall: IQ vs Earnings, Height vs Basketball)

Truncated Regression Model:

- (*unknown*) distribution D over covariates x
- (*unknown*) response mechanism $h_\theta : x \mapsto y$, $\theta \in \Theta$
- (*unknown*) noise distribution N_w
- (*known*) filtering mechanism $\phi : y \mapsto p \in [0,1]$

Goal: given filtered data $(x_i, y_i)_i$ recover θ

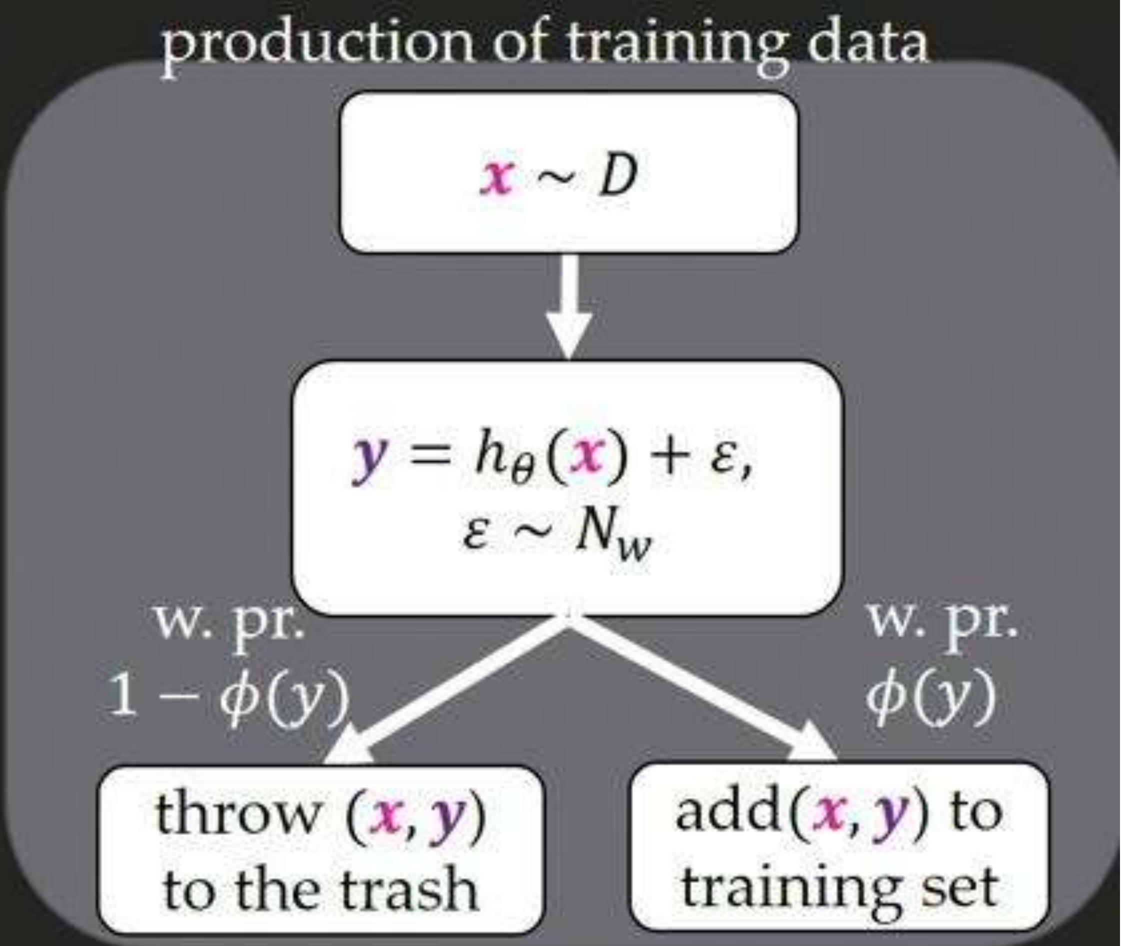


Problem 1: Truncation on the Y-Axis

(recall: IQ vs Earnings, Height vs Basketball)

Truncated Regression Model:

- (*unknown*) distribution D over covariates x
- (*unknown*) response mechanism $h_\theta: x \mapsto y,\ \theta \in \Theta$
- (*unknown*) noise distribution N_w
- (*known*) filtering mechanism $\phi: y \mapsto p \in [0,1]$



Goal: given filtered data $(x_i, y_i)_i$ recover θ

Results [w/ Gouleakis, Tzamos, Zampetakis COLT'19, w/ Ilyas, Rao, Zampetakis'19] :

- Practical, SGD-based likelihood optimization framework
- Computationally and statistically efficient recovery of true parameters for truncated linear/probit*/logistic regression*
  - **prior work:** inefficient algorithms, and error rates exponential in dimension

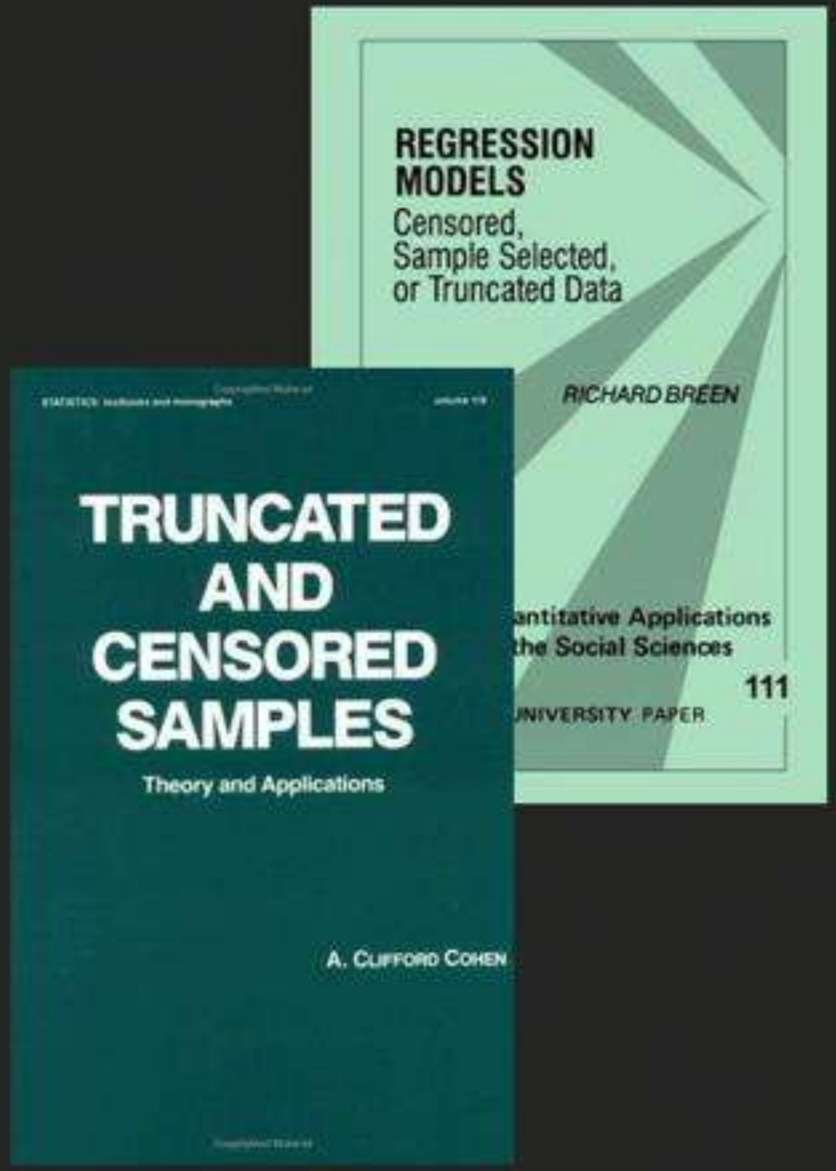
Comparison to Prior Work On Truncated Regression

Asymptotic Analysis of Truncated/Censored Regression [Tobin 1958], [Amemiya 1973], [Hausman, Wise 1976], [Breen 1996], [Hajivassiliou-McFadden'97], [Balakrishnan, Cramer 2014], Limited Dependent Variables models, Method of Simulated Scores, GHK Algorithm

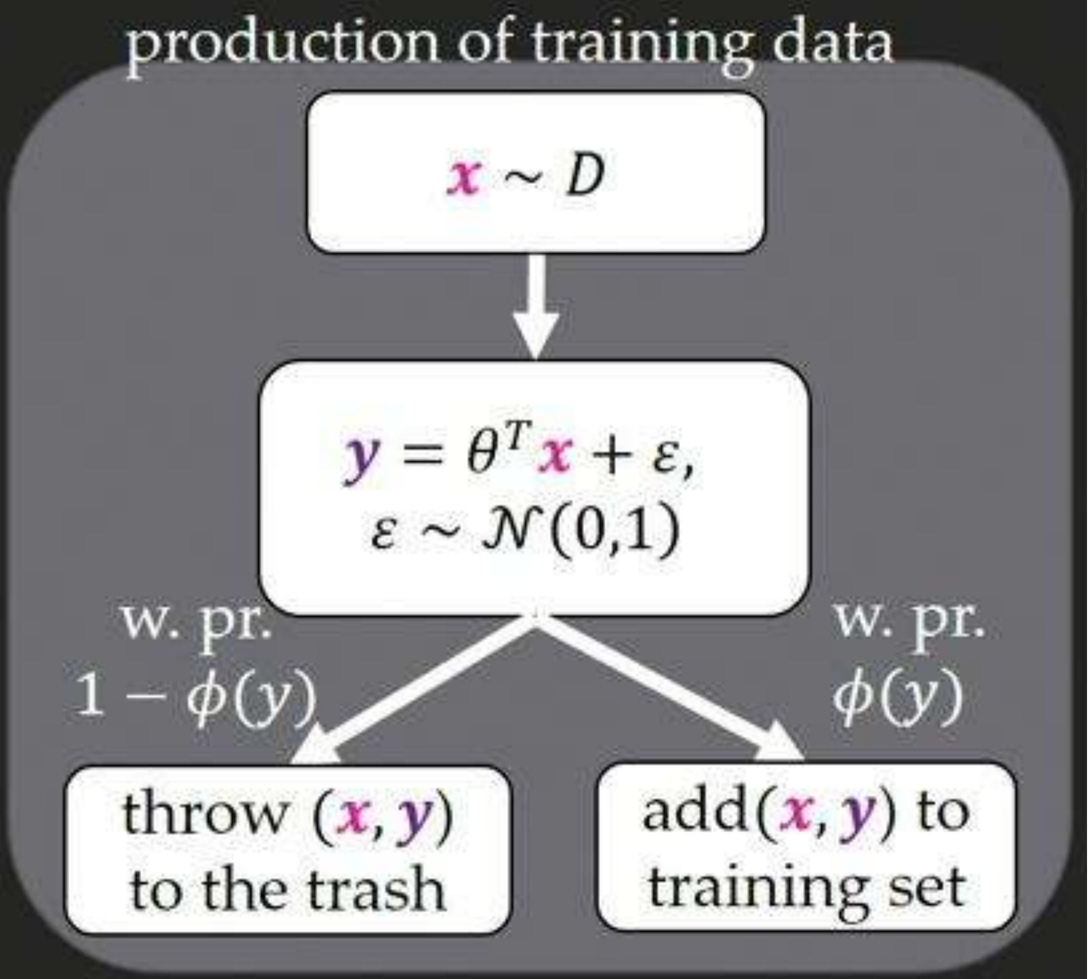
Technical Bottlenecks:

- Convergence rates: $O_d\left(\frac{1}{\sqrt{n}}\right)$
- Computationally inefficient algorithms

Our work: optimal rates $O\left(\sqrt{\frac{d}{n}}\right)$, efficient algorithms, arbitrary truncation sets

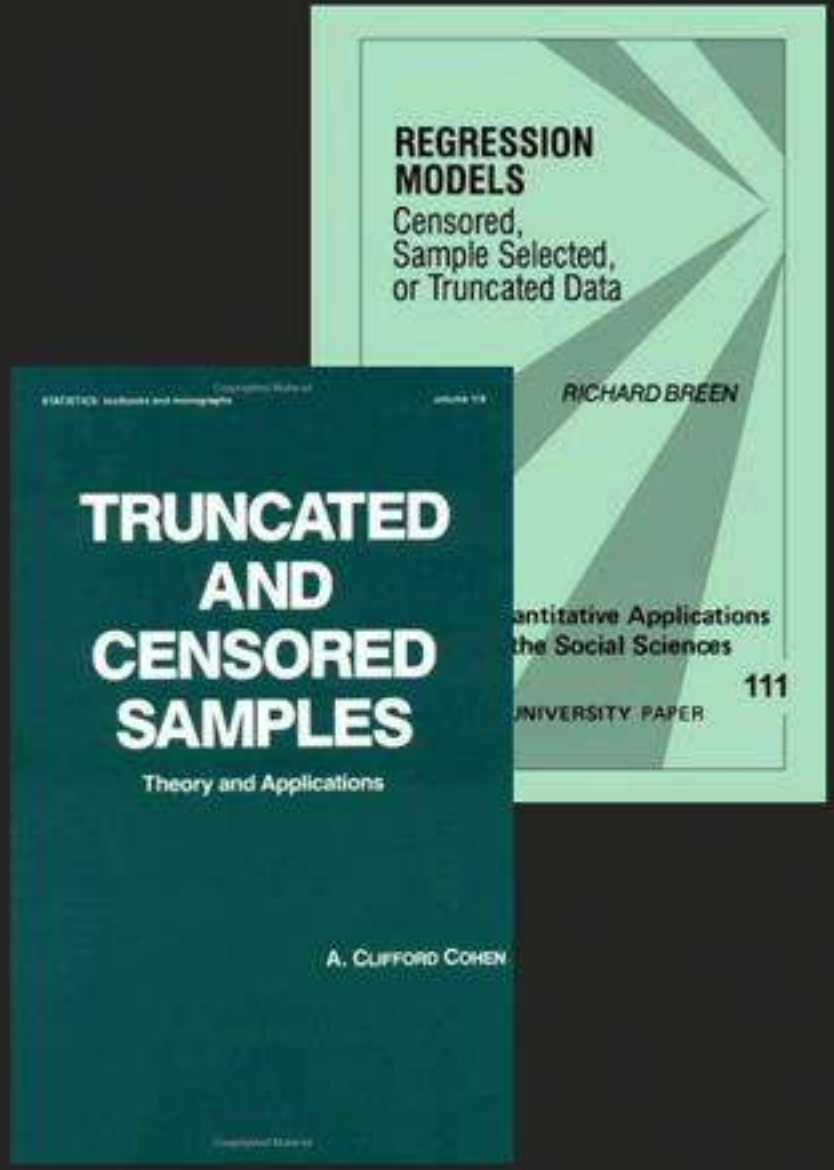


Technical Vignette: Truncated Linear Regression



Comparison to Prior Work On Truncated Regression

Asymptotic Analysis of Truncated/Censored Regression [Tobin 1958], [Amemiya 1973], [Hausman, Wise 1976], [Breen 1996], [Hajivassiliou-McFadden'97], [Balakrishnan, Cramer 2014], Limited Dependent Variables models, Method of Simulated Scores, GHK Algorithm



Technical Bottlenecks:

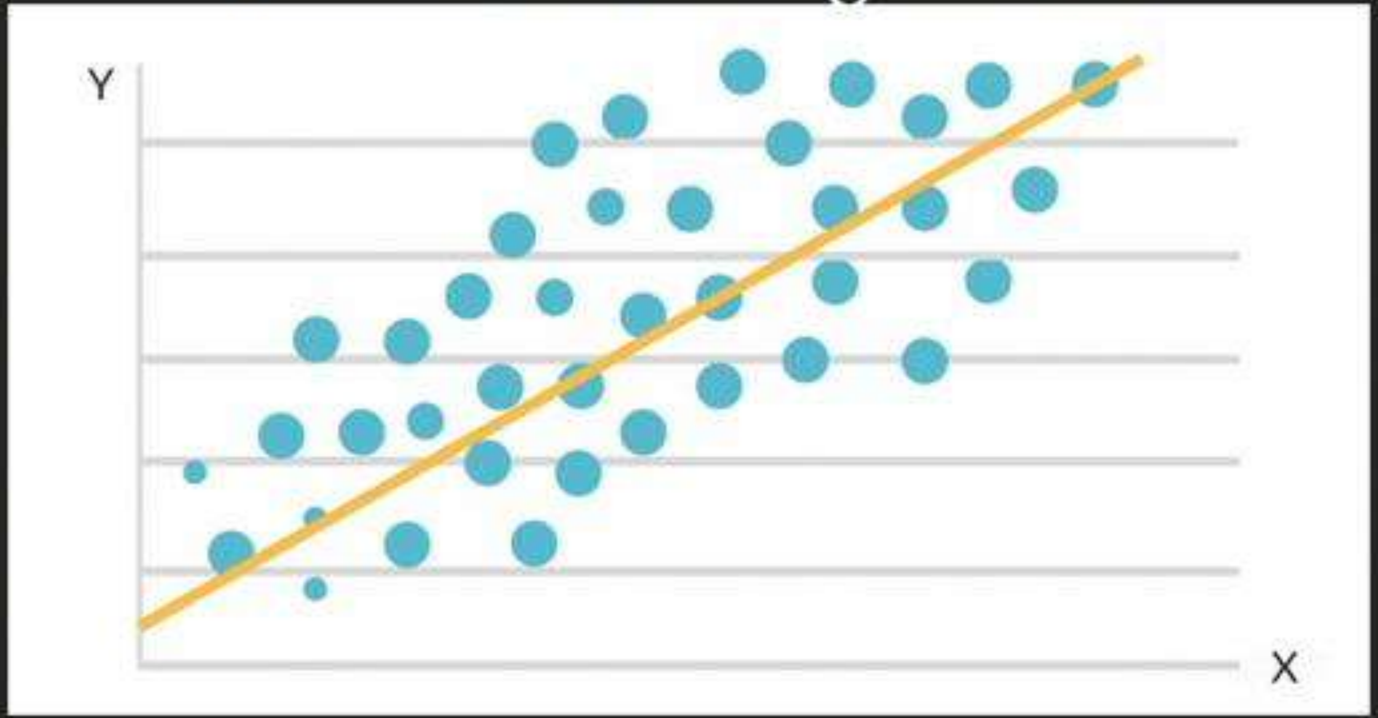
- Convergence rates: $O_d\left(\frac{1}{\sqrt{n}}\right)$
- Computationally inefficient algorithms

Our work: optimal rates $O\left(\sqrt{\frac{d}{n}}\right)$, efficient algorithms, arbitrary truncation sets

What Happened?

