10-605/805 – ML for Large Datasets
Lecture 5: Distributed Linear Regression

Henry Chai

9/13/22

Front Matter

- HW1 released 8/30, due 9/13 (today!) at 11:59 PM
  - Recitation 3 on 9/16 will go over HW1 solutions
- HW2 released 9/8, due 9/22 at 11:59 PM
- Mini-project details released 9/9

Background: Big O Notation

- Used to describe an algorithm's time or space (storage) complexity in terms of the input size
- Formally:

$$f(x) = O\left(g(x)\right) \Leftrightarrow \exists\, C, x_0 \text{ s.t. } f(x) \leq Cg(x)\ \forall\, x \geq x_0$$

  - $O(1) =$ constant time/space, i.e., a fixed number of operations or storage regardless of input
  - $O(\log(n)) =$ logarithmic time/space
  - $O(n) =$ linear time/space
- An algorithm's time and space complexity can be different
  - Example: multiplying an $a \times b$ matrix with an $b \times c$ matrix takes $O(abc)$ time (ac dot products between b-length vectors) but the result uses $O(ac)$ storage

Background: Empirical Risk Minimization

- A common framework for supervised learning
- Given:
  - some labelled training dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$
  - a loss function $\ell: \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$
  - a hypothesis class or set of functions $\mathcal{F}$

the goal is to find

$$\hat{f} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{i=1}^{n} \ell\left(f\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right)$$

with the hope that

$$\mathbb{E}_{p(\boldsymbol{x}, y)}[\ell(f(\boldsymbol{x}), y)] \approx \sum_{i=1}^{n} \ell\left(f\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right)$$

Background: Empirical Risk Minimization

- A common framework for supervised learning
- Given:
  - some labelled training dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$
  - a loss function $\ell: \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$
  - a hypothesis class or set of functions $\mathcal{F}$

  the goal is to find

$$\hat{f} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{i=1}^{n} \ell\left(f\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right)$$

- Depending on the choice of $\mathcal{F}$ and ℓ, this objective function may be convex (easy to optimize) or non-convex (hard)
- Our focus will be solving this problem for large n and/or k

Background: Regression

- A type of supervised learning
- Given:
  - some labelled training dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$
  - a loss function $\ell: \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$ where $\mathcal{Y} = \mathbb{R}$
  - a hypothesis class or set of functions $\mathcal{F}$

the goal is to find

$$\hat{f} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{i=1}^{n} \ell\left(f\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right)$$

- Fun example: predicting the year a song was released based on (a representation of) its audio (see HW2)

Background: Linear Regression (Ordinary Least Squares)

- A type of supervised learning
- Given:
  - some labelled training dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$
  - $\ell(y, y') = (y - y')^2$
  - $\mathcal{F}$ = all functions of the form $f(\boldsymbol{x}) = w_0 + \sum_{d=1}^{k} w_d x_d$

the goal is to find

$$\hat{f} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(f\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2$$

Background: Linear Regression (Ordinary Least Squares)

- A type of supervised learning
- Given:
  - some labelled training dataset $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$
  - $\ell(y, y') = (y - y')^2$ $\qquad [w_0, w_1, \ldots, w_k]$
  - $\mathcal{F}$ = all functions of the form $f(\boldsymbol{x}) = \boldsymbol{w}^T[1\ \boldsymbol{x}^T]^T$

the goal is to find

$$\hat{f} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(f\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2$$

Background: Linear Regression (Ordinary Least Squares)

- A type of supervised learning
- Given:
  - some labelled training dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$
  - $\ell(y, y') = (y - y')^2$
  - $\mathcal{F}$ = all functions of the form $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$

1 implicitly prepended

the goal is to find

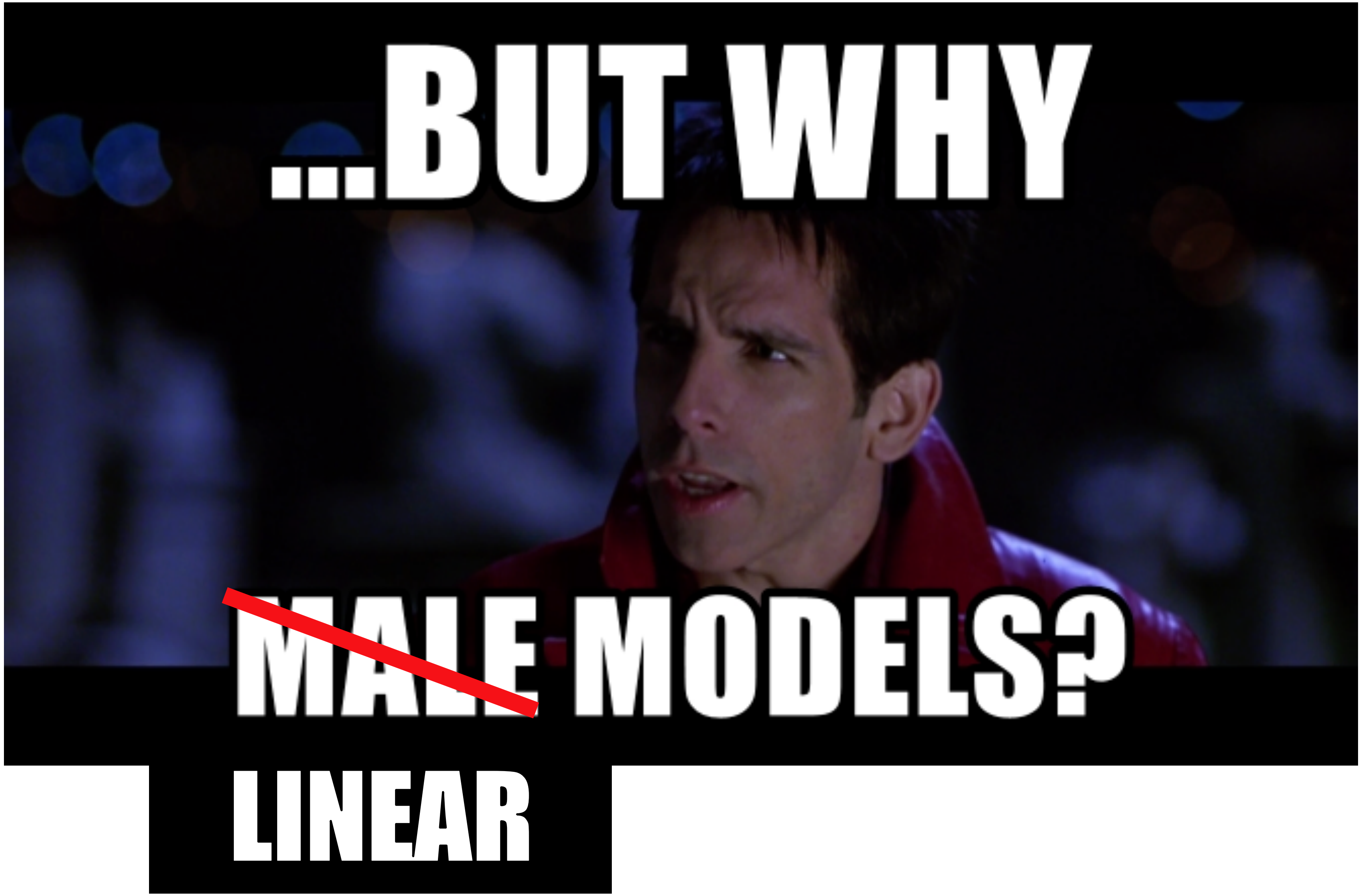
$$\widehat{\boldsymbol{w}} = \underbrace{\underset{\boldsymbol{w}}{\operatorname{argmin}}} \sum_{i=1}^{n} \left(\boldsymbol{w}^T\boldsymbol{x}^{(i)} - y^{(i)}\right)^2$$

Interpretable

Very easy to optimize
- exact solution

Simple - tend to generalize better

Fast inference



Background: Linear Regression (Ordinary Least Squares)

- A type of supervised learning
- Given:
  - some labelled training dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$
  - $\ell(y, y') = (y - y')^2$
  - $\mathcal{F}$ = all functions of the form $f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$

1 implicitly prepended

the goal is to find

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(\boldsymbol{w}^T \boldsymbol{x}^{(i)} - y^{(i)}\right)^2$$

Background: Linear Regression (Ordinary Least Squares)

- A type of supervised learning
- Given:
  - some labelled training dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$
  - $\ell(y, y') = (y - y')^2$
  - $\mathcal{F}$ = all functions of the form $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$

1 implicitly prepended

the goal is to find

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \ (X\boldsymbol{w} - \boldsymbol{y})^T(X\boldsymbol{w} - \boldsymbol{y})$$

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \ \underline{\|X\boldsymbol{w} - \boldsymbol{y}\|_2^2}$$

- where $X = \begin{bmatrix} \boldsymbol{x}^{(1)^T} \\ \vdots \\ \boldsymbol{x}^{(n)^T} \end{bmatrix}$ and $\boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}$

