# Distributed Linear Regression

Main References

Ameet Talwalkar and Henry Chai, **ML with Large Datasets**, CMU

## Outline

- Machine learning overview
- Large datasets and parallel computing
- Linear regression
- Distributed linear regression

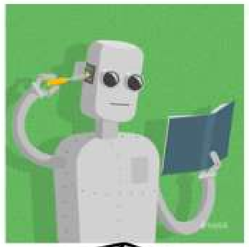
## What is machine learning?

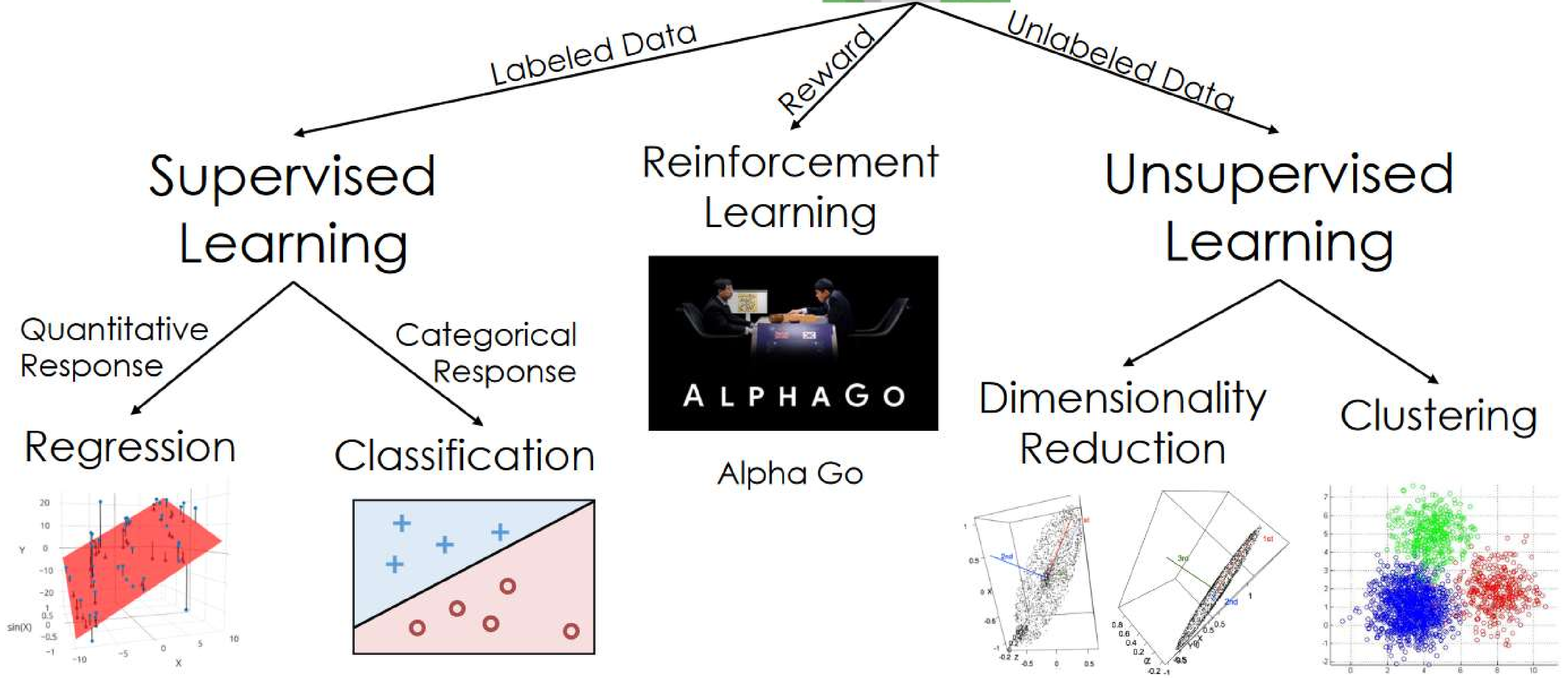
- “Machine learning as a field of study that gives computers the ability to learn without being explicitly programmed” - Arthur Samuel (1959)

- “A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P* if its performance at tasks in *T*, as measured by *P*, improves with experience *E*.” - Tom M. Mitchell (1997)

- Machine learning is about designing algorithms that learn from data.