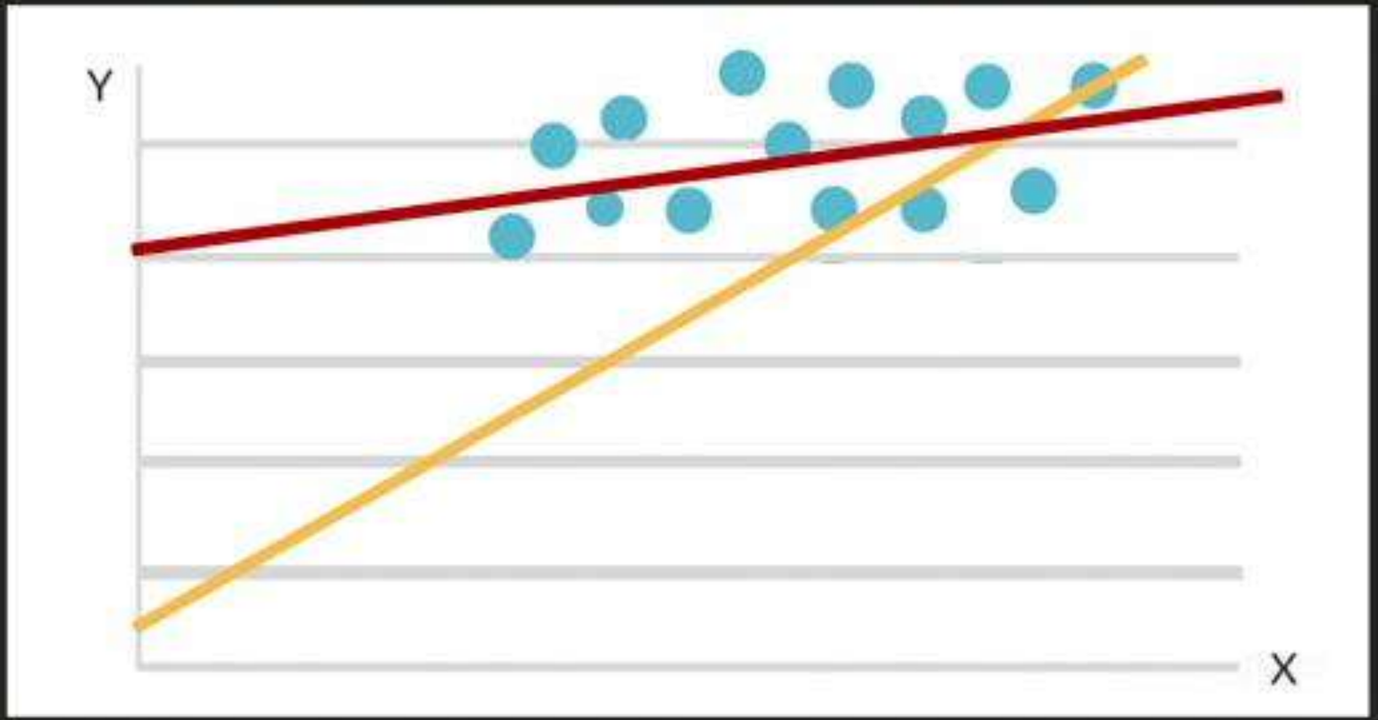
Mental Picture:

Vanilla Linear Regression



Truth: $y_i = \theta \cdot x_i + \varepsilon_i$, for all i

Data truncated on the Y-axis



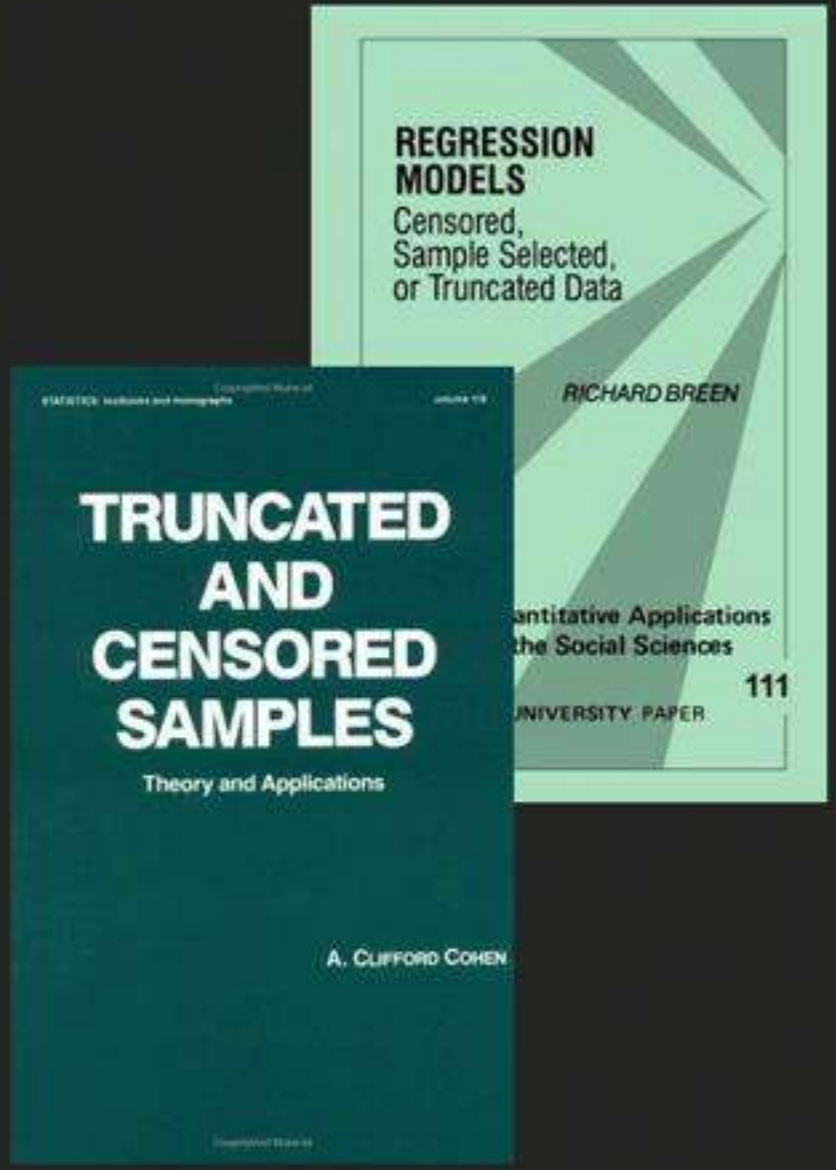
Comparison to Prior Work On Truncated Regression

Asymptotic Analysis of Truncated/Censored Regression [Tobin 1958], [Amemiya 1973], [Hausman, Wise 1976], [Breen 1996], [Hajivassiliou-McFadden'97], [Balakrishnan, Cramer 2014], Limited Dependent Variables models, Method of Simulated Scores, GHK Algorithm

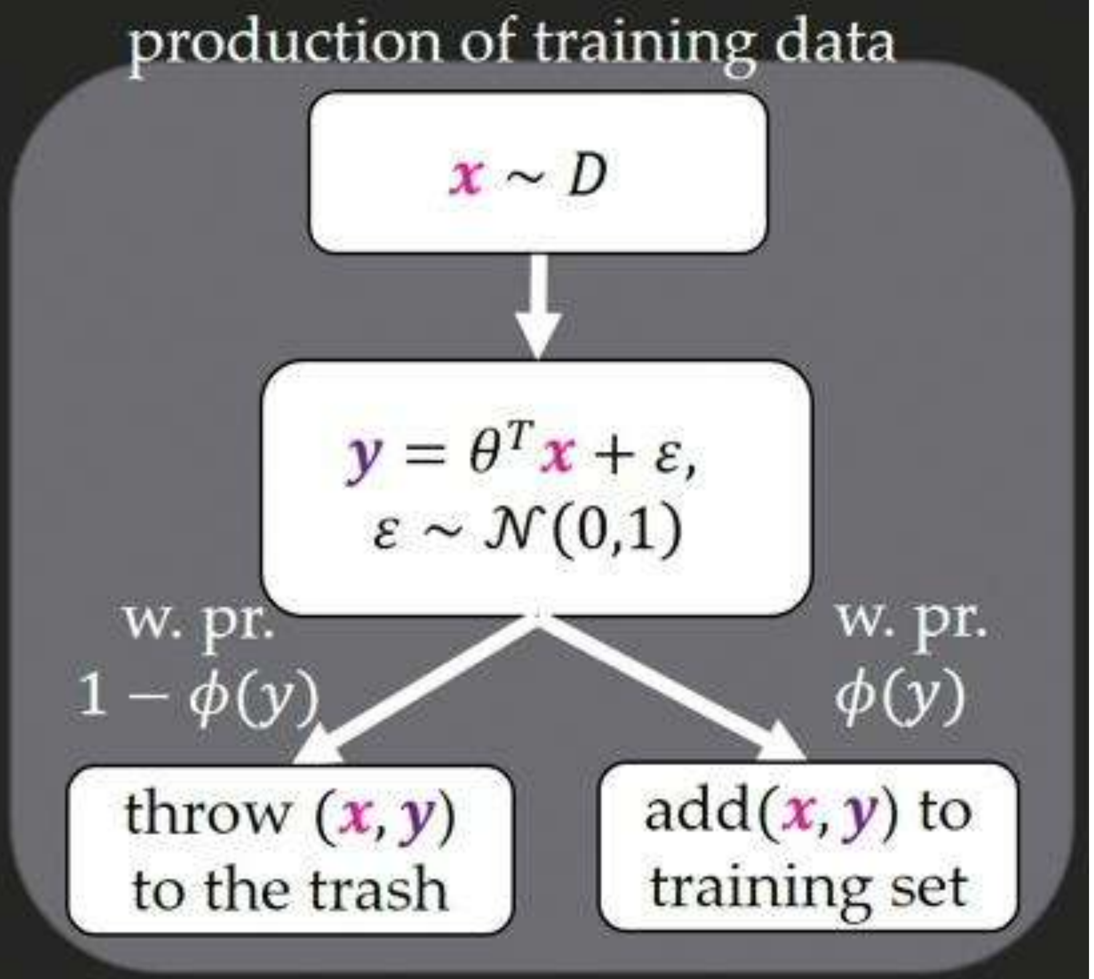
Technical Bottlenecks:

- Convergence rates: $O_d\left(\frac{1}{\sqrt{n}}\right)$
- Computationally inefficient algorithms

Our work: optimal rates $O\left(\sqrt{\frac{d}{n}}\right)$, efficient algorithms, arbitrary truncation sets

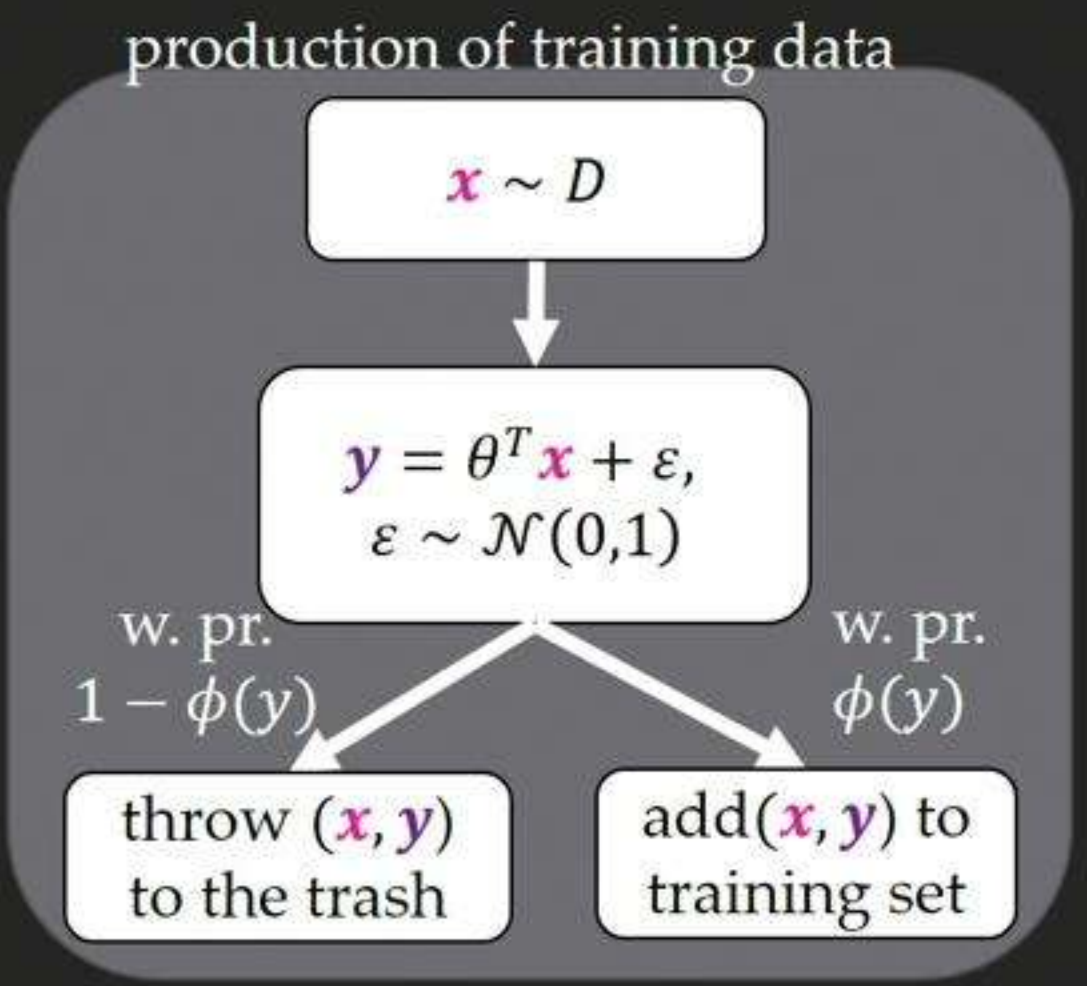


Technical Vignette: Truncated Linear Regression



Technical Vignette: Truncated Linear Regression

Data distribution: $p_\theta(x,y) = \frac{1}{Z} \cdot D(x) \cdot e^{-\frac{(y-\theta^T x)^2}{2}} \cdot \phi(y)$