Background: Linear Regression (Ordinary Least Squares)

$$L_{\mathcal{D}}(\boldsymbol{w}) = (X\boldsymbol{w} - \boldsymbol{y})^T (X\boldsymbol{w} - \boldsymbol{y})$$

$$= w^T X^T X w - 2 w^T X^T y + y^T y$$

$$\nabla_w L_D(w) = 2X^T X w - 2X^T y + 0$$

$$2X^T X \hat{w} - 2X^T y = 0$$

$$\Rightarrow 2X^T X \hat{w} = 2X^T y$$

$$\Rightarrow \hat{w} = (X^T X)^{-1} X^T y$$

$H_w L_D(w) = 2X^T X$ which is positive semi-definite everywhere

Background: Regularization

- A modification to empirical risk minimization that penalizes model complexity in order to combat overfitting
- Given:
  - some labelled training dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$
  - a loss function $\ell: \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$
  - a hypothesis class or set of functions $\mathcal{F}$
  - a regularizer $R: \mathcal{W} \to \mathbb{R}$
  - a coefficient of regularization λ

the goal is to find

$$\widehat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_{i=1}^{n} \ell\left(f_{\boldsymbol{w}}\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right) + \lambda R(\boldsymbol{w})$$

Background: Ridge Regression

- A modification to empirical risk minimization that penalizes model complexity in order to combat overfitting
- Given:
  - some labelled training dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$
  - $\ell(y, y') = (y - y')^2$
  - $\mathcal{F}$ = all functions of the form $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$
  - $R(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2 = \boldsymbol{w}^T\boldsymbol{w}$
  - a coefficient of regularization λ

the goal is to find

$$\widehat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\ (X\boldsymbol{w} - \boldsymbol{y})^T(X\boldsymbol{w} - \boldsymbol{y}) + \lambda\boldsymbol{w}^T\boldsymbol{w}$$

Background: Ridge Regression

$$L_{\mathcal{D}}(\boldsymbol{w}) = (X\boldsymbol{w} - \boldsymbol{y})^T (X\boldsymbol{w} - \boldsymbol{y}) + \lambda \boldsymbol{w}^T \boldsymbol{w}$$

$$\vdots$$

$$\rightarrow \hat{\boldsymbol{w}} = (X^T X + \lambda I_k)^{-1} X^T \boldsymbol{y}$$

where I_k is the $k \times k$ identity matrix

Aside: How can we set λ?

$$L_{\mathcal{D}}(\boldsymbol{w}) = (X\boldsymbol{w} - \boldsymbol{y})^T (X\boldsymbol{w} - \boldsymbol{y}) + \lambda \boldsymbol{w}^T \boldsymbol{w}$$

$$\vdots$$

$$\rightarrow \hat{\boldsymbol{w}} = (X^T X + \lambda I_k)^{-1} X^T \boldsymbol{y}$$

where I_k is the $k \times k$ identity matrix

$(\lambda > 0)$

Recall: Machine Learning Pipeline

- Hyperparameter optimization

- Suppose we want to compare multiple hyperparameter settings $\theta_1, \dots, \theta_C$
  (where do these come from???)
- For $c = 1, 2, \dots, C$
  - Train a model on D_{train} using θ_c
- Evaluate each model on D_{val} and find the best hyperparameter setting, θ_{c^*}
- Compute the error of a model trained with θ_{c^*} on D_{test}

HW2 Preview: Grid Search

- Hyperparameter optimization

- Pick C evenly spaced values over some prespecified range $\lambda_1, \dots, \lambda_C$
  e.g., $\{10^{-8}, 10^{-6}, 10^{-4}, 10^{-2}, 10^{0}\}$
- For $c = 1, 2, \dots, C$
  - Train a model on D_{train} using λ_c
- Evaluate each model on D_{val} and find the best hyperparameter setting, λ_{c^*}
- Compute the error of a model trained with λ_{c^*} on D_{test}

Linear Regression: Computational Cost

$\hat{\boldsymbol{w}} = (X^T X)^{-1} X^T \boldsymbol{y}$

1. Does this quantity exist, i.e., is $X^T X$ invertible?

For the most part yes, $X^T X$ is positive semi-definite; regularization helps

2. If so, how expensive is it to compute?

$X \in \mathbb{R}^{n \times k} \Rightarrow X^T X \in \mathbb{R}^{k \times k}$

Time cost of inverting $X^T X = O(k^3)$

Space cost of inverting $X^T X = O(k^2)$

Linear Regression: Large n, Small k

- Assume $O(k^3)$ computation and $O(k^2)$ storage is possible on a single machine
  - ✔ We can store and invert $X^T X$
  - We cannot compute $X^T X$
  - We cannot store X

  on a single machine

- Idea: distribute storage of X and computation of $X^T X$
  1. Store the rows of X across different machines
  2. Compute $X^T X$ as the sum of outer products

Matrix Multiplication via Inner Products

$$\begin{bmatrix} \boxed{1 & 4 & 5} \\ 3 & 1 & 3 \end{bmatrix} \begin{bmatrix} 2 & 2 \\ 2 & 1 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} & \end{bmatrix}$$

$$1 * 2 + 4 * 2 + 5 * 3 = 25$$

Matrix Multiplication via Inner Products

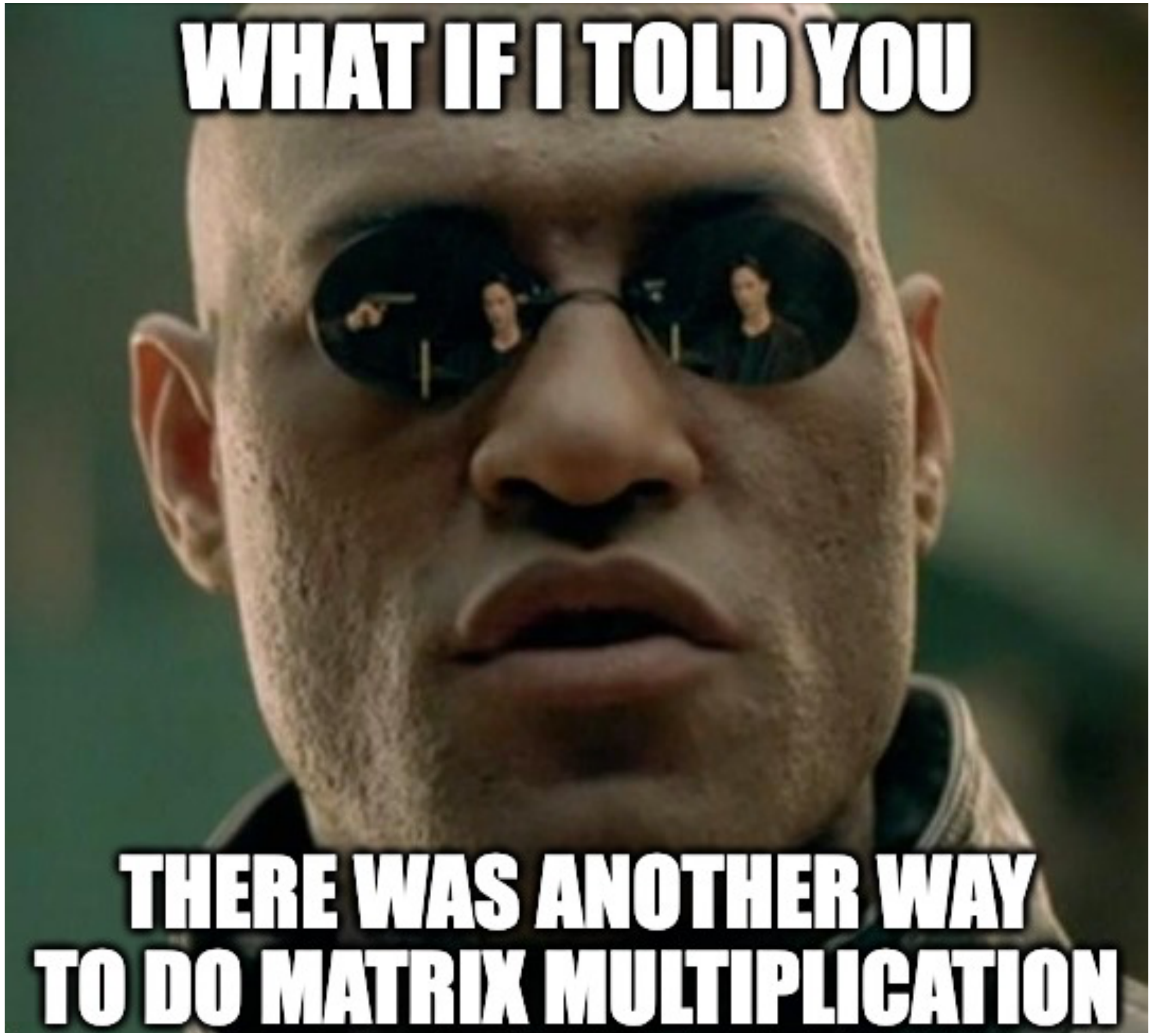
$$\begin{bmatrix} \boxed{1 \quad 4 \quad 5} \\ 3 \quad 1 \quad 3 \end{bmatrix} \begin{bmatrix} 2 & \boxed{\begin{matrix} 2 \\ 1 \\ 4 \end{matrix}} \\ 2 & \\ 3 & \end{bmatrix} = \begin{bmatrix} 25 & \\ & \end{bmatrix}$$

$$1 * 2 + 4 * 1 + 5 * 4 = 26$$

Matrix Multiplication via Inner Products

$$\begin{bmatrix} 1 & 4 & 5 \\ 3 & 1 & 3 \end{bmatrix} \begin{bmatrix} 2 & 2 \\ 2 & 1 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 25 & 26 \\ 17 & 19 \end{bmatrix}$$