# Taxonomy of Machine Learning



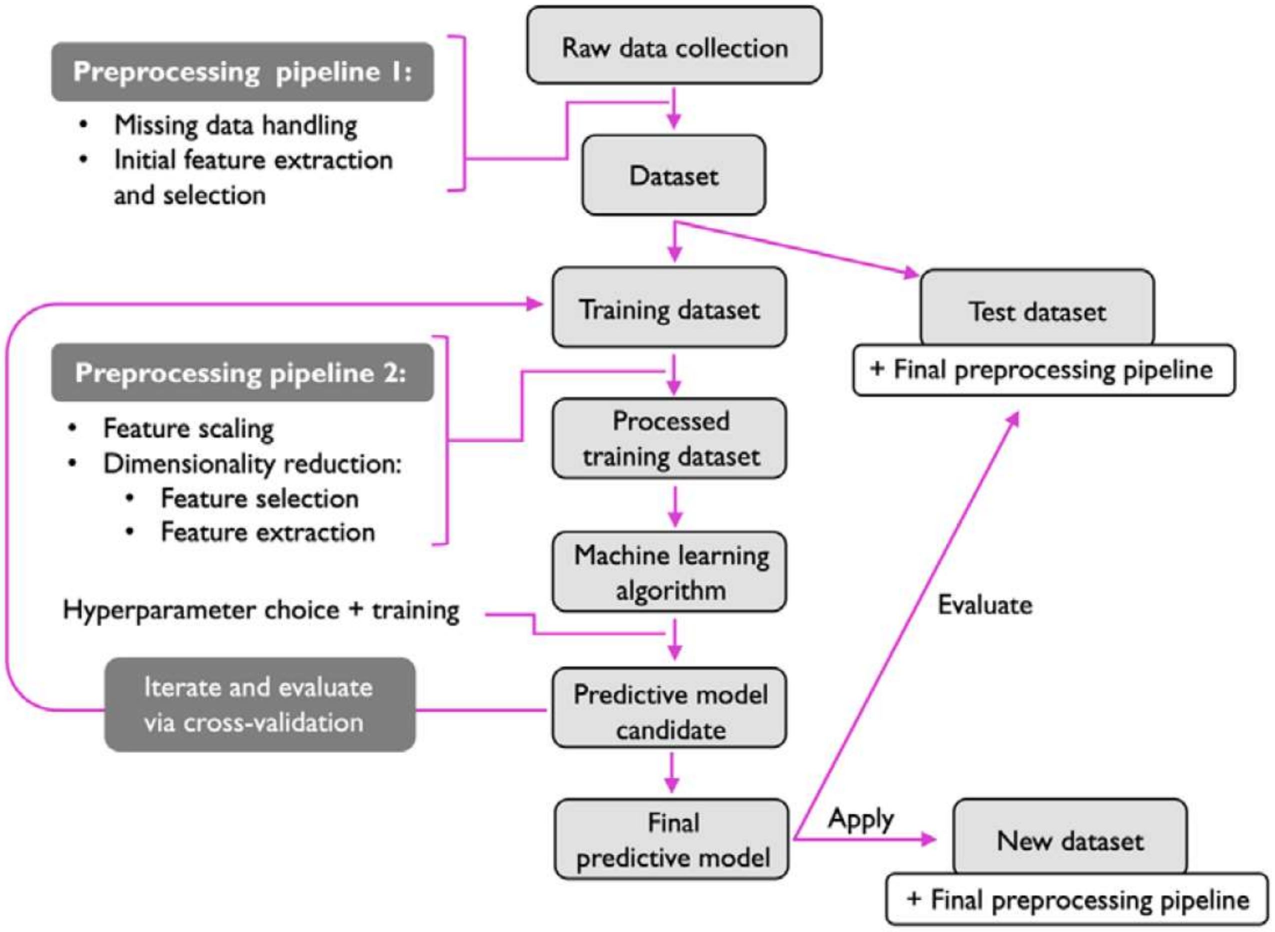


Source: Joseph E. Gonzalez, *AI-Systems Big Ideas*, 2019.