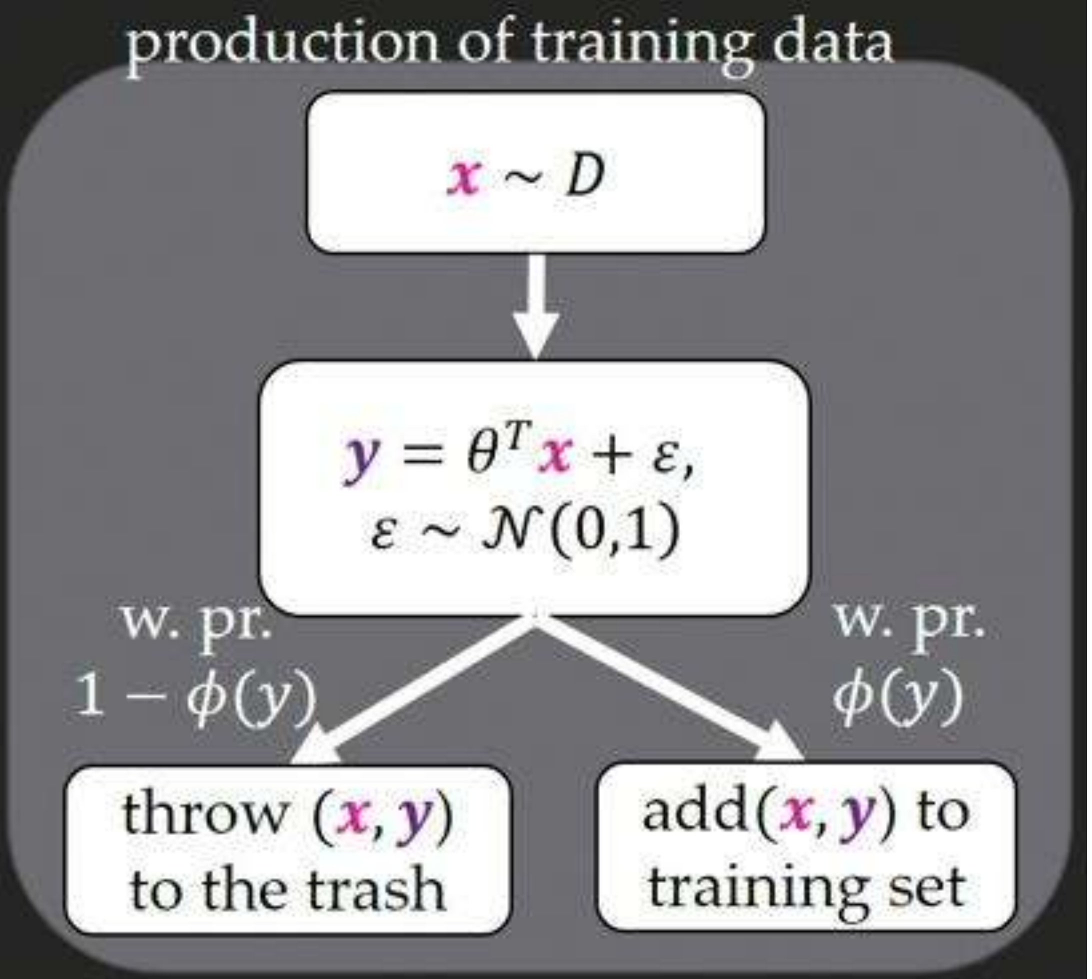


Technical Vignette: Truncated Linear Regression

Data distribution: $p_\theta(x,y) = \frac{1}{Z} \cdot D(x) \cdot e^{-\frac{(y-\theta^T x)^2}{2}} \cdot \phi(y)$

Population Log-Likelihood:

$$\mathrm{LL}(\theta) = \mathbb{E}_{(x,y)\sim \mathrm{p}_{\theta^*}^{\mathrm{train}}}\left[\log D(x) - \frac{(y-\theta^T x)^2}{2} + \log\phi(y) - \log Z\right]$$



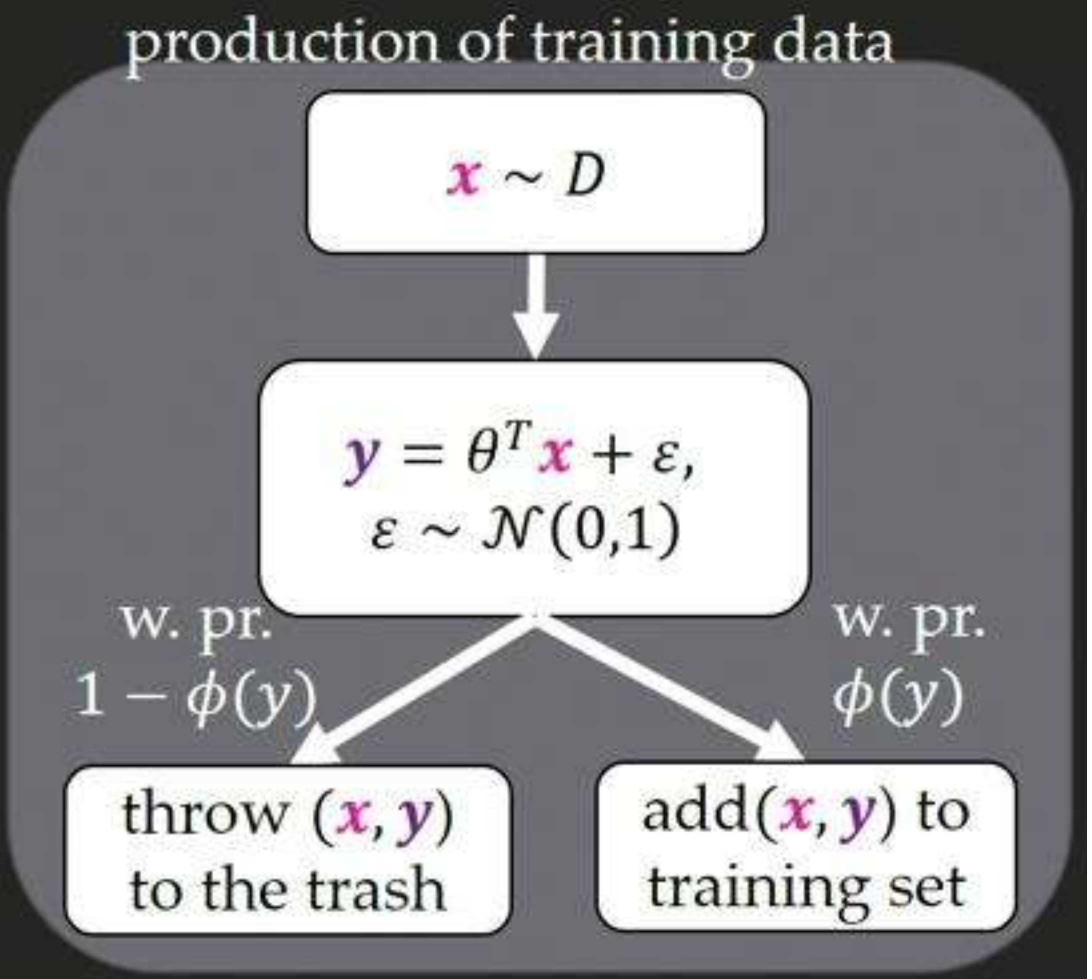
Technical Vignette: Truncated Linear Regression

Data distribution: $p_\theta(x, y) = \frac{1}{Z} \cdot D(x) \cdot e^{-\frac{(y-\theta^T x)^2}{2}} \cdot \phi(y)$

Population Log-Likelihood:

$$\mathrm{LL}(\theta) = \mathbb{E}_{(x,y)\sim \mathrm{p}_{\theta^*}^{\mathrm{train}}}\left[\log D(x) - \frac{(y-\theta^T x)^2}{2} + \log \phi(y) - \log Z\right]$$

Issue: $\mathrm{LL}(\theta)$ involves stuff we don't know (D), and even if we did it involves stuff we wouldn't be able to tractably calculate (Z)



Technical Vignette: Truncated Linear Regression

Data distribution: $p_\theta(x,y) = \frac{1}{Z}\cdot D(x)\cdot e^{-\frac{\left(y-\theta^T x\right)^2}{2}}\cdot \phi(y)$

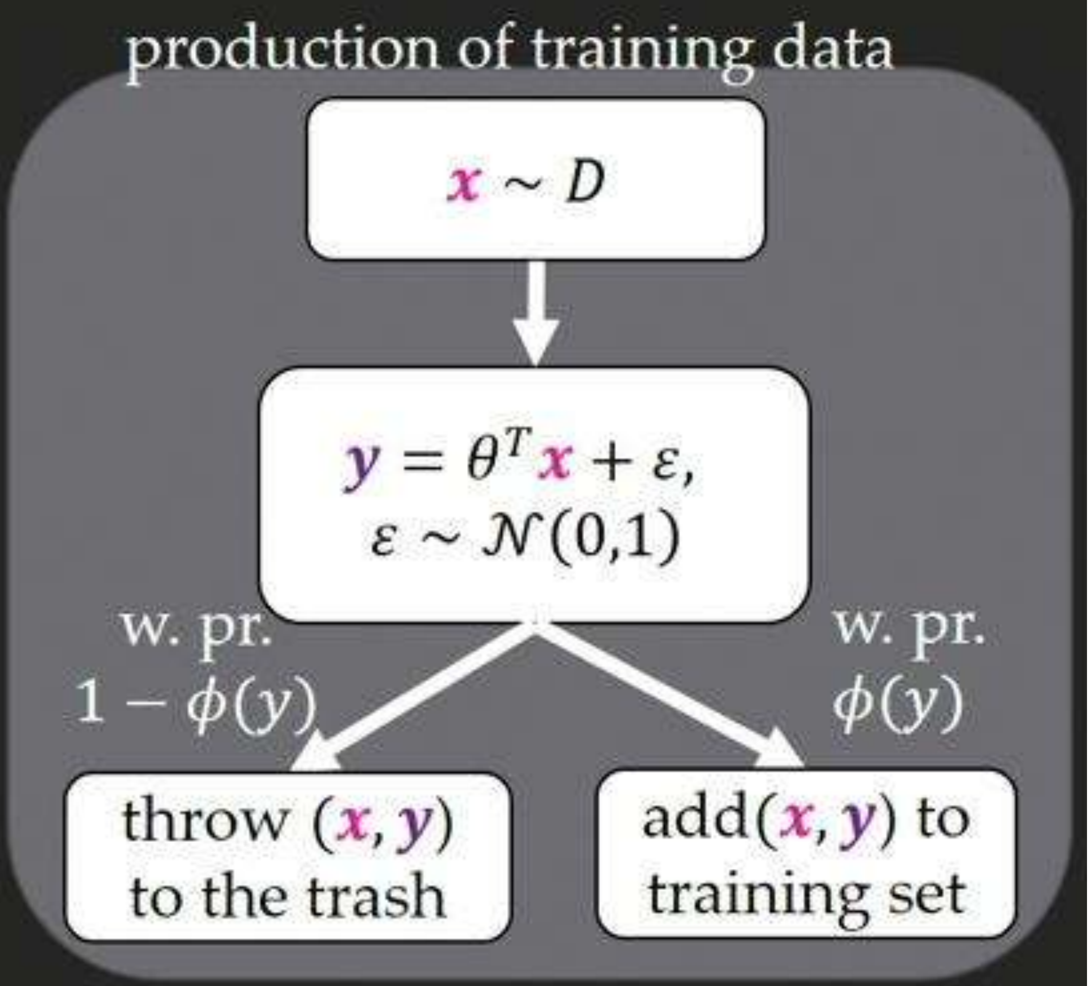
Population Log-Likelihood:

$$\mathrm{LL}(\theta) = \mathbb{E}_{(x,y)\sim \mathrm{p}_{\theta^*}^{\mathrm{train}}}\left[\log D(x) - \frac{\left(y-\theta^T x\right)^2}{2} + \log\phi(y) - \log Z\right]$$

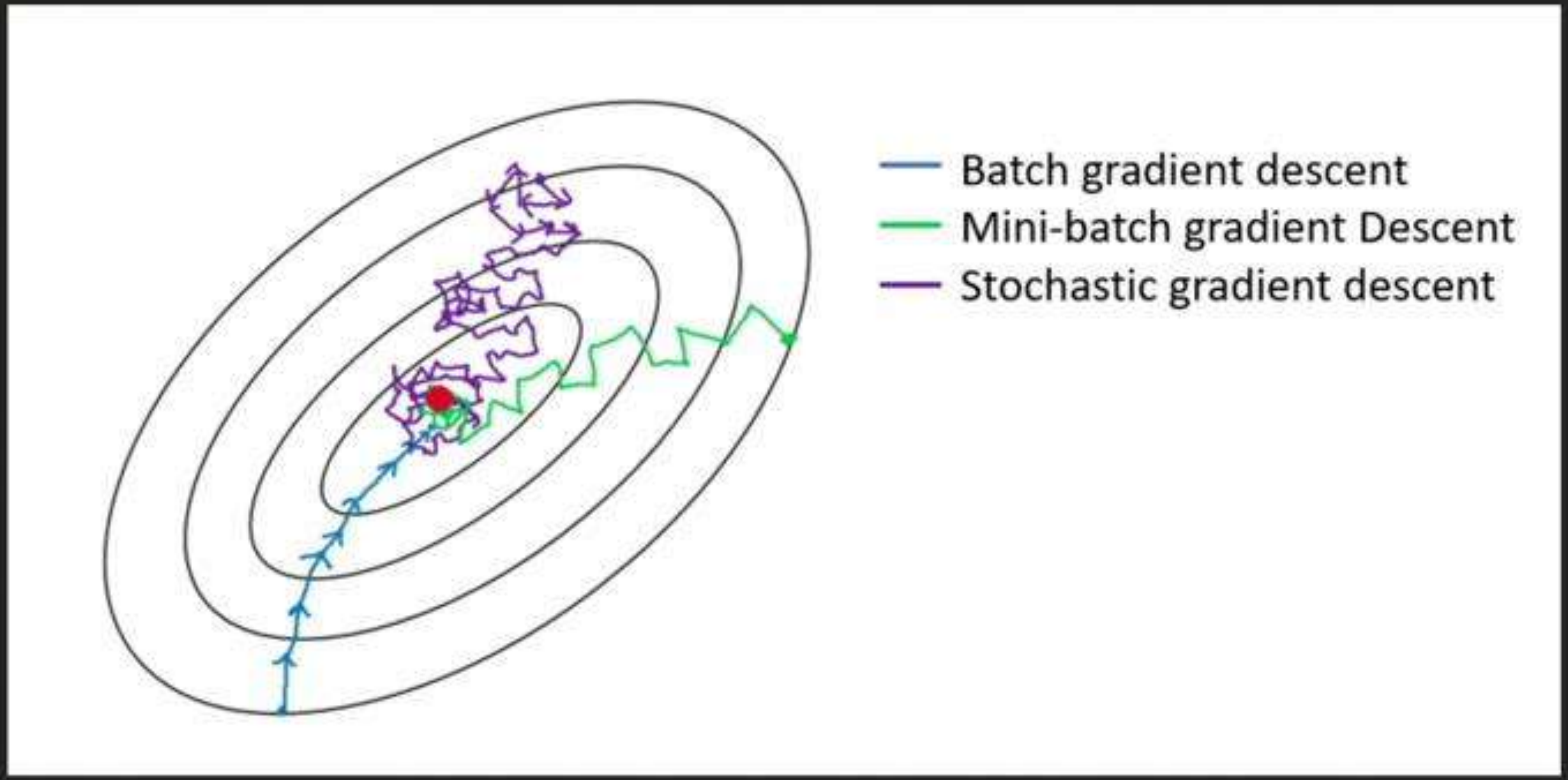
Issue: $\mathrm{LL}(\theta)$ involves stuff we don't know (D), and even if we did it involves stuff we wouldn't be able to tractably calculate (Z)