Matrix Multiplication via Inner Products



Matrix Multiplication via Outer Products

$$\begin{bmatrix}1 & 4 & 5\\ 3 & 1 & 3\end{bmatrix}\begin{bmatrix}2 & 2\\ 2 & 1\\ 3 & 4\end{bmatrix} = \begin{bmatrix} \quad & \quad \end{bmatrix}$$

$$\begin{bmatrix}2 & 2\\ 6 & 6\end{bmatrix}$$

Matrix Multiplication via Outer Products

$$\begin{bmatrix} 1 & \boxed{4} & 5 \\ 3 & \boxed{1} & 3 \end{bmatrix} \begin{bmatrix} 2 & 2 \\ \boxed{2 \quad 1} \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} & \end{bmatrix}$$

$$\begin{bmatrix} 2 & 2 \\ 6 & 6 \end{bmatrix} + \begin{bmatrix} 8 & 4 \\ 2 & 1 \end{bmatrix}$$

Matrix Multiplication via Outer Products

$$\begin{bmatrix} 1 & 4 & \boxed{5} \\ 3 & 1 & \boxed{3} \end{bmatrix} \begin{bmatrix} 2 & 2 \\ 2 & 1 \\ \boxed{3} & \boxed{4} \end{bmatrix} = \begin{bmatrix} & \end{bmatrix}$$

$$\begin{bmatrix} 2 & 2 \\ 6 & 6 \end{bmatrix} + \begin{bmatrix} 8 & 4 \\ 2 & 1 \end{bmatrix} + \begin{bmatrix} 15 & 20 \\ 9 & 12 \end{bmatrix}$$

Matrix Multiplication via Outer Products

$$\begin{bmatrix} 1 & 4 & 5 \\ 3 & 1 & 3 \end{bmatrix} \begin{bmatrix} 2 & 2 \\ 2 & 1 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 25 & 26 \\ 17 & 19 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 2 \\ 6 & 6 \end{bmatrix} + \begin{bmatrix} 8 & 4 \\ 2 & 1 \end{bmatrix} + \begin{bmatrix} 15 & 20 \\ 9 & 12 \end{bmatrix}$$

Matrix Multiplication via Outer Products

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{bmatrix} \begin{bmatrix} B_{11} & \cdots & B_{1k} \\ B_{21} & \cdots & B_{2k} \\ B_{31} & \cdots & B_{3k} \\ \vdots & \ddots & \vdots \\ B_{m1} & \cdots & B_{mk} \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{i=1}^{m} A_{1i} B_{i1} & \cdots & \sum_{i=1}^{m} A_{1i} B_{ik} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^{m} A_{ni} B_{i1} & \cdots & \sum_{i=1}^{m} A_{ni} B_{ik} \end{bmatrix} = \sum_{i=1}^{m} \begin{bmatrix} A_{1i} B_{i1} & \cdots & A_{1i} B_{ik} \\ \vdots & \ddots & \vdots \\ A_{ni} B_{i1} & \cdots & A_{ni} B_{ik} \end{bmatrix}$$

Distributed Computation of $(X^T X)^{-1}$

$$X^T X = \begin{bmatrix} \uparrow & \uparrow & \cdots & \uparrow \\ \boldsymbol{x}^{(1)} & \boldsymbol{x}^{(2)} & \cdots & \boldsymbol{x}^{(n)} \\ \downarrow & \downarrow & \cdots & \downarrow \end{bmatrix} \begin{bmatrix} \leftarrow & \boldsymbol{x}^{(1)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(2)^T} & \rightarrow \\ \vdots & \vdots & \vdots \\ \leftarrow & \boldsymbol{x}^{(n)^T} & \rightarrow \end{bmatrix} = \sum_{i=1}^{n} \boldsymbol{x}^{(i)} \boldsymbol{x}^{(i)^T}$$

- Idea: distribute $\boldsymbol{x}^{(i)}$ and compute summands in parallel

Workers	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(1)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(4)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(2)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(3)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(5)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(7)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$

Distributed Computation of $(X^T X)^{-1}$

Workers	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(1)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(4)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(2)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(3)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(5)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(7)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$
Map	$\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)^T}$	$\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)^T}$	$\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)^T}$

Distributed Computation of $(X^T X)^{-1}$

Workers	$\begin{bmatrix} \leftarrow & {\boldsymbol{x}^{(1)}}^T & \rightarrow \\ \leftarrow & {\boldsymbol{x}^{(4)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & {\boldsymbol{x}^{(2)}}^T & \rightarrow \\ \leftarrow & {\boldsymbol{x}^{(3)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & {\boldsymbol{x}^{(5)}}^T & \rightarrow \\ \leftarrow & {\boldsymbol{x}^{(7)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$
Map	$\boldsymbol{x}^{(i)}{\boldsymbol{x}^{(i)}}^T$	$\boldsymbol{x}^{(i)}{\boldsymbol{x}^{(i)}}^T$	$\boldsymbol{x}^{(i)}{\boldsymbol{x}^{(i)}}^T$
Reduce		$\left(\sum_{i=1}^{n} \boldsymbol{x}^{(i)}{\boldsymbol{x}^{(i)}}^T\right)^{-1}$	

Distributed Computation of $(X^T X)^{-1}$

Workers	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(1)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(4)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(2)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(3)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(5)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(7)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$
Map	$\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)^T}$	$\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)^T}$	$\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)^T}$
Reduce		$\left(\sum_{i=1}^{n} \boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)^T}\right)^{-1}$	

$O(nk)$ distributed storage (total)

Distributed Computation of $(X^TX)^{-1}$

Workers	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(1)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(4)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(2)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(3)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(5)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(7)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$
Map	$\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)^T}$	$\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)^T}$	$\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)^T}$
Reduce	$\left(\sum_{i=1}^{n} \boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)^T}\right)^{-1}$		

$O(nk)$ distributed storage (total)

$O(nk^2)$ distributed work (total)

$O(k^2)$ local storage

Distributed Computation of $(X^T X)^{-1}$

Distributed Computation of $(X^T X)^{-1}$

Workers	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(1)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(4)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(2)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(3)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(5)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(7)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$
Map	$\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)^T}$	$\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)^T}$	$\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)^T}$
Reduce		$\left(\sum_{i=1}^{n} \boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)^T}\right)^{-1}$	

$O(nk)$ distributed storage (total)

$O(nk^2)$ distributed work (total)

$O(k^2)$ local storage

$O(k^3)$ local work

$O(k^2)$ local storage

Distributed Computation of $(X^TX)^{-1}$

Workers	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(1)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(4)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(2)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(3)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(5)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(7)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$O(nk)$ distributed storage (total)	
`trainData.map(compute_outer_prods)`				$O(nk^2)$ distributed work (total)	$O(k^2)$ local storage
`trainData.reduce(sum_and_invert)`				$O(k^3)$ local work	$O(k^2)$ local storage

Linear Regression: Large n, Large k

- Now, $O(k^3)$ computation and $O(k^2)$ storage is *not* possible on a single machine

  We cannot store and invert $X^T X$

  We cannot compute $X^T X$

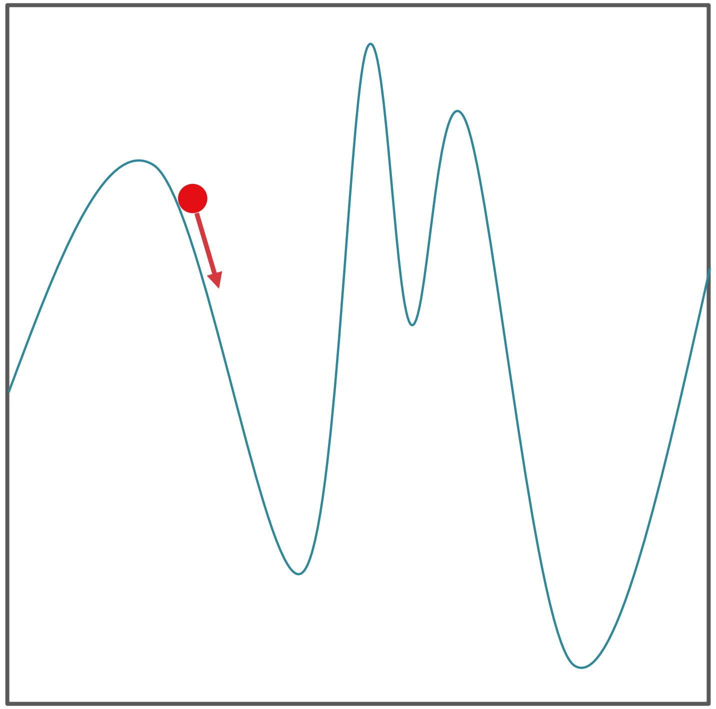
  We cannot store X

10-605/805 Principle #1: computation and storage should be at most linear in n and k

- Idea: use a different algorithm!