## Machine learning terminology

- Training example: A row in a table representing the dataset and synonymous with an observation, record, instance
- Training: Model fitting
- Feature, abbrev. x: A column in a data table or data matrix. Synonymous with predictor, variable, input, attribute, or covariate.
- Target, abbrev. y: Synonymous with outcome, output, response variable, dependent variable, (class) label, and ground truth.
- Loss function: measured for a single data point. Sometimes, also called a error function
- Cost function: The loss (average or summed) over the entire dataset

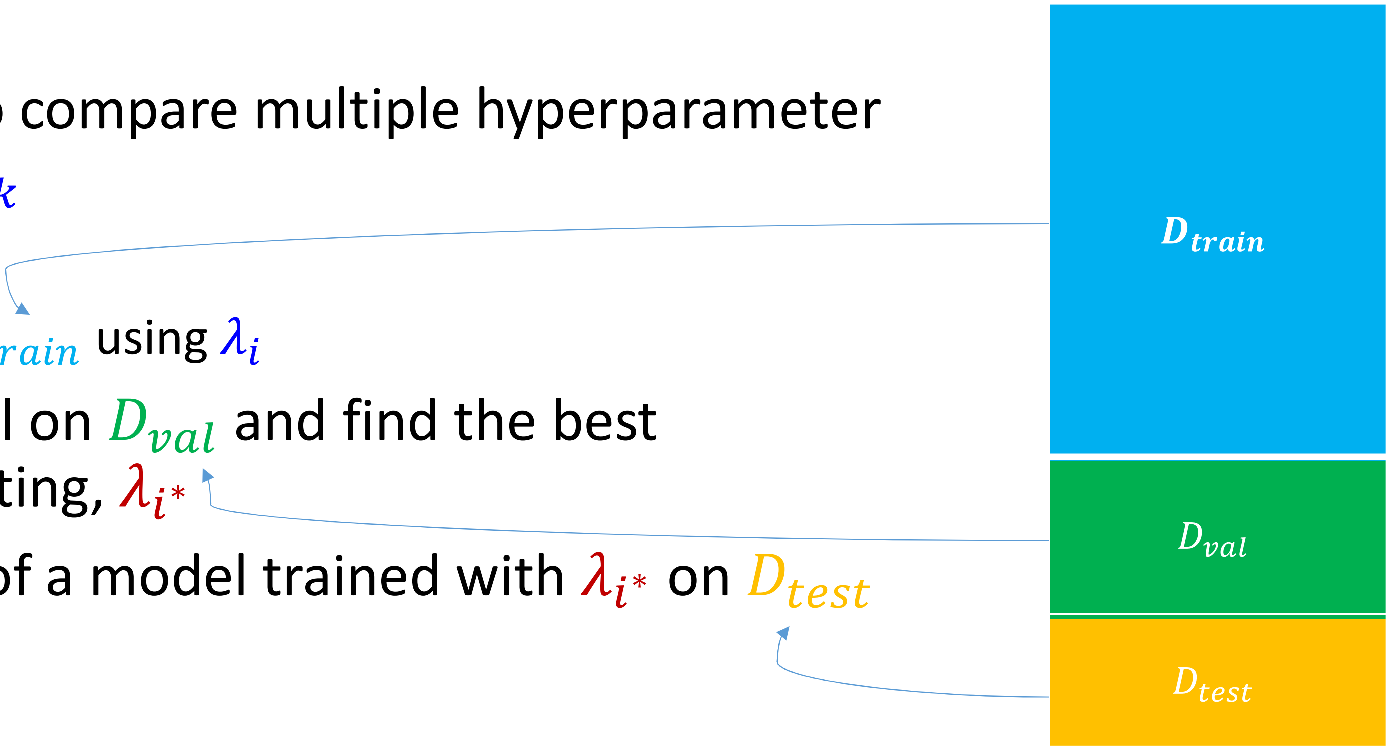
## Machine learning workflow/pipeline


Raw data collection
Preprocessing pipeline 1:
• Missing data handling
• Initial feature extraction and selection
Dataset
Training dataset
Test dataset
+ Final preprocessing pipeline
Preprocessing pipeline 2:
• Feature scaling
• Dimensionality reduction:
• Feature selection
• Feature extraction
Processed training dataset
Machine learning algorithm
Hyperparameter choice + training
Evaluate
Iterate and evaluate via cross-validation
Predictive model candidate
Final predictive model
Apply
New dataset
+ Final preprocessing pipeline

Sebastian Raschka, et al., *Giving Computers the Ability to Learn from Data*, In *Machine Learning with PyTorch and Scikit-Learn* (pp. 1–18), Packt Publishing, 2022.