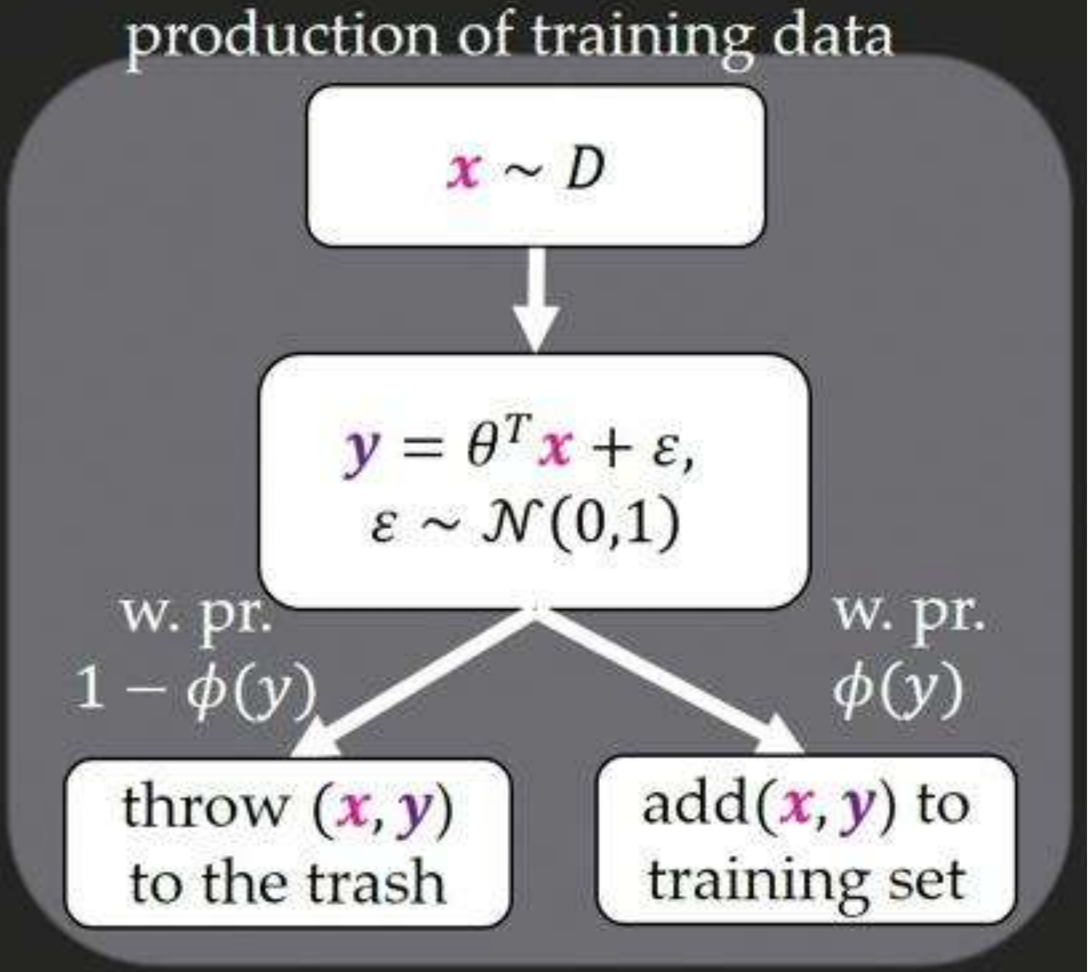
Yet, Stochastic Gradient Descent (SGD) *can* be performed on negative log-likelihood

In particular, easy to define random variable whose expectation is the gradient at a given θ, without knowledge of D and no need to compute Z



Technical Vignette: Truncated Linear Regression

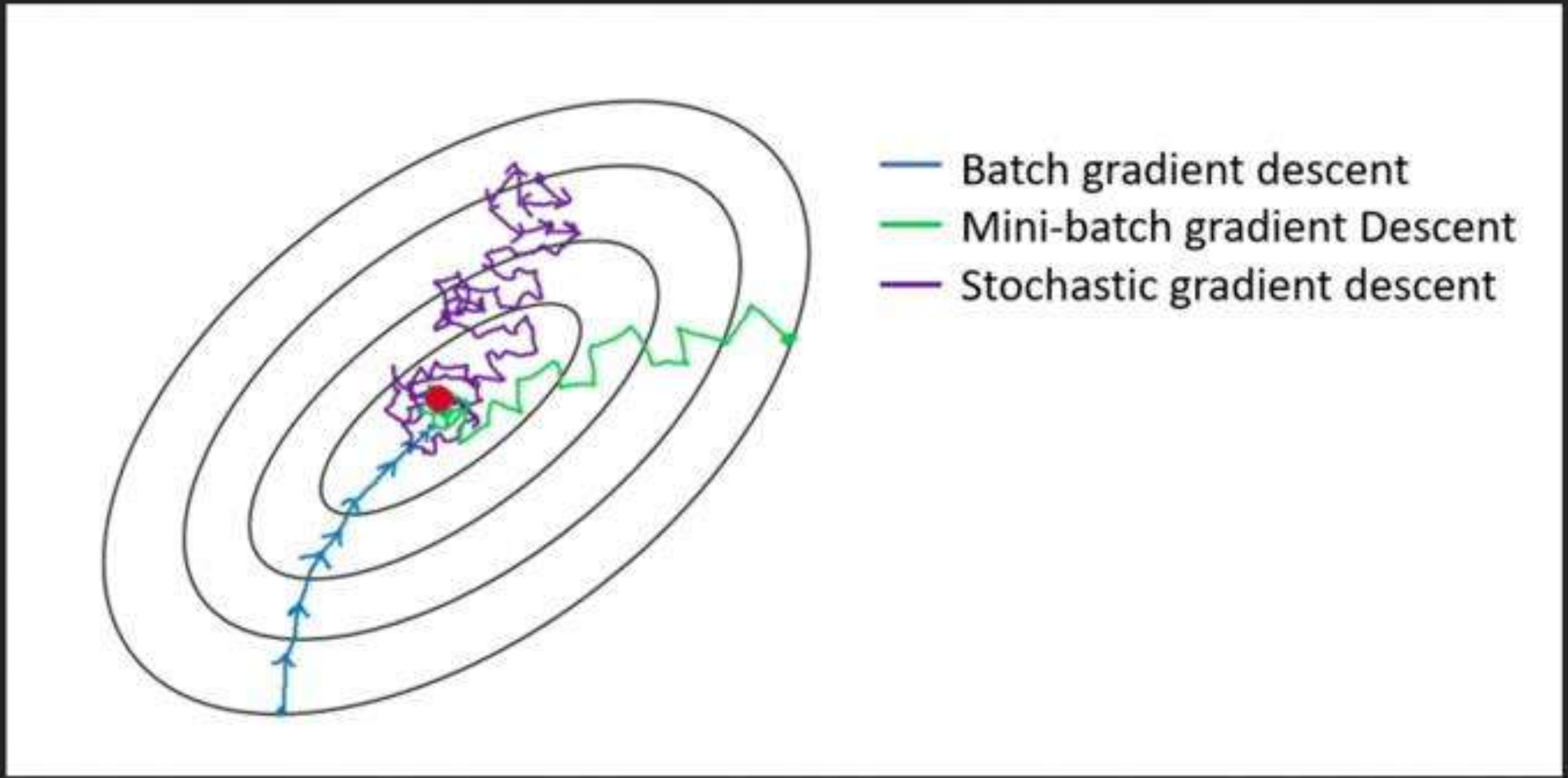




$$\mathrm{LL}(\theta) = \mathbb{E}_{(x,y)\sim \mathrm{p}_{\theta^*}^{\mathrm{train}}}\left[\log D(x) - \frac{\left(y - \theta^T x\right)^2}{2} + \log \phi(y) - \log Z\right]$$

$$\nabla_\theta \mathrm{LL}(\theta) = \mathbb{E}_{(x,y)\sim \mathrm{p}_{\theta^*}^{\mathrm{train}}}\left[-\left(y - \theta^T x\right)x\right] - \mathbb{E}_{(x,y)\sim \mathrm{p}_{\theta}^{\mathrm{train}}}\left[-\left(y - \theta^T x\right)x\right]$$

Technical Vignette: Truncated Linear Regression



$$\mathrm{LL}(\theta) = \mathbb{E}_{(x,y)\sim p_{\theta^*}^{\text{train}}}\left[\log D(x) - \frac{(y - \theta^T x)^2}{2} + \log \phi(y) - \log Z\right]$$

$$\nabla_\theta \mathrm{LL}(\theta) = \mathbb{E}_{(x,y)\sim p_{\theta^*}^{\text{train}}}\left[-(y - \theta^T x)x\right] - \mathbb{E}_{(x,y)\sim p_{\theta}^{\text{train}}}\left[-(y - \theta^T x)x\right]$$

Easy to define random variable whose expectation is the gradient at a given θ, without knowledge of D and no need to compute Z

Summary: We cannot run **blue** or **green**, but we *can* run **purple**

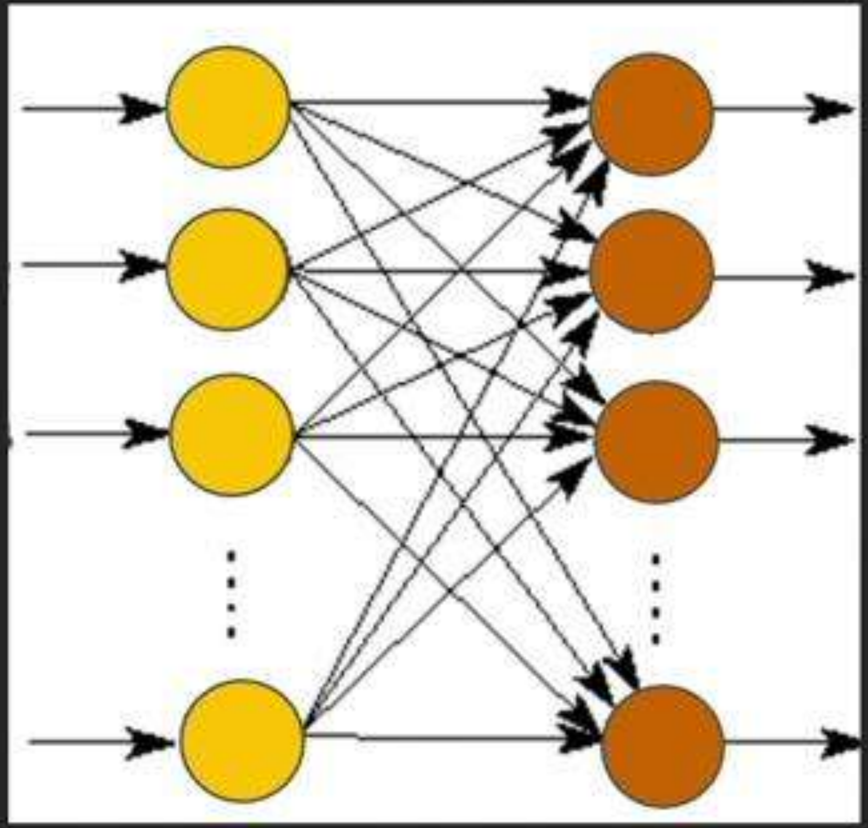
Issue 2: this random variable better be efficiently samplable, have small variance

Requires restricting optimization in appropriately defined space

Issue 3: for parameter estimation need neg. log-likelihood to be strongly convex

Requires (anti-)concentration of measure

E.g. Application: Learning Single-layer Relu Nets



 $= \text{Noisy-Relu} = \max\{0, w^T \cdot x + \varepsilon\}$,
where $\varepsilon \sim \mathcal{N}(0,1)$

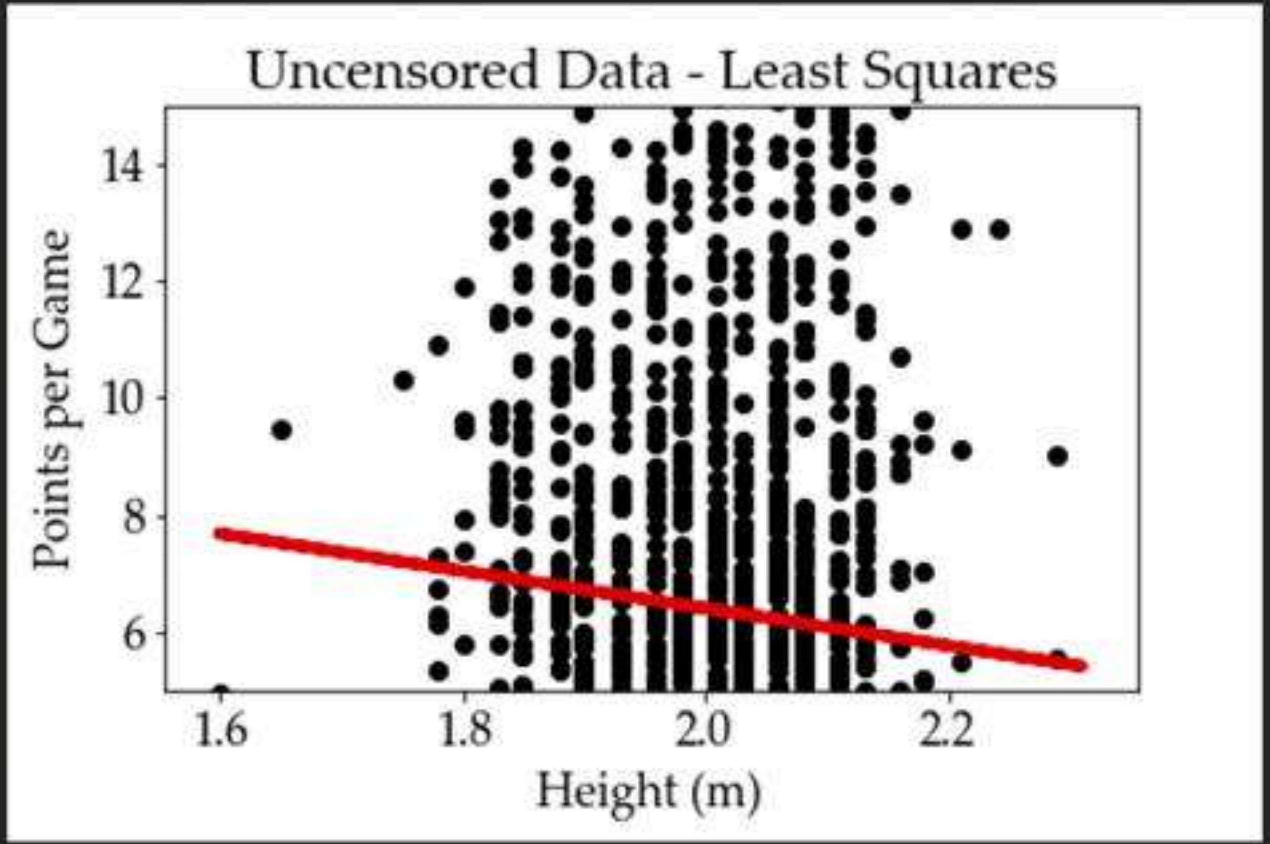
Direct corollary: In the realizable setting, given input-output pairs, obtain $O\left(\sqrt{\frac{input-dimension}{n}}\right)$ error rate

E.g. Application 2: NBA data

NBA player data after year 2000:

x_i: height of player i

y_i: number of points per game of player i



Points per Game are **negatively correlated** with height!

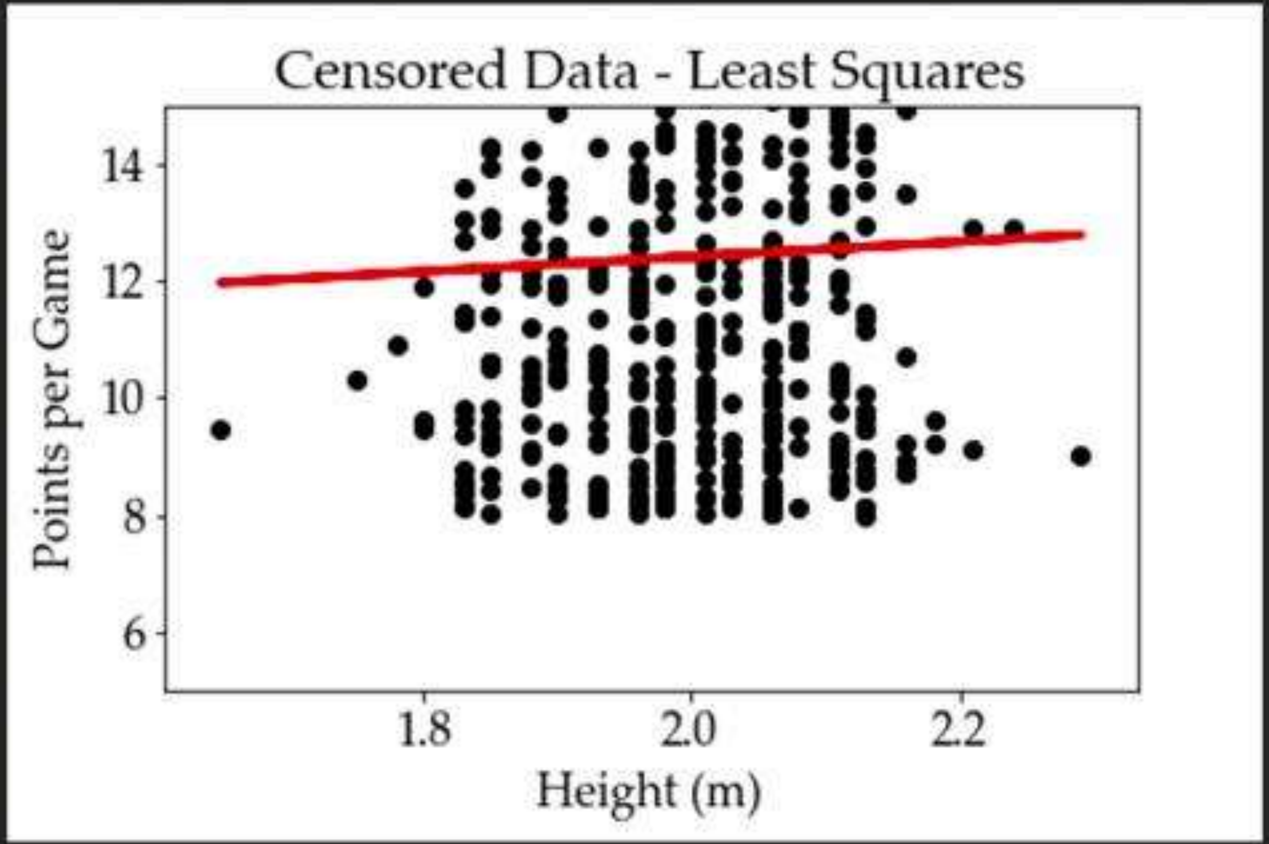
E.g. Application 2: NBA data

NBA player data after year 2000:

x_i: height of player i

y_i: number of points per game of player i

filtering : look at players with at least 8 points per game



Points per Game seem **positively correlated** with height!