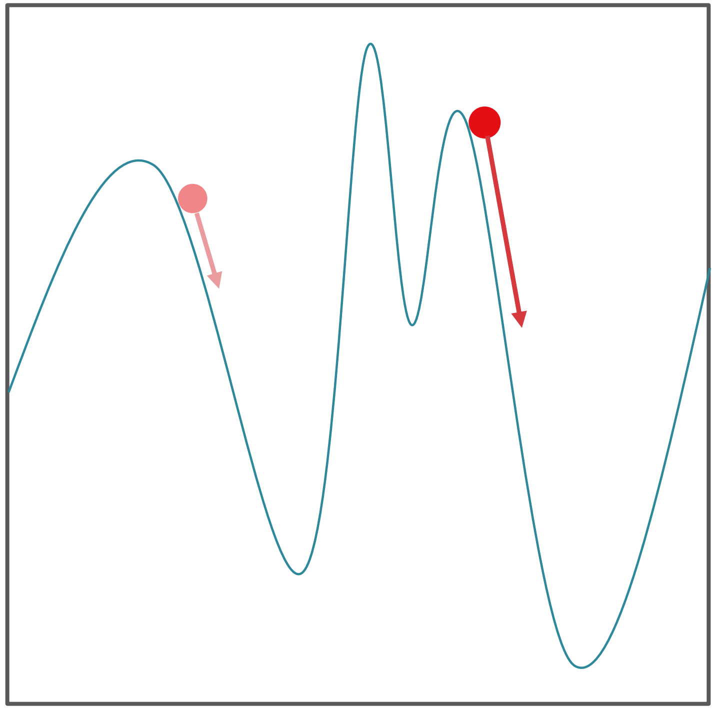
Background: Gradient Descent

- An iterative method for minimizing functions
- Requires the gradient to exist everywhere



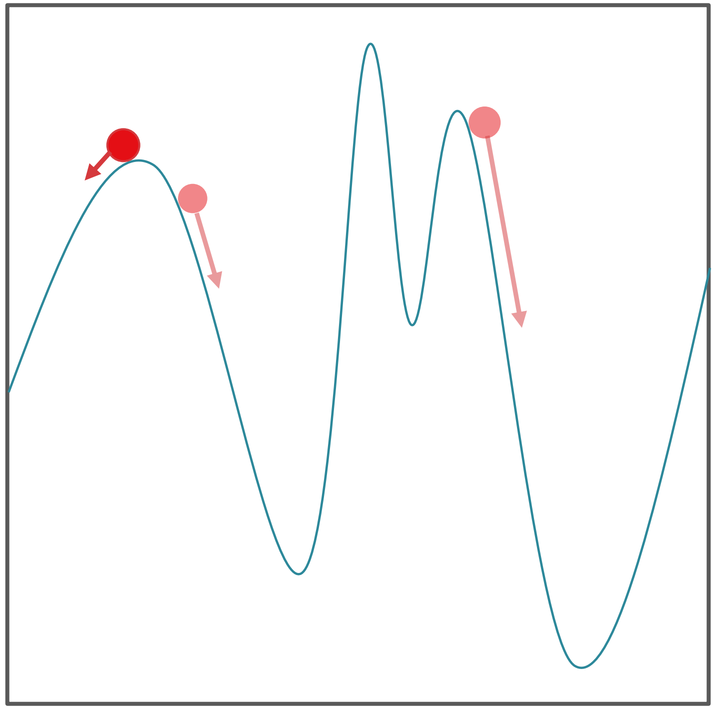
Background: Gradient Descent

- An iterative method for minimizing functions
- Requires the gradient to exist everywhere



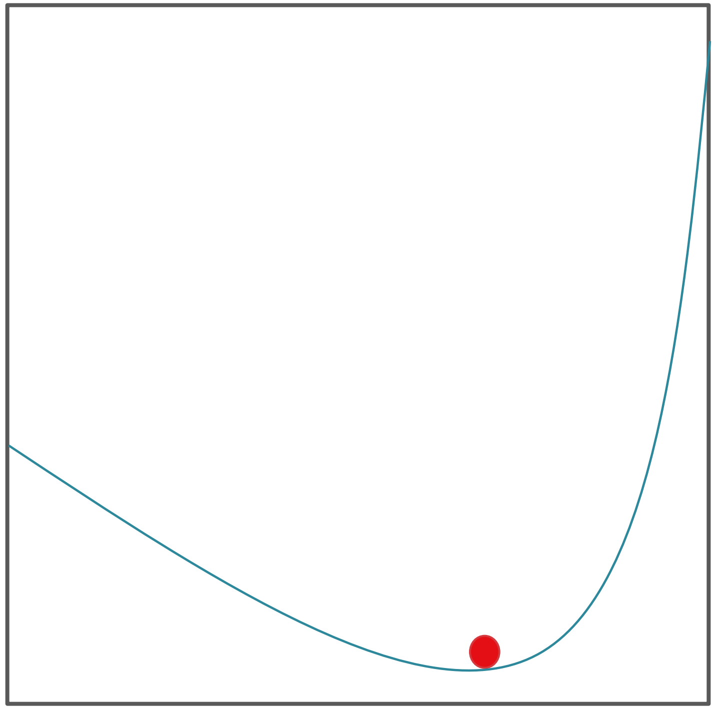
Background: Gradient Descent

- An iterative method for minimizing functions
- Requires the gradient to exist everywhere



Background: Gradient Descent

- An iterative method for minimizing functions
- Requires the gradient to exist everywhere



- Good news: the linear regression objective is convex so gradient descent will always converge to the global minimum

Background: Gradient Descent

- Suppose we're trying to minimize some function L and we're currently at some location $\boldsymbol{w}^{(t)}$
- Move some distance, α, in the "most downhill" direction, $\boldsymbol{v}$:

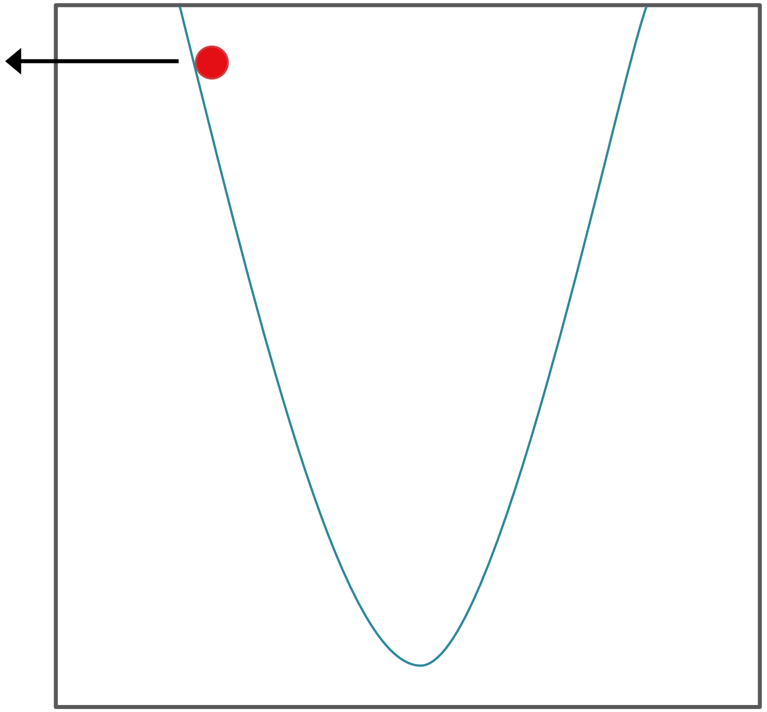
$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \alpha \boldsymbol{v}$$

- The gradient points in the direction of steepest *increase* …
- … so let's move in the opposite direction!

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \alpha \nabla_{\boldsymbol{w}} L\left(\boldsymbol{w}^{(t)}\right)$$

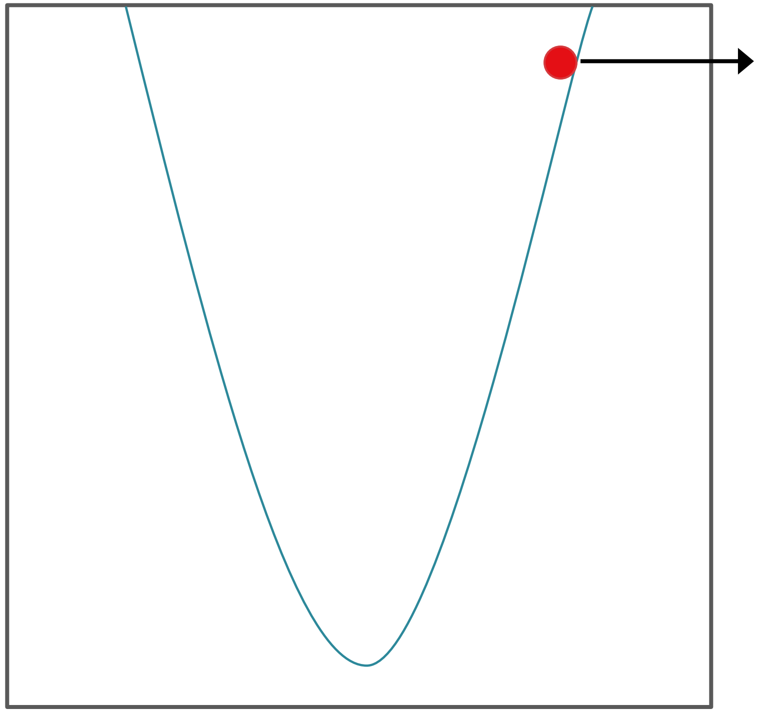
Background: Gradient Descent

Direction of gradient



$2x < 0$ for $x < 0$

Direction of gradient



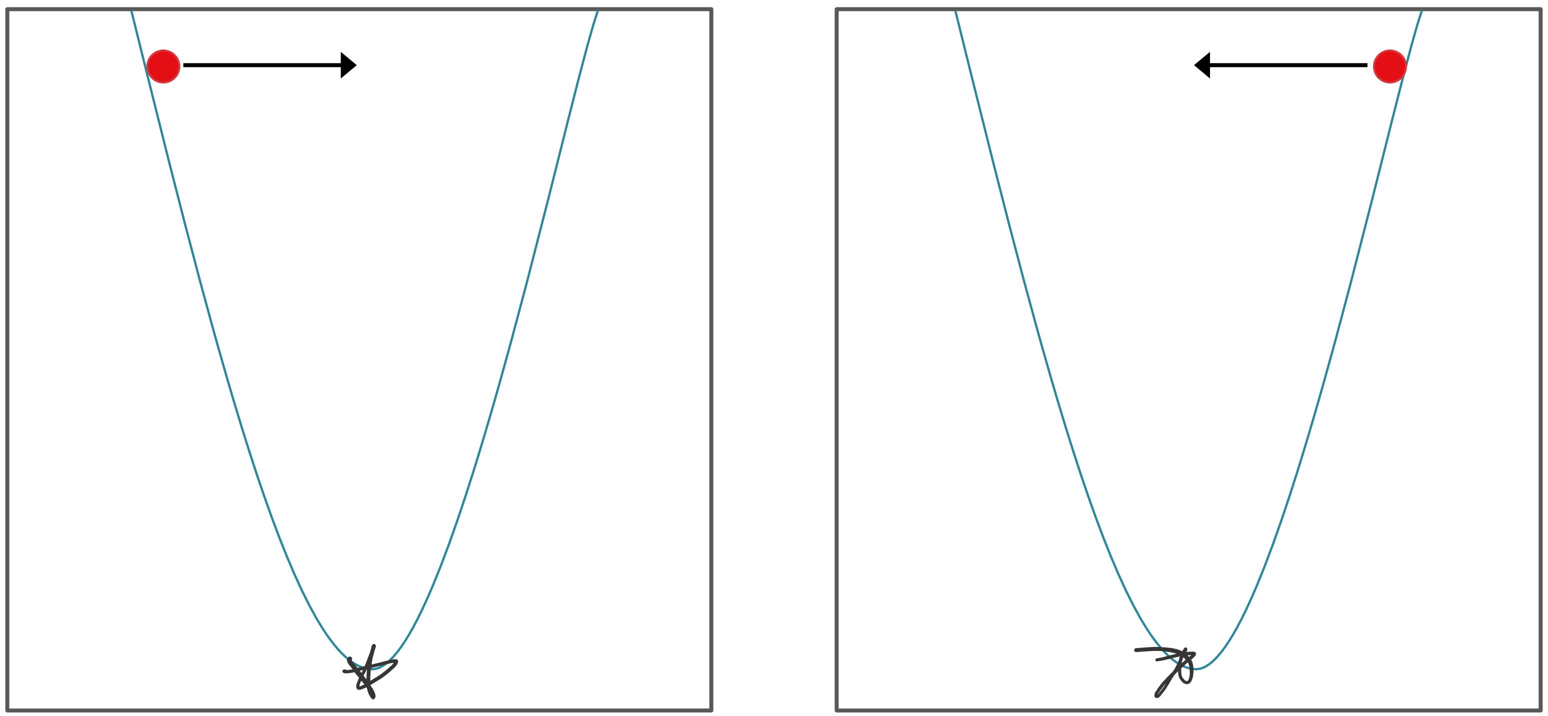
$2x > 0$ for $x > 0$

$$\frac{\partial}{\partial x} x^2 = 2x$$

Background: Gradient Descent

Direction of global minimum

Direction of global minimum

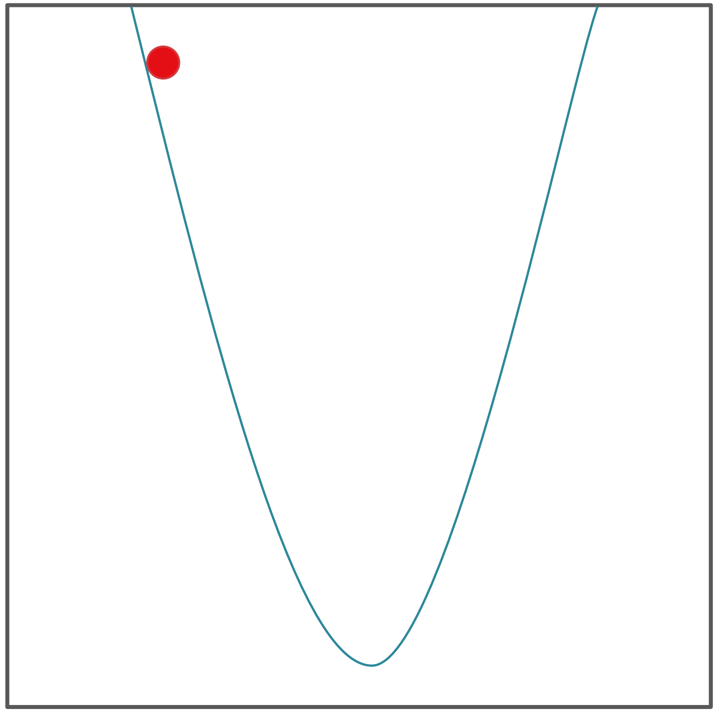


$2x < 0$ for $x < 0$

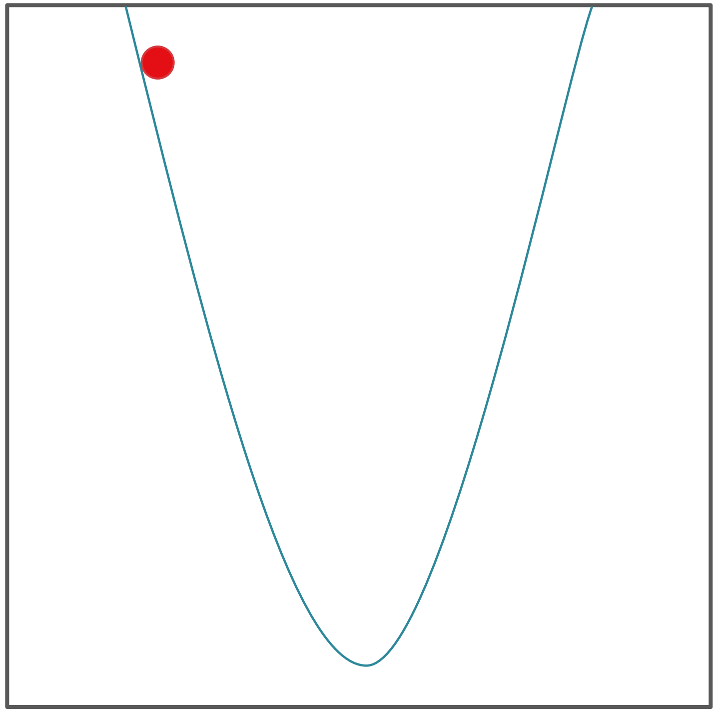
$2x > 0$ for $x > 0$

$$\frac{\partial}{\partial x}x^2 = 2x$$

Background: Gradient Descent Step Size

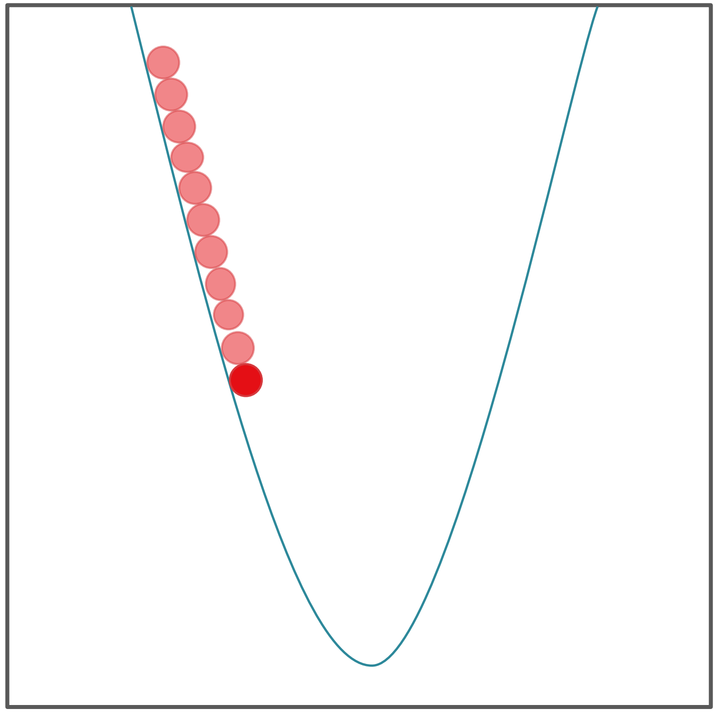


Small α

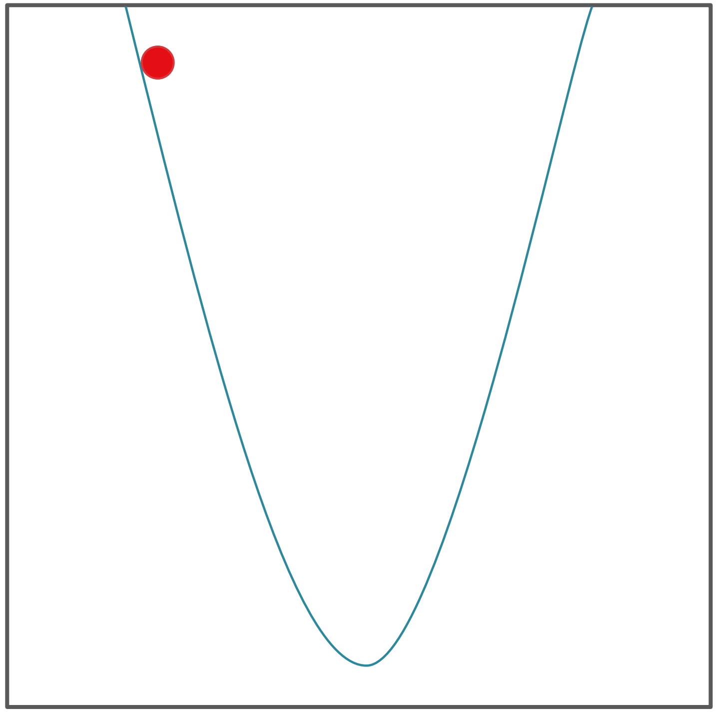


Large α

Background: Gradient Descent Step Size

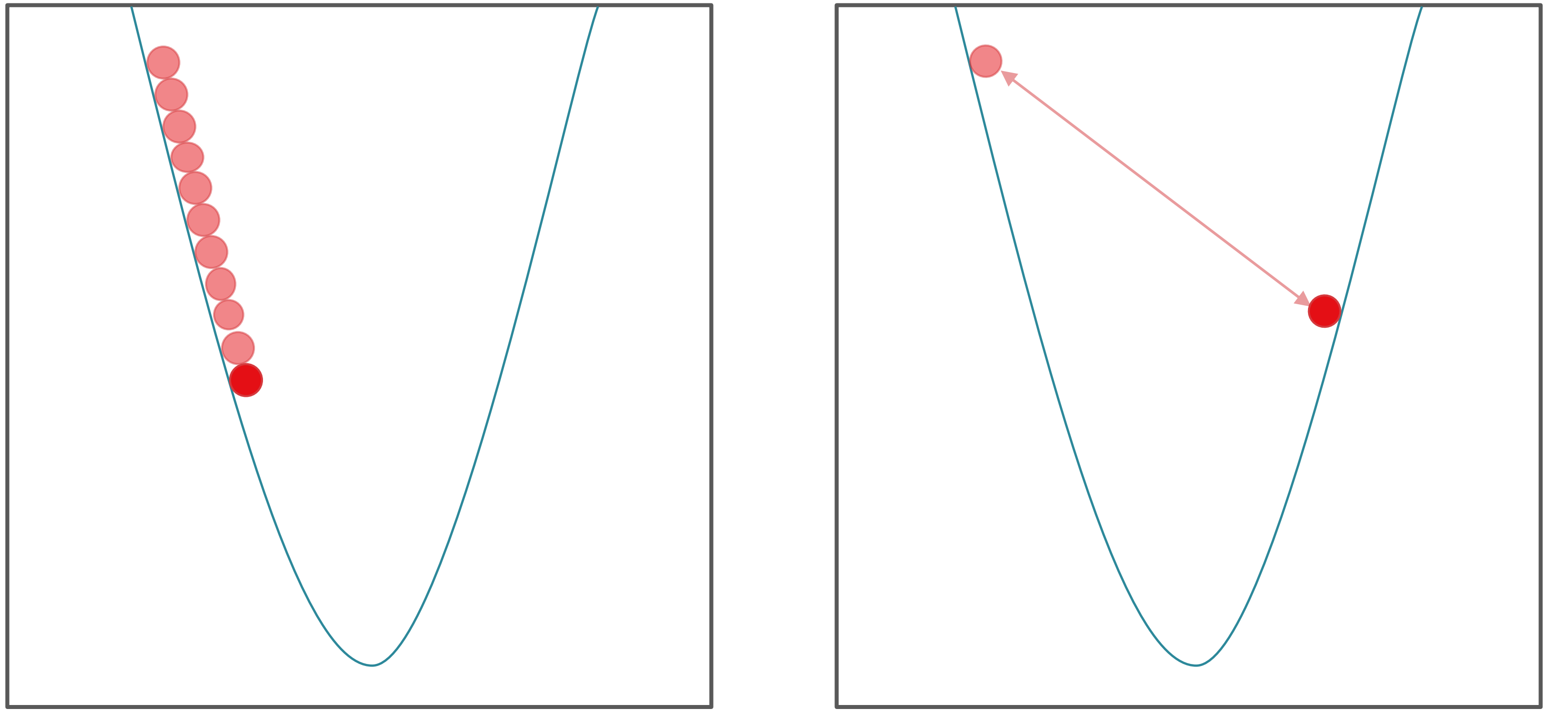


Small α



Large α

Background: Gradient Descent Step Size

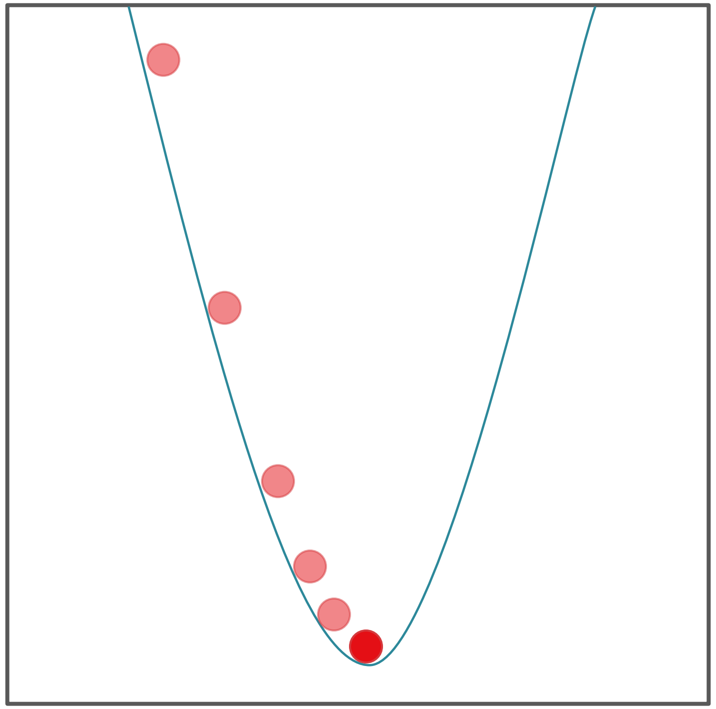


Small α

Large α

Background: Gradient Descent Step Size

- Use a variable $\alpha^{(t)}$ instead of a fixed α!



- Example: $\alpha^{(t)} = \frac{\alpha}{n\sqrt{t}}$

Gradient Descent for Linear Regression

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}, \alpha$

1. Initialize $\boldsymbol{w}^{(0)}$ to all zeros and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. Compute the gradient:

   $$\nabla_{\boldsymbol{w}} L_{\mathcal{D}}\left(\boldsymbol{w}^{(t)}\right) = \left(2X^T X \boldsymbol{w}^{(t)} - 2X^T \boldsymbol{y}\right)$$

   b. Update the weights:

   $$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \frac{\alpha}{n\sqrt{t}}\left(2X^T X \boldsymbol{w}^{(t)} - 2X^T \boldsymbol{y}\right)$$

   c. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{w}^{(t)}$

Gradient Descent for Linear Regression

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}, \alpha$

1. Initialize $\boldsymbol{w}^{(0)}$ to all zeros and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. Compute the gradient:
   $$\nabla_{\boldsymbol{w}} L_{\mathcal{D}}\left(\boldsymbol{w}^{(t)}\right) = 2\sum_{i=1}^{n}\left(\boldsymbol{w}^{(t)^T}\boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$$
   b. Update the weights:
   $$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \frac{\alpha}{n\sqrt{t}}\sum_{i=1}^{n}\left(\boldsymbol{w}^{(t)^T}\boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$$
   c. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{w}^{(t)}$

Idea: distribute $\boldsymbol{x}^{(i)}$ and compute summands in parallel

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}, \alpha$

1. Initialize $\boldsymbol{w}^{(0)}$ to all zeros and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. Compute the gradient:

$$\nabla_{\boldsymbol{w}} L_{\mathcal{D}}\left(\boldsymbol{w}^{(t)}\right) = 2 \underbrace{\sum_{i=1}^{n} \left(\boldsymbol{w}^{(t)^T} \boldsymbol{x}^{(i)} - y^{(i)}\right) \boldsymbol{x}^{(i)}}$$

   b. Update the weights:

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \frac{\alpha}{n\sqrt{t}} \sum_{i=1}^{n} \left(\boldsymbol{w}^{(t)^T} \boldsymbol{x}^{(i)} - y^{(i)}\right) \boldsymbol{x}^{(i)}$$

   c. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{w}^{(t)}$

Workers	$\begin{bmatrix} \leftarrow & {x^{(1)}}^T & \rightarrow \\ \leftarrow & {x^{(4)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & {x^{(2)}}^T & \rightarrow \\ \leftarrow & {x^{(3)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & {x^{(5)}}^T & \rightarrow \\ \leftarrow & {x^{(7)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$
Map	$\left({\boldsymbol{w}^{(t)}}^T \boldsymbol{x}^{(i)} - y^{(i)}\right) \boldsymbol{x}^{(i)}$	$\left({\boldsymbol{w}^{(t)}}^T \boldsymbol{x}^{(i)} - y^{(i)}\right) \boldsymbol{x}^{(i)}$	$\left({\boldsymbol{w}^{(t)}}^T \boldsymbol{x}^{(i)} - y^{(i)}\right) \boldsymbol{x}^{(i)}$
Reduce	$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \frac{\alpha}{n\sqrt{t}} \sum_{i=1}^{n} \left({\boldsymbol{w}^{(t)}}^T \boldsymbol{x}^{(i)} - y^{(i)}\right) \boldsymbol{x}^{(i)}$		

Distributed Gradient Descent

Workers	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(1)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(4)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(2)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(3)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(5)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(7)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$
Map	$\left(\boldsymbol{w}^{(t)^T}\boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$	$\left(\boldsymbol{w}^{(t)^T}\boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$	$\left(\boldsymbol{w}^{(t)^T}\boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$
Reduce	$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \frac{\alpha}{n\sqrt{t}}\sum_{i=1}^{n}\left(\boldsymbol{w}^{(t)^T}\boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$		

$O(nk)$ distributed storage (total)

Distributed Gradient Descent

Workers	$\begin{bmatrix} \leftarrow & {\boldsymbol{x}^{(1)}}^T & \rightarrow \\ \leftarrow & {\boldsymbol{x}^{(4)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & {\boldsymbol{x}^{(2)}}^T & \rightarrow \\ \leftarrow & {\boldsymbol{x}^{(3)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & {\boldsymbol{x}^{(5)}}^T & \rightarrow \\ \leftarrow & {\boldsymbol{x}^{(7)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$
Map	$\left({\boldsymbol{w}^{(t)}}^T \boldsymbol{x}^{(i)} - y^{(i)}\right) \boldsymbol{x}^{(i)}$	$\left({\boldsymbol{w}^{(t)}}^T \boldsymbol{x}^{(i)} - y^{(i)}\right) \boldsymbol{x}^{(i)}$	$\left({\boldsymbol{w}^{(t)}}^T \boldsymbol{x}^{(i)} - y^{(i)}\right) \boldsymbol{x}^{(i)}$
Reduce	$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \frac{\alpha}{n\sqrt{t}} \sum_{i=1}^{n} \left({\boldsymbol{w}^{(t)}}^T \boldsymbol{x}^{(i)} - y^{(i)}\right) \boldsymbol{x}^{(i)}$		

$O(nk)$ distributed storage (total)

$O(nk)$ distributed work (total)

$O(k)$ local storage

Distributed Gradient Descent

Workers	$\begin{bmatrix} \leftarrow & {\boldsymbol{x}^{(1)}}^T & \rightarrow \\ \leftarrow & {\boldsymbol{x}^{(4)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & {\boldsymbol{x}^{(2)}}^T & \rightarrow \\ \leftarrow & {\boldsymbol{x}^{(3)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & {\boldsymbol{x}^{(5)}}^T & \rightarrow \\ \leftarrow & {\boldsymbol{x}^{(7)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$O(nk)$ distributed storage (total)	
Map	$\left({\boldsymbol{w}^{(t)}}^T \boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$	$\left({\boldsymbol{w}^{(t)}}^T \boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$	$\left({\boldsymbol{w}^{(t)}}^T \boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$	$O(nk)$ distributed work (total)	$O(k)$ local storage
Reduce	$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \frac{\alpha}{n\sqrt{t}} \sum_{i=1}^{n} \left({\boldsymbol{w}^{(t)}}^T \boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$			$O(k)$ local work	$O(k)$ local storage

Distributed Gradient Descent

Workers	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(1)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(4)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(2)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(3)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(5)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(7)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$O(nk)$ distributed storage (total)	
Map	$\left(\boldsymbol{w}^{(t)^T}\boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$	$\left(\boldsymbol{w}^{(t)^T}\boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$	$\left(\boldsymbol{w}^{(t)^T}\boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$	$O(nk)$ distributed work (total)	$O(k)$ local storage
Reduce	$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \frac{\alpha}{n\sqrt{t}}\sum_{i=1}^{n}\left(\boldsymbol{w}^{(t)^T}\boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$			$O(k)$ local work	$O(k)$ local storage

Issue: all workers must have the latest weight vector

Distributed Gradient Descent

Workers	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(1)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(4)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(2)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(3)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(5)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(7)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$

$O(nk)$ distributed storage (total)

`trainData.map(compute_pointwise_grads(`$\boldsymbol{w}^{(t)}$`))`

$O(nk)$ distributed work (total)

$O(k)$ local storage

`trainData.sum()`

$O(k)$ local work

$O(k)$ local storage

Issue: all workers must have the latest weight vector

Distributed Gradient Descent

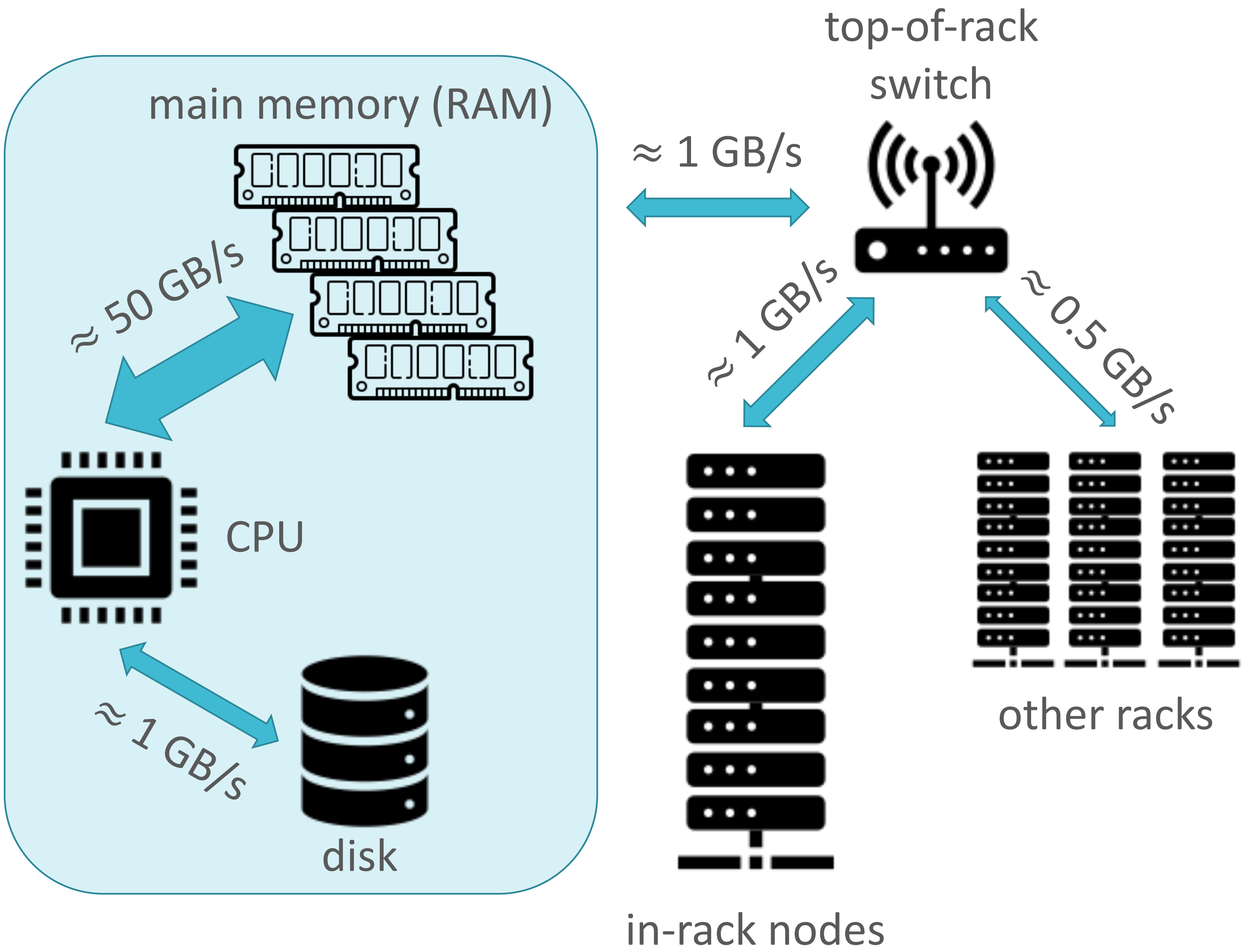
Gradient Descent

- Pros:
  - Trivially parallelizable
  - Each individual iteration is cheap
    - Can be further improved using stochastic variants
  - Guaranteed to converge on convex objective functions
- Cons:
  - Potentially slow convergence
  - Introduction of a hyperparameter
  - Network communication in each iteration