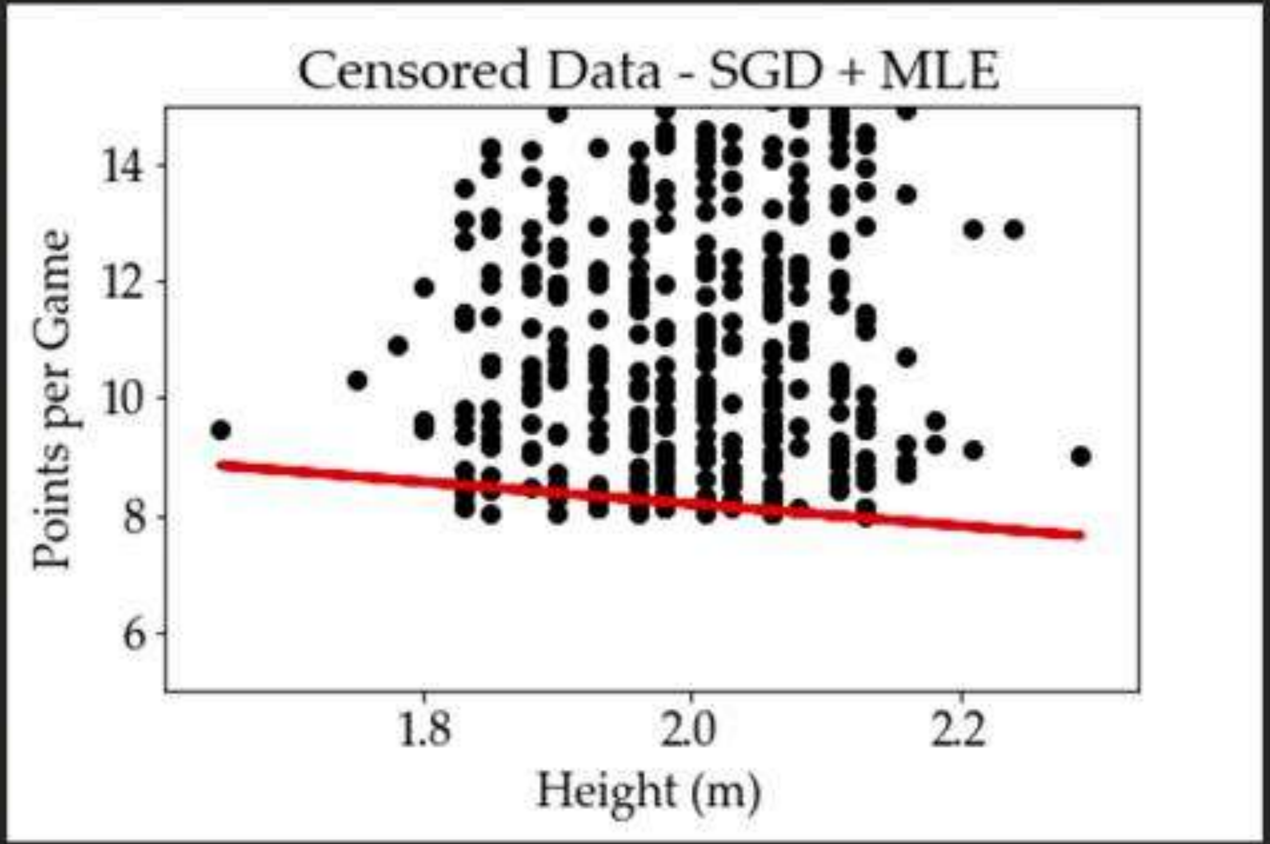
E.g. Application 2: NBA data

NBA player data after year 2000:

x_i: height of player i

y_i: number of points per game of player i

filtering : look at players with at least 8 points per game



Points per Game are **negatively correlated** with height!

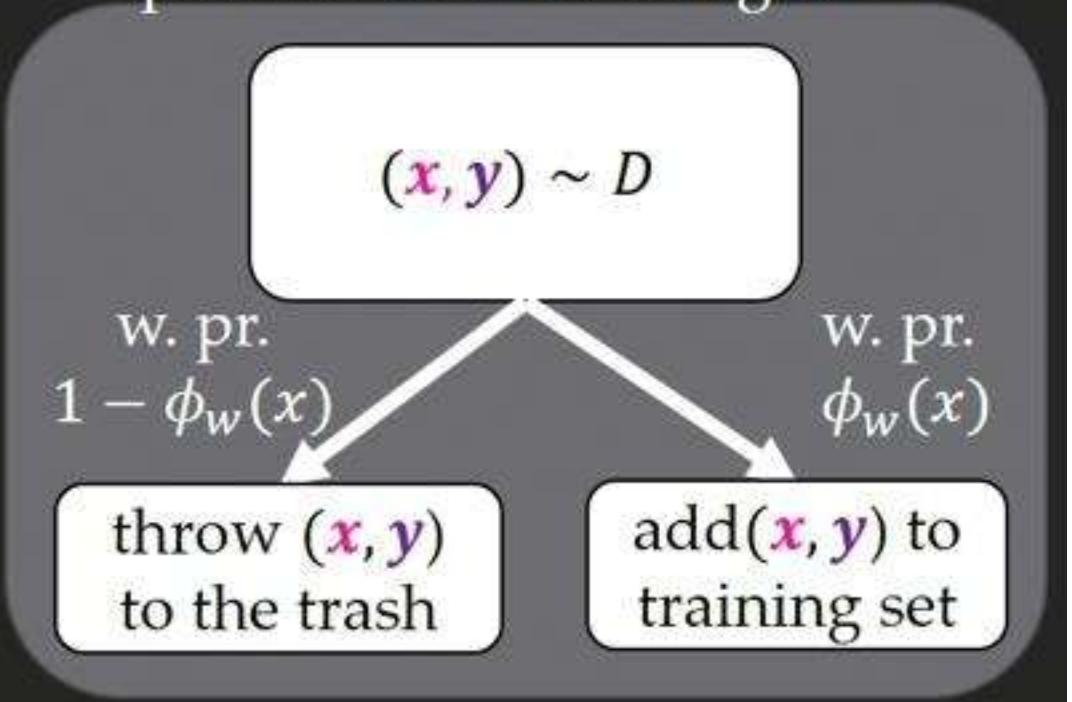
Problem 2: (Unknown) Truncation on the X-Axis

(recall: gender classification viz-a-viz skin tone)

Truncated Classification Model:

- (unknown) distribution D over uncensored image-label pairs $(x, y) \sim D$
- (unknown) filtering mechanism $\phi_w, w \in \Omega$, s.t. (x, y) is included in train set with probability $\phi_w(x)$
- (sample access) unlabeled image distribution D_x i.e. big enough test set (of images)

production of training data



- **Goals:** given filtered data $(x_i, y_i)_i$ and sample access to D_x (unfiltered image dist'n)
  - find image-to-label classifier minimizing *classification loss on uncensored data*
- **Results:** practical, SGD-based likelihood optimization framework [w/ Kontonis,Tzamos,Zampetakis]
  - alternative to other domain adaptation approaches

Example Application: Gender Classification 2

Train gender classifier on an adversarially constructed *balanced* training set of labeled male-female images, which predominantly contains images that a 95% accurate gender classifier misclassifies

Example Application: Gender Classification 2

Train gender classifier on an adversarially constructed *balanced* training set of labeled male-fe ains images that a 95% accurate gende

Example Application: Gender Classification 2

Train gender classifier on an adversarially constructed *balanced* training set of labeled male-female images, which predominantly contains images that a 95% accurate gender classifier misclassifies:

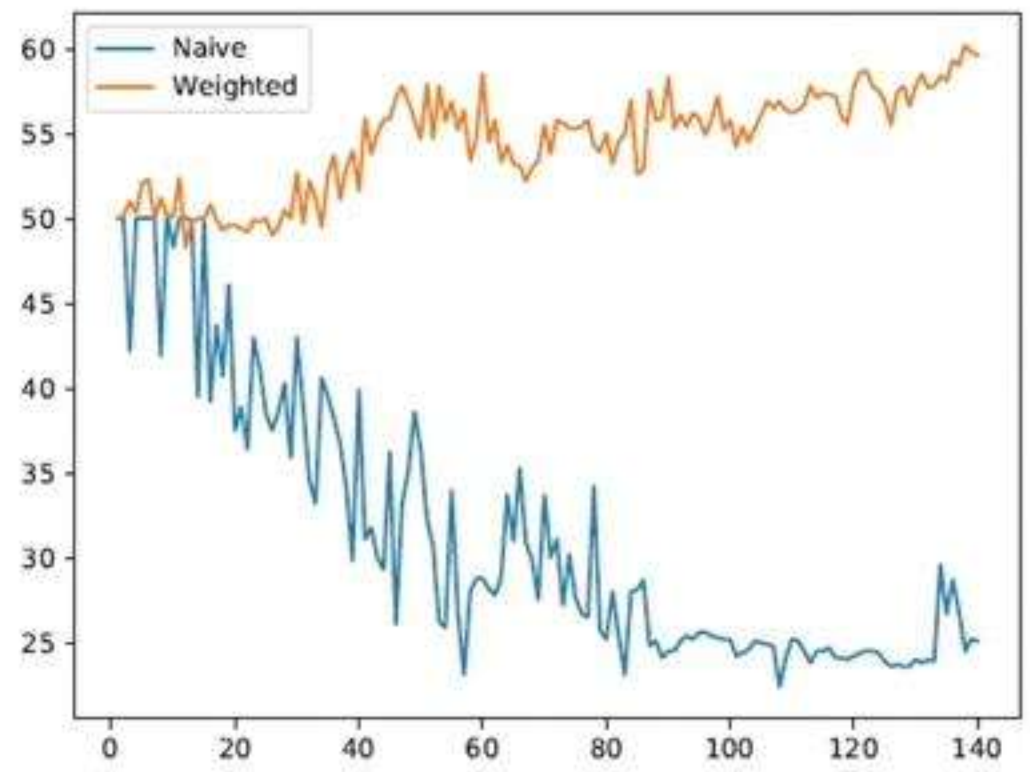
- use 1000 misclassified males and females, and 100 correctly classified males and females

Test classifier on a random balanced subset of CelebA dataset

Example Application: Gender Classification 2

Train gender classi *balanced* training set of
labeled male-femal tains images that a 95%
accurate gender cla

- use 1000 misclas orrectly classified males and females

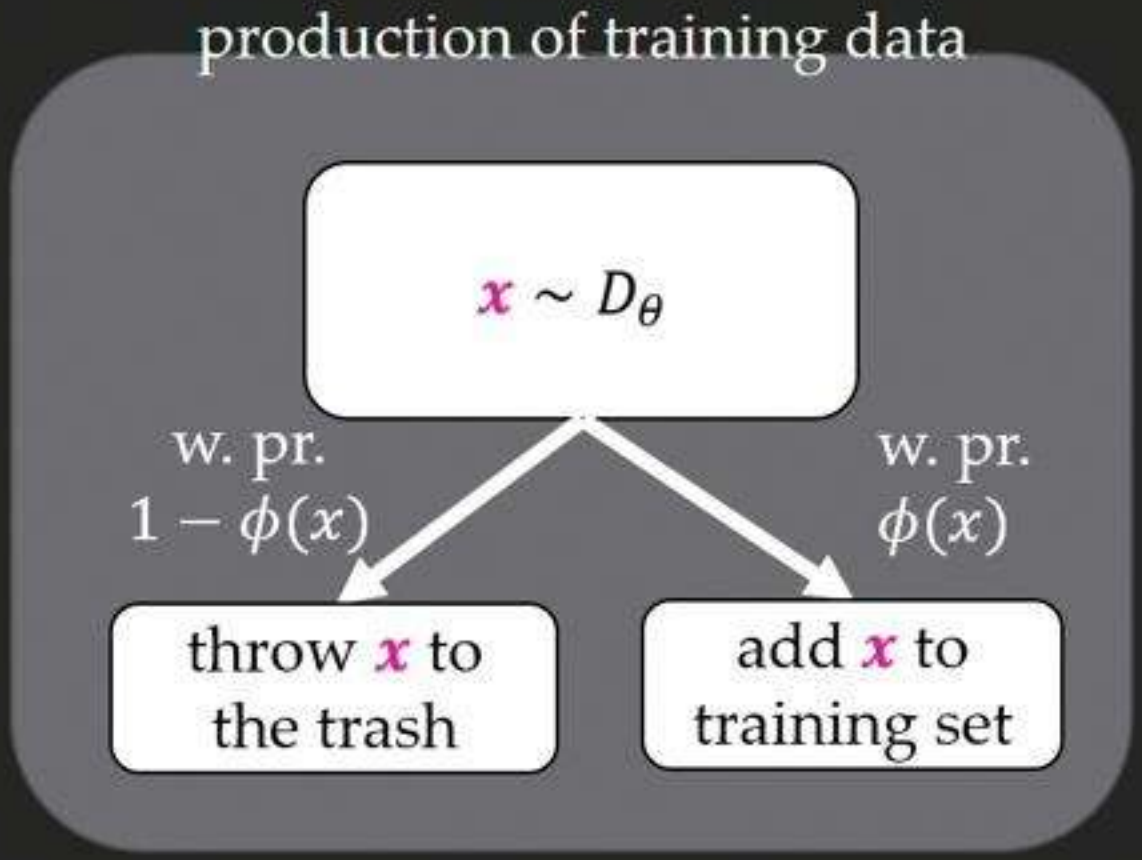


(a) A comparison of the accuracy of a classifier trained using our weighting method vs. a naively trained classifier on CelebA as a function of the training epochs.

Test classifier on a dataset

Problem 3: Truncated Density Estimation

Model:

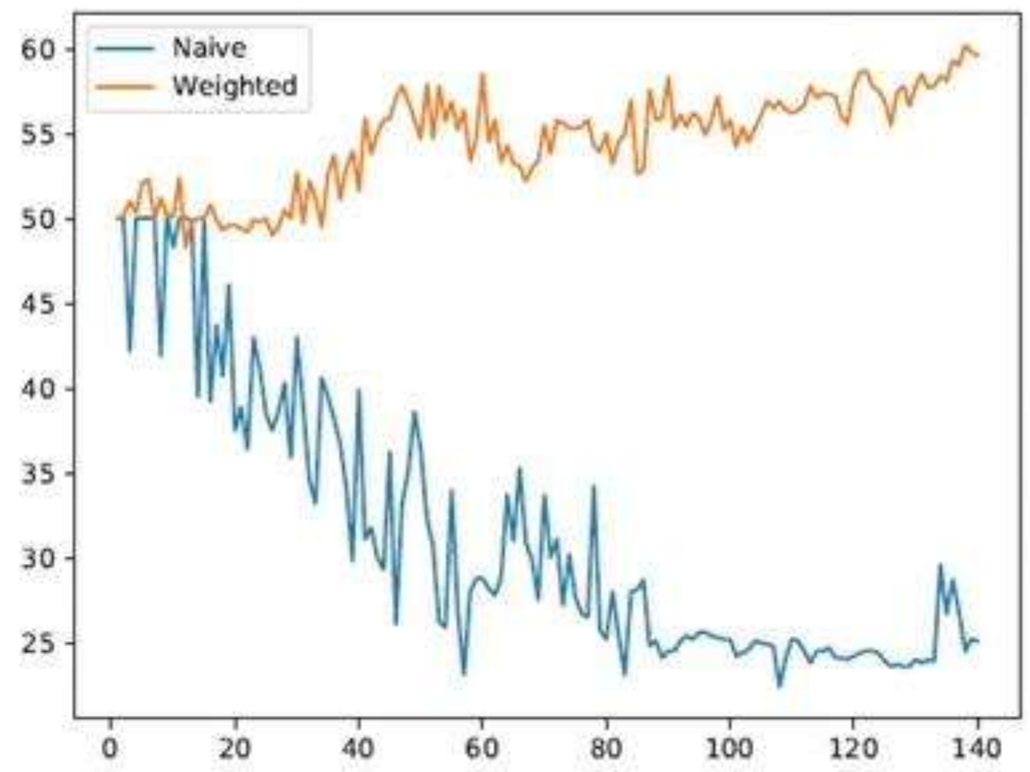


Example Application: Gender Classification 2

Train gender classi ... *balanced* training set of labeled male-femal ... tains images that a 95% accurate gender cla...