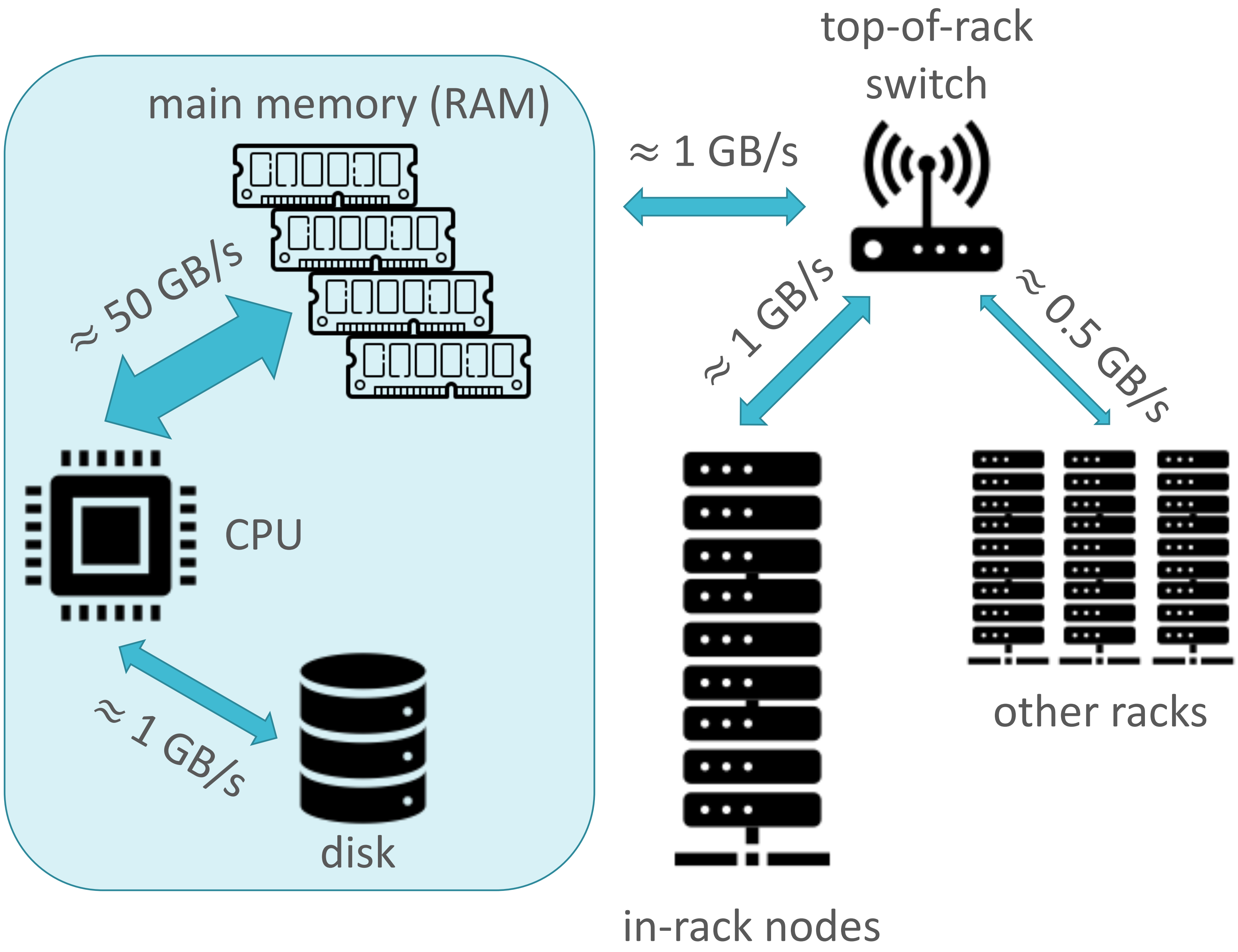
Gradient Descent

- Pros:
  - Trivially parallelizable
  - Each individual iteration is cheap
    - Can be further improved using stochastic variants
  - Guaranteed to converge on convex objective functions
- Cons:
  - Potentially slow convergence
  - Introduction of a hyperparameter
  - **Network communication in each iteration**

Recall: Communication Hierarchy



10-605/805 Principle #2: Perform parallel and in-memory computation whenever possible



10-605/805 Principle #2: Perform parallel and in-memory computation whenever possible

- Persisting data in-memory reduces communication, especially for iterative procedures

```
trainData.cache()   or   .persist()

for t in range(num_iters):
    alpha_t = alpha / n * sqrt(t)
    grad = trainData.map(compute_pointwise_grads(w)).sum()
    w -= alpha_t * grad
```

10-605/805 Principle #3: Minimize network communication

- Inherently at odds with Principle #2 → need to tradeoff between parallelism and network communication
- Three types of objects that may need to be communicated:
  - Data
  - Models
  - Intermediate objects
- Strategies:
  - Keep large objects local
  - Reduce the number of iterations

Workers	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(1)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(4)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(2)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(3)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & \boldsymbol{x}^{(5)^T} & \rightarrow \\ \leftarrow & \boldsymbol{x}^{(7)^T} & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$
Map	$\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)^T}$	$\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)^T}$	$\boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)^T}$
Reduce		$\left(\sum_{i=1}^{n} \boldsymbol{x}^{(i)}\boldsymbol{x}^{(i)^T}\right)^{-1}$	

$O(nk)$ distributed storage (total)

$O(nk^2)$ distributed work (total)

$O(k^2)$ local storage

$O(k^3)$ local work

$O(k^2)$ local storage

Data Parallel: Compute outer products locally

Workers	$\begin{bmatrix} \leftarrow & {\boldsymbol{x}^{(1)}}^T & \rightarrow \\ \leftarrow & {\boldsymbol{x}^{(4)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & {\boldsymbol{x}^{(2)}}^T & \rightarrow \\ \leftarrow & {\boldsymbol{x}^{(3)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$	$\begin{bmatrix} \leftarrow & {\boldsymbol{x}^{(5)}}^T & \rightarrow \\ \leftarrow & {\boldsymbol{x}^{(7)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$
Map	$\left({\boldsymbol{w}^{(t)}}^T\boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$	$\left({\boldsymbol{w}^{(t)}}^T\boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$	$\left({\boldsymbol{w}^{(t)}}^T\boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$
Reduce	$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \frac{\alpha}{n\sqrt{t}}\sum_{i=1}^{n}\left({\boldsymbol{w}^{(t)}}^T\boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$		

$O(nk)$ distributed storage (total)

$O(nk)$ distributed work (total)

$O(k)$ local storage

$O(k)$ local work

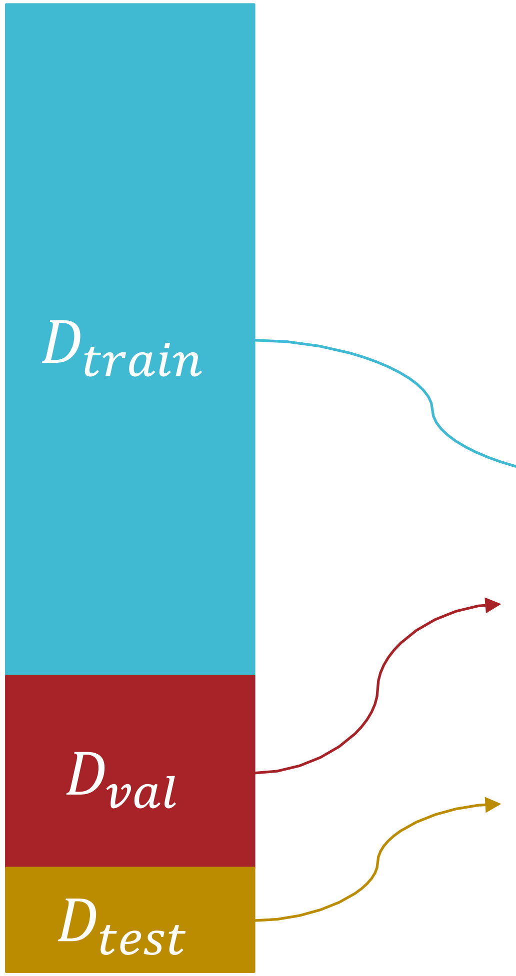
$O(k)$ local storage

Issue: all workers must have the latest weight vector

Data Parallel: Compute pointwise gradients locally

Model Parallel: Train each hyperparameter setting on different machine(s)

- Hyperparameter optimization



- Pick C evenly spaced values over some prespecified range $\lambda_1, \ldots, \lambda_C$
  e.g., $\{10^{-8}, 10^{-6}, 10^{-4}, 10^{-2}, 10^{0}\}$
- For $c = 1, 2, \ldots, C$
  - Train a model on D_{train} using λ_c
- Evaluate each model on D_{val} and find the best hyperparameter setting, λ_{c^*}
- Compute the error of a model trained with λ_{c^*} on D_{test}

Key Takeaways

- 10-605/805 Principles:
  1. Computation and storage should be linear in n and k
     - For linear regression:
       - When k is small, distribute covariance matrix computation using outer products
       - When k is large, minimize squared error via distributed gradient descent
  2. Perform parallel and in-memory computation whenever possible
  3. Minimize network communication
     - Data vs model parallelism