- use 1000 misclas ... orrectly classified males and females

Test classifier on a ... dataset

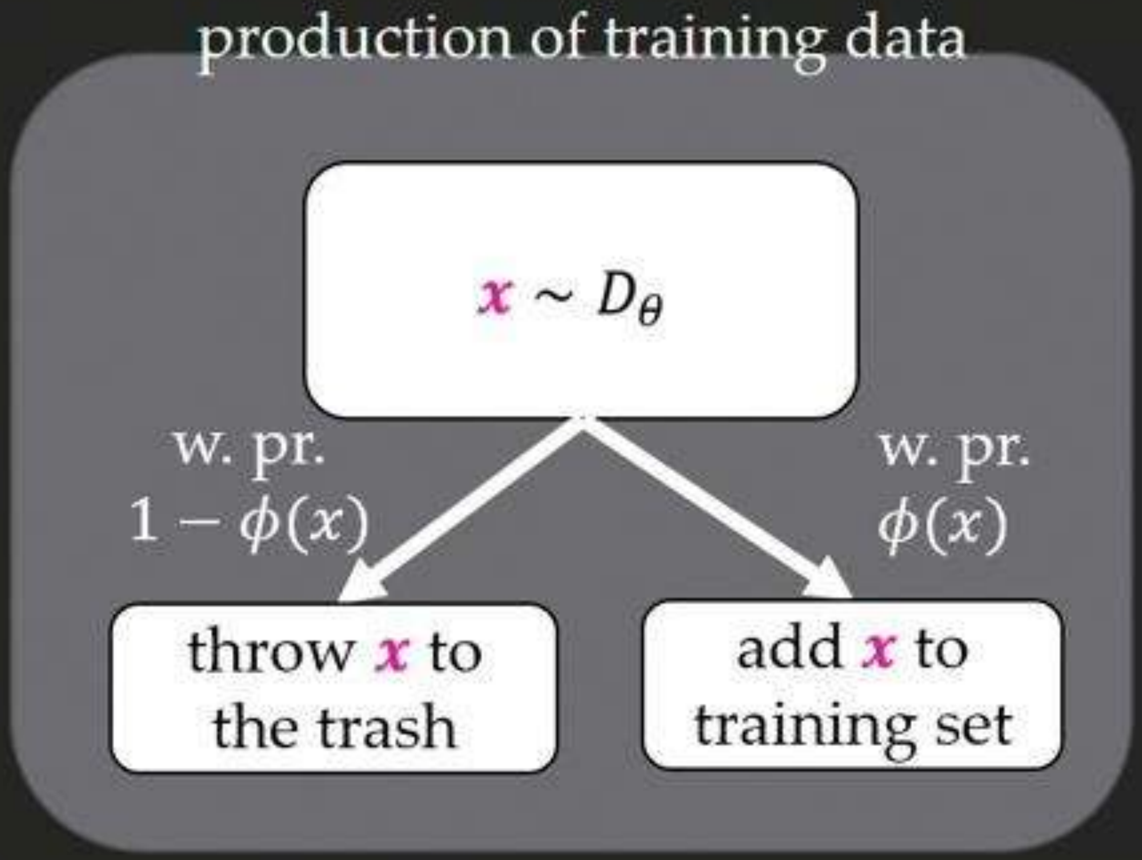


(a) A comparison of the accuracy of a classifier trained using our weighting method vs. a naively trained classifier on CelebA as a function of the training epochs.

Problem 3: Truncated Density Estimation

Model:

- (*unknown*) parametric distribution D_θ over $\mathbb{R}^d$
  - uncensored data-points are vectors $x \sim D_\theta$
- (*known*) filtering mechanism $\phi: \mathbb{R}^d \to [0,1]$
  - x included in train set with probability $\phi(x)$



Goal: given filtered data $(x_i)_i$ recover θ

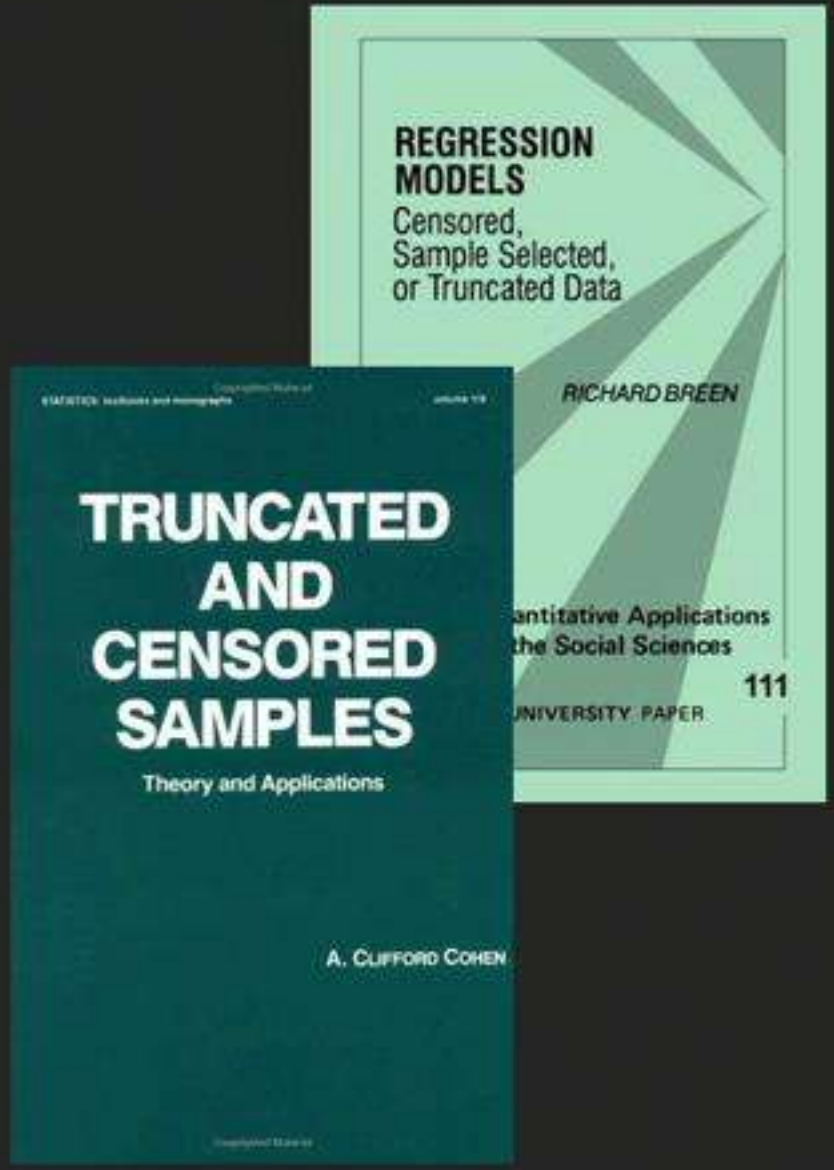
Results: practical SGD & MLE based framework **[w/ Ilyas, Zampetakis]**

- Fast rates + rigorous recovery of true parameters for Gaussians and other exponential families **[w/ Gouleakis, Tzamos, Zampetakis FOCS'18]**
- Unknown 0/1 filtering: **[Kontonis, Tzamos, Zampetakis FOCS'19]**

Comparison to Prior Work On Truncated Density Estimation

Learning Truncated/Censored Distributions

[Galton 1897], [Pearson 1902], [Pearson, Lee 1908], [Lee 1914], [Fisher 1931], [Hotelling 1948, [Tukey 1949],…,[Cohen'16]



Technical Bottlenecks:

- Convergence rates: $O_d\left(\frac{1}{\sqrt{n}}\right)$
- Computationally inefficient algorithms

Our work: optimal rates $O\left(\sqrt{\frac{\#\text{params}}{n}}\right)$, efficient algorithms, arbitrary truncation sets

Summary

- ❑ **Missing Observations**

  ⇒ **train set dist'n ≠ test set distribution**

  ⇒ **prediction bias (a.k.a. "AI bias")**

- ❑ **Our Work:** decrease bias, by developing machine learning methods more robust to **censored and truncated samples**
- ❑ **General Framework:** SGD on Population Log-Likelihood
- ❑ **End-to-end guarantees:** optimal rates and efficient algorithms for truncated Gaussian estimation, and truncated linear/logistic/probit regression

Summary

- **Missing Observations**
  - ⇒ **train set dist'n ≠ test set distribution**
  - ⇒ **prediction bias (a.k.a. "AI bias")**
- **Our Work:** decrease bias, by developing machine learning methods more robust to **censored and truncated samples**
- **General Framework:** SGD on Population Log-Likelihood
- **End-to-end guarantees:** optimal rates and efficient algorithms for truncated Gaussian estimation, and truncated linear/logistic/probit regression

Thank you!

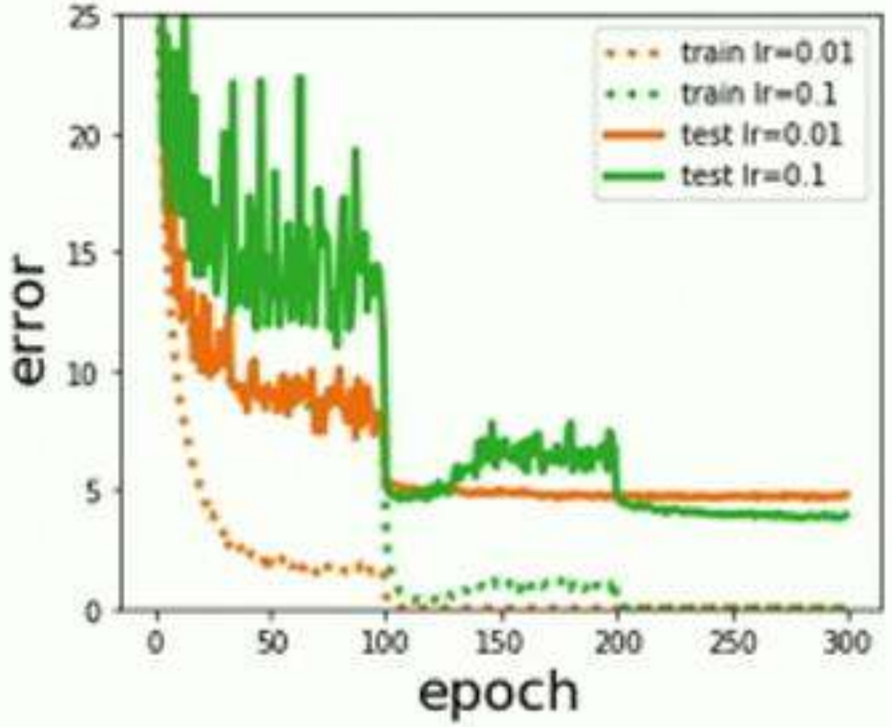
Towards Explaining the Regularization Effect of Initial Large Learning Rate

Yuanzhi Li* , Colin Wei *, Tengyu Ma

Stanford University

How do we design faster optimizers for deep learning?

Faster training is not that difficult: just use a smaller learning rate!



Algorithms can regularize!

The lack of understanding of the generalization hampers the study of optimization!

Why does large initial learning rate help generalization?

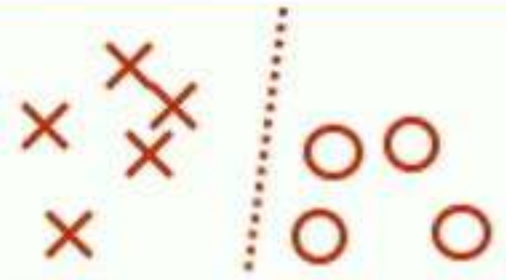
[Keskar et al'17, Hoffer et al'18]

Analysis Has to Be History-Sensitive --- The Initial Learning Rate Makes a Difference

- Linear models do not have this property
  - with regularization: strongly convex loss, unique minimizer
  - w/o regularization: the distribution of SGD iterates largely depends on the final learning rate
- Studying the limiting behavior of SGD (as $T \to \infty$) does not suffice
- NTK is almost a linear model with convex optimization in Kernel space
- This work: for a toy data distribution and two-layer neural nets, we show that SGD learns various patterns in different orders with different learning rate schedules, which results in different generalizations.

Two Types of Patterns

- Pattern 1: hard-to-generalize, easy-to-fit pattern
  - needs only simple model to fit
  - requires many samples (> dimension) to generalize

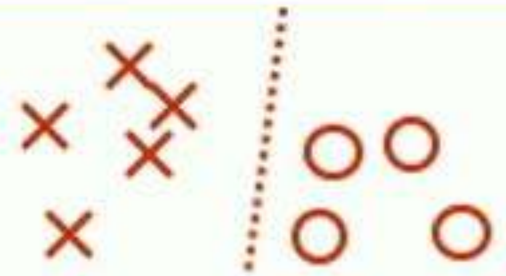


linearly classifiable patterns

Two Types of Patterns

- Pattern 1: hard-to-generalize, easy-to-fit pattern
  - needs only simple model to fit
  - requires many samples (> dimension) to generalize

- Pattern 2: easy-to-generalize, hard-to-fit patterns
  - requires complex models to fit
  - requires few samples to generalize



linearly classifiable patterns

clustered but not linearly separable

Learning Order Matters When Data Distribution is Heterogenous

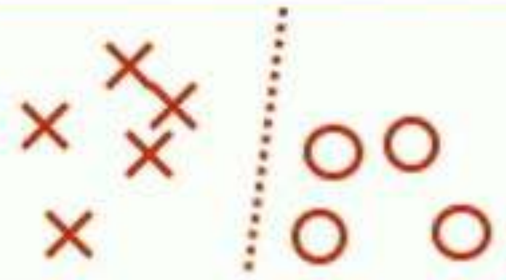
- A toy data distribution with mixed patterns
  - A datapoint $x = (x_1, x_2)$, $x_1 \in \mathbb{R}^d$, $x_2 \in \mathbb{R}^d$
  - Type I: 20% of examples = $(x_1, 0)$, $x_1 \sim$ pattern 1
  - Type II: 20% of examples = $(0, x_2)$, $x_2 \sim$ pattern 2
  - Type III: 60% of examples = (x_1, x_2)

linearly classifiable patterns

clustered but not linearly separable

Learning Order Matters When Data Distribution is Heterogenous

- A toy data distribution with mixed patterns
  - A datapoint $x = (x_1, x_2)$, $x_1 \in \mathbb{R}^d$, $x_2 \in \mathbb{R}^d$
  - Type I: 20% of examples $= (x_1, 0)$, $x_1 \sim$ pattern 1
  - Type II: 20% of examples $= (0, x_2)$, $x_2 \sim$ pattern 2
  - Type III: 60% of examples $= (x_1, x_2)$



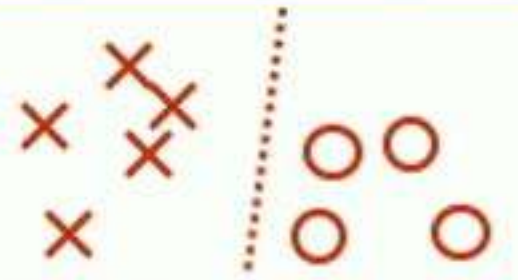
linearly classifiable patterns

clustered but not linearly separable

- Alg. 1:
  - First learns pattern 1: best linear fit to type I & III data
  - Then learns pattern 2: non-linear fit to type II data
- Alg. 2:
  - First learns pattern 2: non-linear fit to type II & III data
  - Then learns pattern 1: best linear fit to type I data

Learning Order Matters When Data Distribution is Heterogenous

- A toy data distribution with mixed patterns
  - A datapoint $x = (x_1, x_2)$, $x_1 \in \mathbb{R}^d$, $x_2 \in \mathbb{R}^d$
  - Type I: 20% of examples = $(x_1, 0)$, $x_1 \sim$ pattern 1
  - Type II: 20% of examples = $(0, x_2)$, $x_2 \sim$ pattern 2
  - Type III: 60% of examples = (x_1, x_2)



linearly classifiable patterns

clustered but not linearly separable

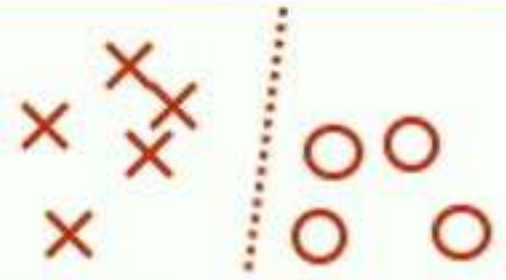
- Alg. 1:
  - First learns pattern 1: best linear fit to type I & III data
  - Then learns pattern 2: non-linear fit to type II data

80% of data, generalize better

- Alg. 2:
  - First learns pattern 2: non-linear fit to type II & III data
  - Then learns pattern 1: best linear fit to type I data

20% of data, generalize worse

Learning Order Matters When Data Distribution is Heterogenous



linearly classifiable patterns

clustered but not linearly separable

- A toy data distribution with mixed patterns
  - A datapoint $x = (x_1, x_2)$, $x_1 \in \mathbb{R}^d$, $x_2 \in \mathbb{R}^d$
  - Type I: 20% of examples $= (x_1, 0)$, $x_1 \sim$ pattern 1
  - Type II: 20% of examples $= (0, x_2)$, $x_2 \sim$ pattern 2
  - Type III: 60% of examples $= (x_1, x_2)$

- Alg. 1: large learning rate + annealing
  - First learns pattern 1: best linear fit to type I & III data → 80% of data, generalize better
  - Then learns pattern 2: non-linear fit to type II data
- Alg. 2: small learning rate
  - First learns pattern 2: non-linear fit to type II & III data
  - Then learns pattern 1: best linear fit to type I data → 20% of data, generalize worse
- (Note: generalization of pattern 2 is always good regardless the # samples used to learn it)

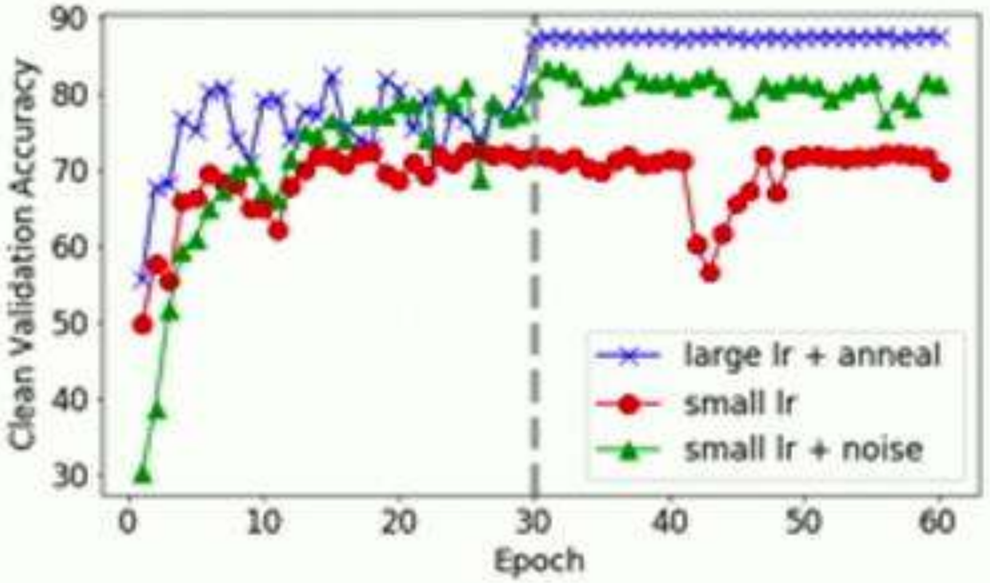
Learning Order Matters When Data Distribution is Heterogenous

History-dependency from logistic loss: once an example is fit with good confidence, it will not affect much the training later

- Alg. 1: large learning rate + annealing
  - First learns pattern 1: best linear fit to type I & III data
  - Then learns pattern 2: non-linear fit to type II data
- Alg. 2: small learning rate
  - First learns pattern 2: non-linear fit to type II & III data
  - Then learns pattern 1: best linear fit to type I data
- (Note: generalization of pattern 2 is always good regardless the # samples used to learn it)

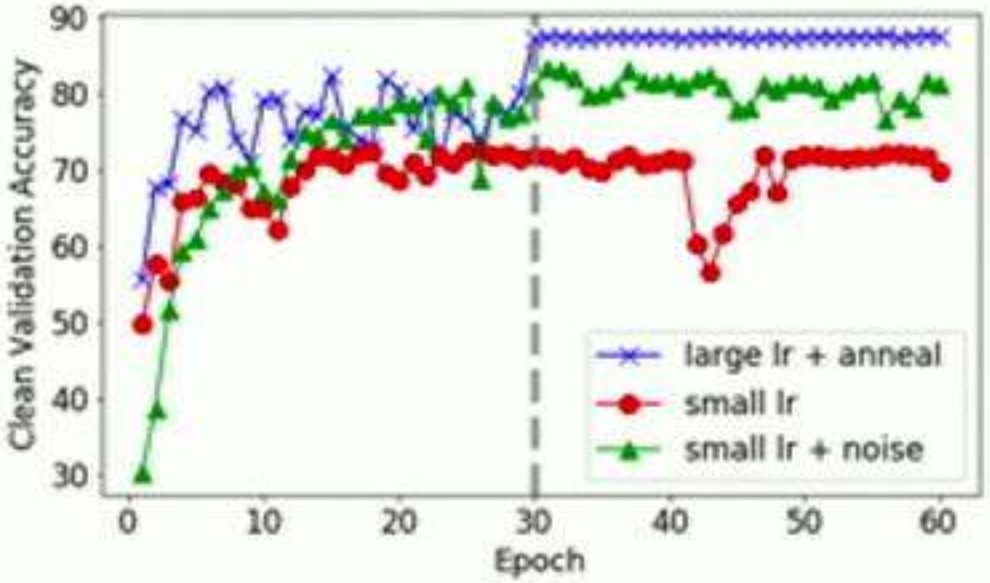
Interlude: Experiments on Artificial Datasets
Learning Orders with Synthetic Artificially Easy-to-Generalize Patterns

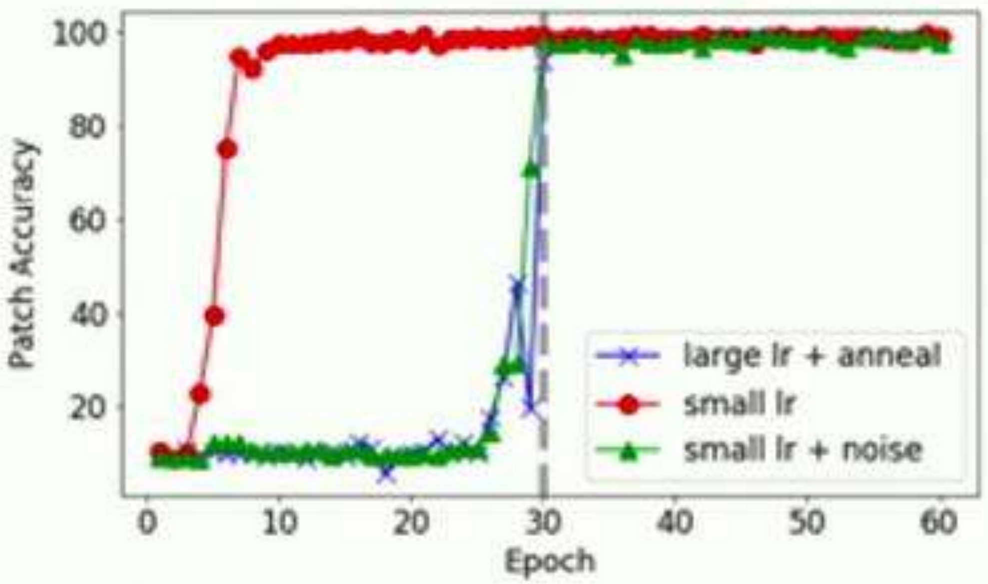
- Add easy-to-generalize patches to CIFAR images
  - Two patches v_i, v_i' for each class i
  - 20% of examples have no patch
  - 20% of examples have only patch
  - 60% of examples are mixed



Interlude: Experiments on Artificial Datasets
Learning Orders with Synthetic Artificially Easy-to-Generalize Patterns

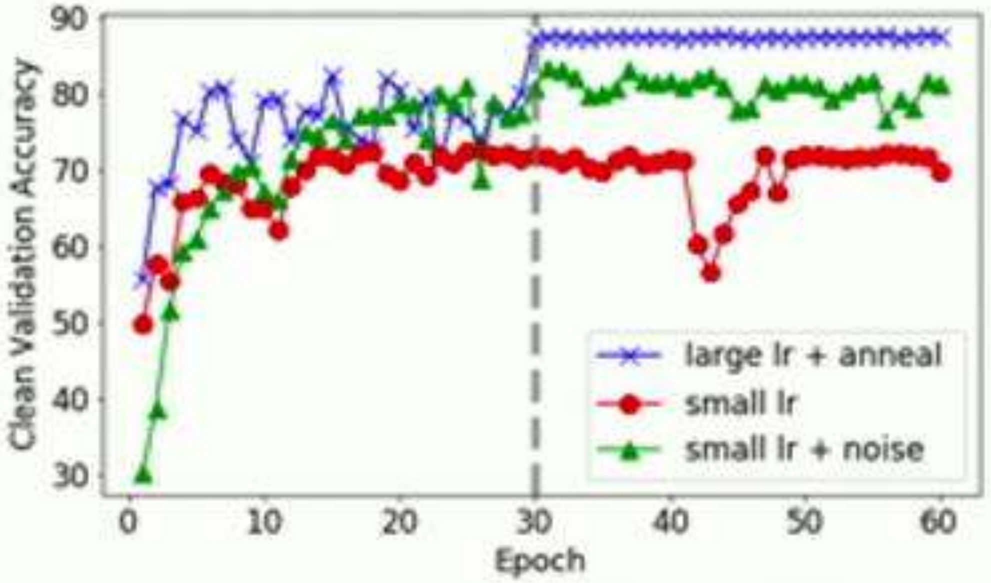
- Add easy-to-generalize patches to CIFAR images
  - Two patches v_i, v_i' for each class i
  - 20% of examples have no patch
  - 20% of examples have only patch
  - 60% of examples are mixed

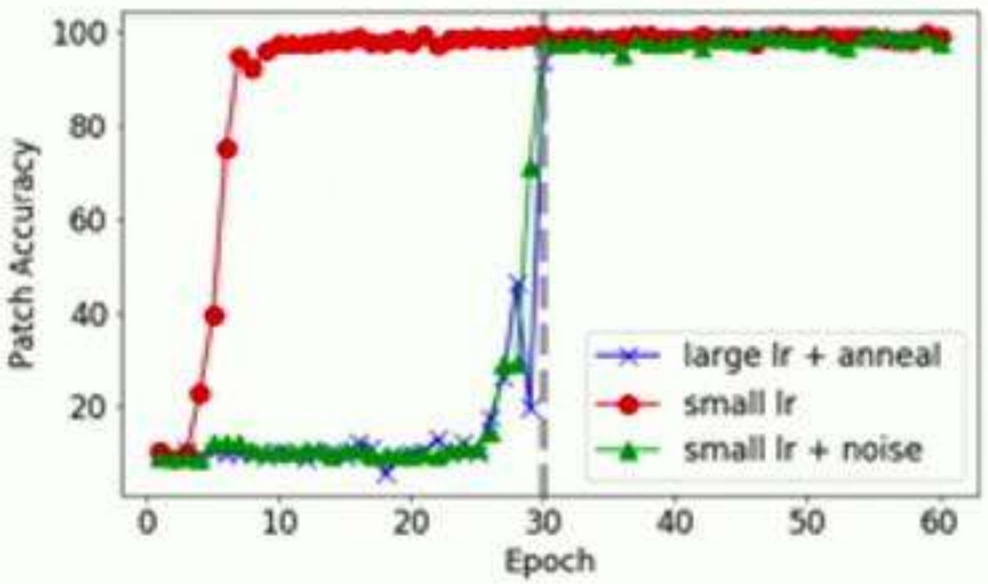




Interlude: Experiments on Artificial Datasets
Learning Orders with Synthetic Artificially Easy-to-Generalize Patterns

- Add easy-to-generalize patches to CIFAR images
  - Two patches v_i, v_i' for each class i
  - 20% of examples have no patch
  - 20% of examples have only patch
  - 60% of examples are mixed



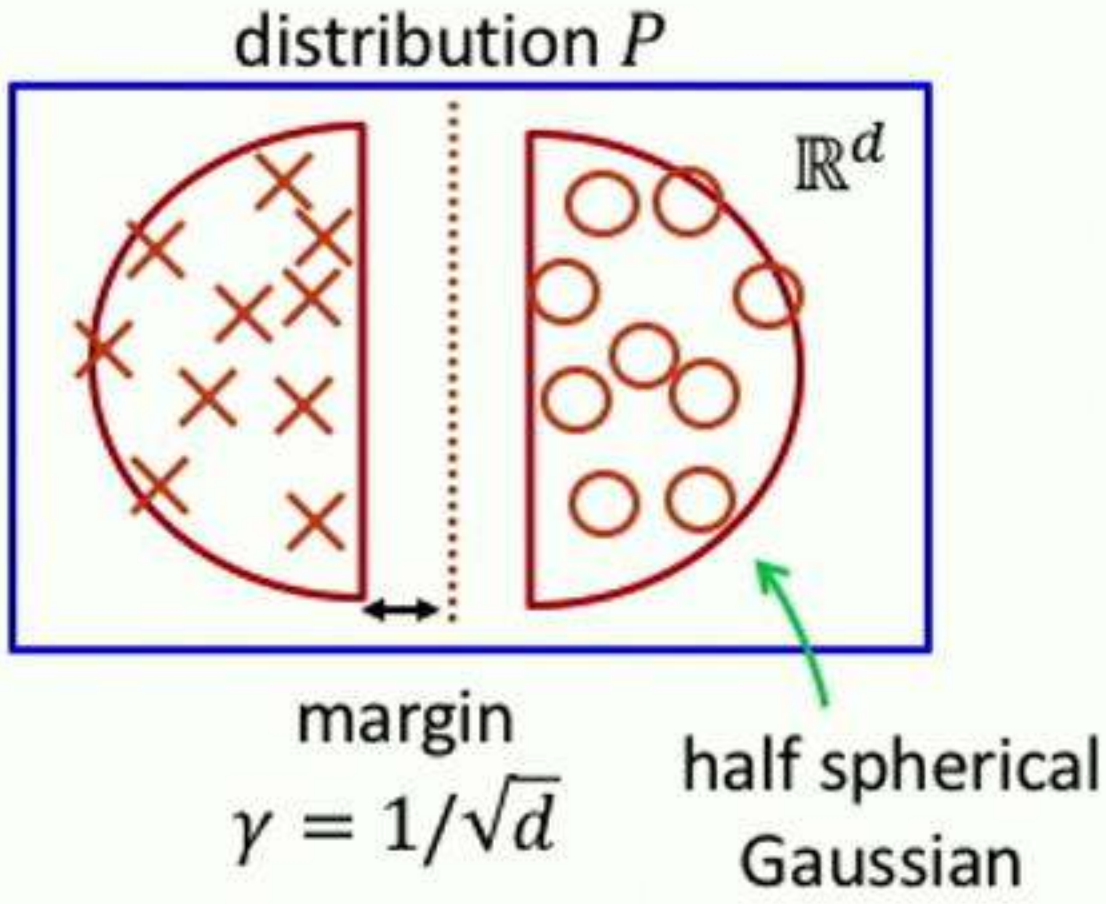


- Large learning rate doesn't learn the patches
- Learning the patches early hurts the generalization of clean images

A Toy Data Distribution with Theoretical Analysis

A datapoint $x = (x_1, x_2)$, $x_1 \in \mathbb{R}^d$, $x_2 \in \mathbb{R}^d$

- Type I: with prob. p, $x = (x_1, 0)$, $(x_1, y) \sim P$
- Type II: with prob. p, $x = (0, x_2)$, $(x_2, y) \sim Q$

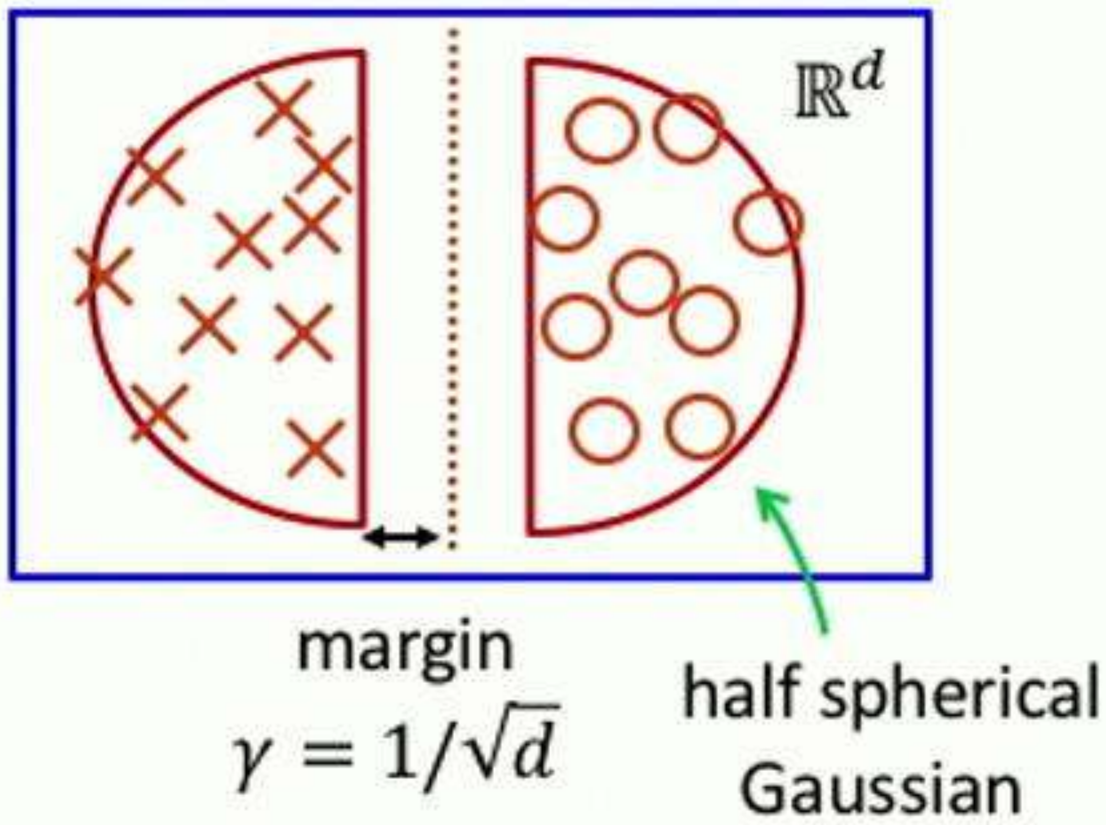


A Toy Data Distribution with Theoretical Analysis

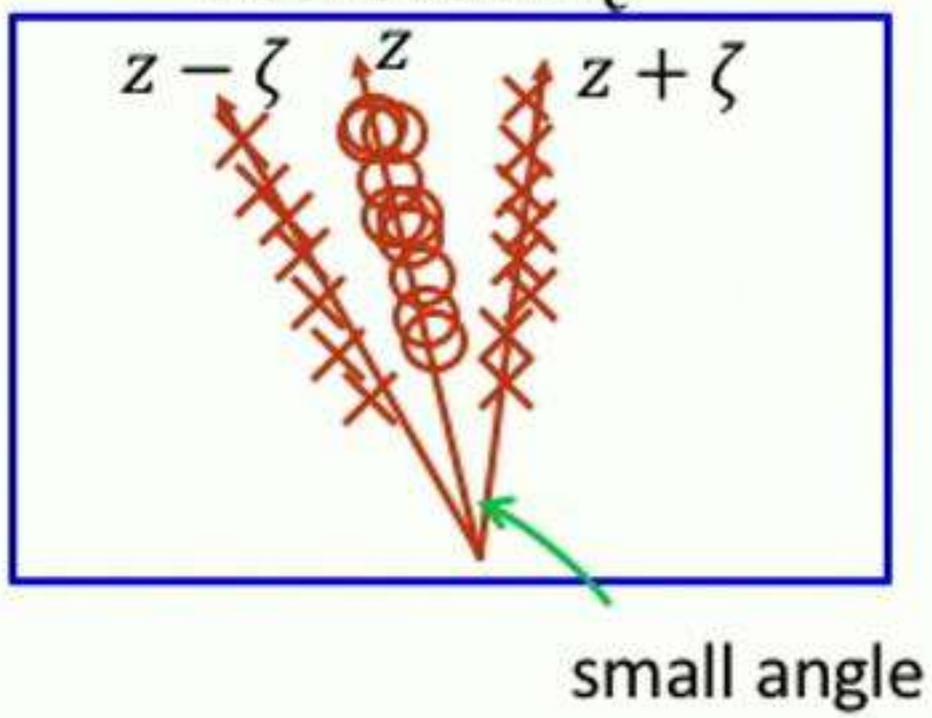
A datapoint $x = (x_1, x_2)$, $x_1 \in \mathbb{R}^d$, $x_2 \in \mathbb{R}^d$

- Type I: with prob. p, $x = (x_1, 0)$, $(x_1, y) \sim P$
- Type II: with prob. p, $x = (0, x_2)$, $(x_2, y) \sim Q$
- Type III: with prob. $1 - 2p$, $x = (x_1, x_2)$, $(x_1, y) \sim P, (x_2, y) \sim Q$

distribution P



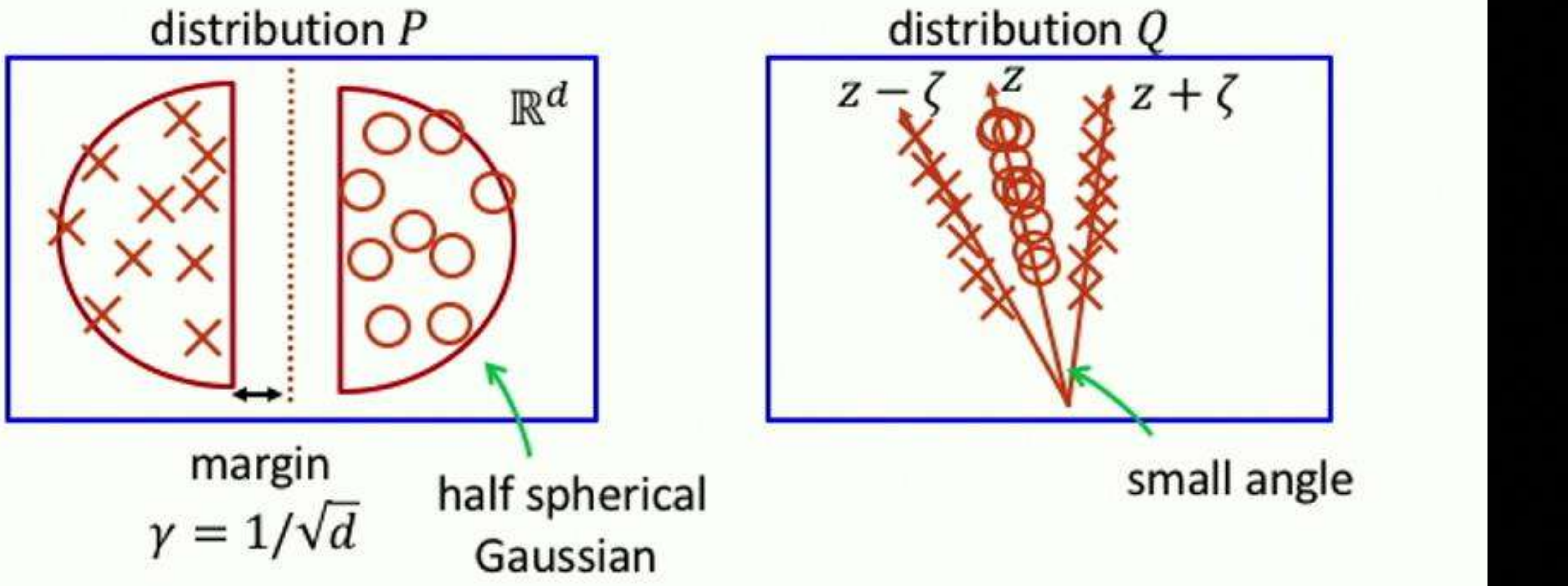
distribution Q



A Toy Data Distribution with Theoretical Analysis

Not linear separable

Classifiable by two layer neural nets



Main Theoretical Statement (Informal)

- Model: $f(x) = w^{\top}\mathrm{relu}(Wx_1) + v^{\top}\mathrm{relu}(Vx_2)$ (with wide hidden layer)
- Loss: regularized cross-entropy loss
- Algorithm: gradient descent with spherical Gaussian noise
- A lot of other assumptions on the hyperparameters

- Both algorithms learn pattern 2, and generalize for pattern 2

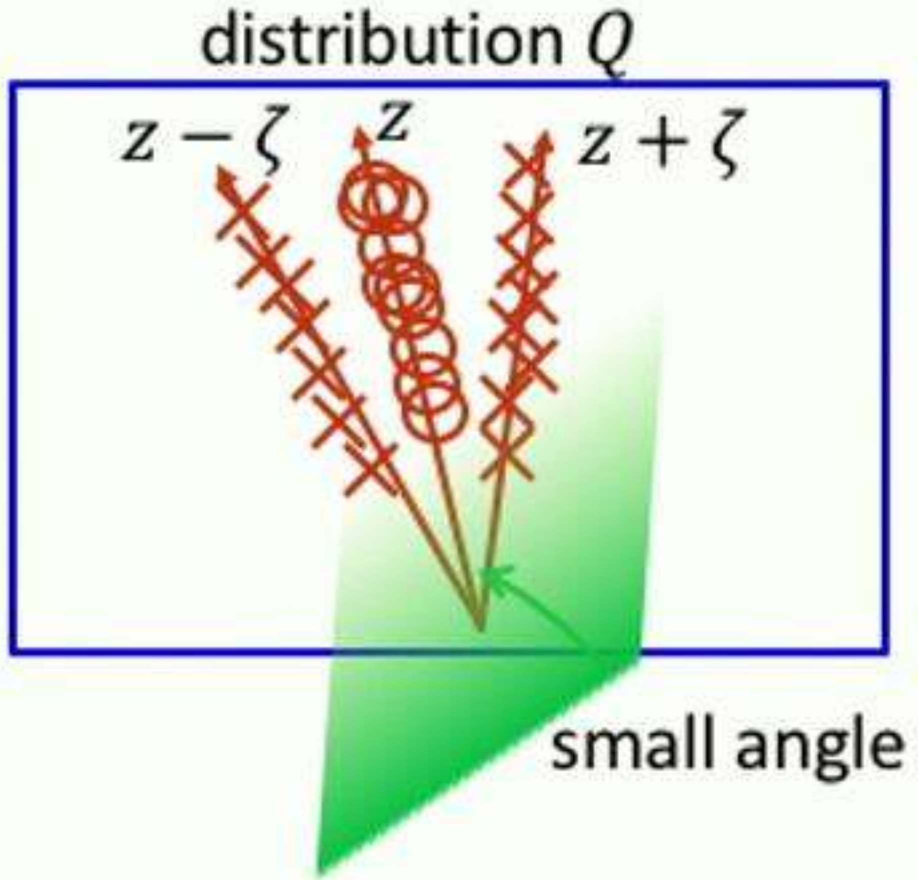
Main Theoretical Statement (Informal)

- Model: $f(x) = w^\top \mathrm{relu}(Wx_1) + v^\top \mathrm{relu}(Vx_2)$ (with wide hidden layer)
- Loss: regularized cross-entropy loss
- Algorithm: gradient descent with spherical Gaussian noise
- A lot of other assumptions on the hyperparameters

- Both algorithms learn pattern 2, and generalize for pattern 2
- Alg. 1 (large learning rate + annealing) learns pattern 1 first utilizing $(1 - p)$ fraction of data; generalization error $\lesssim \sqrt{d/((1-p)n)}$
- Alg. 2 (small learning rate throughout) learns pattern 1 after pattern 2 is learned, and thus only utilizes p fraction of data. When $pn \leq d/2$, no generalization for pattern 1.

Basic Intuitions

- Small learning rate: the noise from SGD is small (essentially NTK regime)
- Large learning rate: the noise is big; weights change a lot
  - Unless a weight vector w defines a hyperplane that separates $z-\zeta$ and $z+\zeta$, the neuron $\mathrm{relu}(w^\top x)$ behaves as a linear function
    $$\mathrm{relu}(w^\top(z-\zeta)) - 2\mathrm{relu}(w^\top z) + \mathrm{relu}(w^\top(z+\zeta)) = 0$$
  - Network behaves like linear functions on distribution Q



More Technical Intuitions

- Decomposition of weight matrix U under SGD

$$U_t = \underbrace{\overline{U}_t}_{\text{cumulative contribution of the full gradient}} + \underbrace{\widetilde{U}_t}_{\text{cumulative contribution of randomness (= the noises and initialization)}}$$

- Large learning rate: $\widetilde{U}_t$ changes fast
- Small learning rate: $\widetilde{U}_t$ changes slowly
- (This is true even if we re-scale the timescale to take into account that smaller learning rate trains slower)

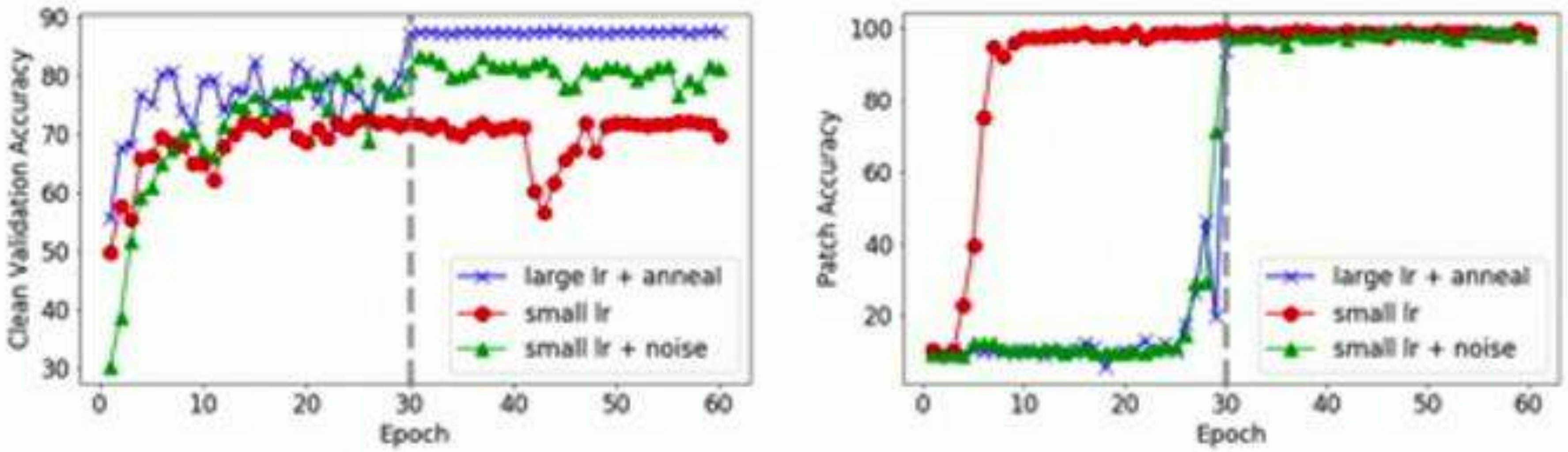
More Technical Intuitions (Cont'd)

- Neural network "Taylor expansion"

$$\begin{aligned}
\mathrm{relu}(U_t x) &= 1(U_t x) \odot U_t x \\
&= 1(U_t x) \odot \bar{U}_t x + \underbrace{1(U_t x) \odot \widetilde{U}_t x}_{\approx 0, \text{cancellation due to } \widetilde{U}_t x} \\
&\approx 1(\widetilde{U}_t x) \odot \bar{U}_t x, \quad \text{up to } o(\|\bar{U}_t\|)
\end{aligned}$$

Mitigation Strategy

- Theory suggests that large learning rate injects larger noises in activation patterns, which helps avoid learning the easy-to-generalize pattern
- Empirical strategy: add pre-activation noises



- Also helps in training clean data with small learning rate

Table 1: Validation accuracies for WideResNet16 trained and tested on original CIFAR-10 images without data augmentation.

Method	Val. Acc
Large LR + anneal	90.41%
Small LR + noise	89.65%
Small LR	84.93%

Wrap Up

- Large learning rate learns hard-to-generalize, easy-to-fit pattern
- Small learning rate learns easy-to-generalize, hard-to-fit patterns
- How do we identify these patterns in real data?
- Can we make the result more general (instead of only on a contrived toy example)?

Some Broader Outlook

- Algorithmic/implicit regularization could be very challenging/subtle to understand and manipulate
  - Not clear what complexity measure the algorithm is regularizing in our toy case
- Shortcut: could we find explicit regularizers that subsume the algorithmic/implicit regularization?
  - Data-dependent regularization is promising [Wei-M.'19]
- Heterogeneous datasets are likely where the interesting phenomenon occurs, and where practical improvements are easier
  - Standard datasets are well-tuned for algorithmic regularization
  - [Cao-Wei-Gaidon-Arechiga-M.'19] explicit regularization for imbalanced datasets: very simple theory with good empirical performance

Some Broader Outlook

- Algorithmic/implicit regularization could be very challenging/subtle to understand and manipulate
  - Not clear what complexity measure the algorithm is regularizing in our toy case
- Shortcut: could we find explicit regularizers that subsume the algorithmic/implicit regularization?
  - Data-dependent regularization is promising [Wei-M.'19]
- Heterogeneous datasets are likely where the interesting phenomenon occurs, and where practical improvements are easier
  - Standard datasets are well-tuned for algorithmic regularization
  - [Cao-Wei-Gaidon-Arechiga-M.'19] explicit regularization for imbalanced datasets: very simple theory with good empirical performance

Thank you!