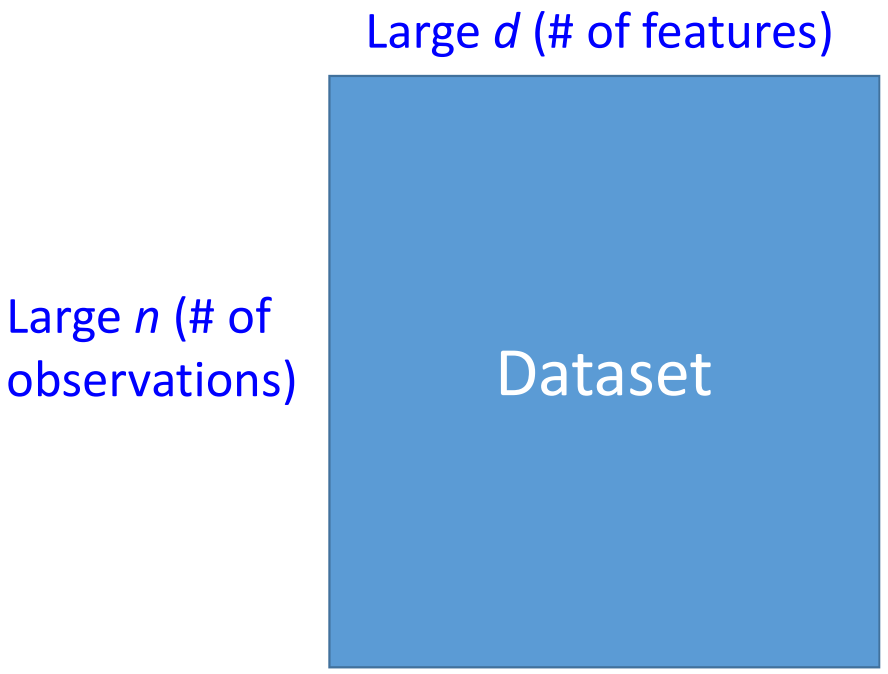
## Model selection and model evaluation

- Suppose we want to compare multiple hyperparameter settings $\lambda_1, \lambda_2, \cdots, \lambda_k$
- For $i = 1,2, \ldots, k$
  - Train a model on $D_{train}$ using $\lambda_i$
- Evaluate each model on $D_{val}$ and find the best hyperparameter setting, $\lambda_{i^*}$
- Compute the error of a model trained with $\lambda_{i^*}$ on $D_{test}$



## Large Datasets

- Datasets can be large in two ways



## Large Datasets: Example

- Image processing
  - Large n: potentially massive number of observations (e.g., pictures on the internet)
  - Use-cases: object recognition, annotation generation
- Medical data
  - Large d: potentially massive feature set (e.g., genome sequence, electronic medical records, etc...)
  - Use-cases: personalized medicine, diagnosis prediction
- Business analytics
  - Large n (e.g., all customers & all products) and d (e.g., customer data, product specifications, transaction records, etc...)
  - Use-cases: product recommendations, customer segmentation

## Large *d* (# of features)

- High-dimensional datasets present numerous issues
  - Curse of dimensionality
  - Overfitting
  - Computational issues

- Strategies
  - Learn low-dimensional representations
  - Perform feature selection

## Large *n* (# of observations)

- Typically, we consider exponential time complexity (e.g., $O(2^n)$) bad and polynomial complexity (e.g., $O(n^3)$) good
- However, if n is massive, then even O(n) can be problematic!

- Strategies
  - Speed up processing e.g., stochastic gradient descent vs. gradient descent
  - Sampling (make approximations)
  - Parallel computing (large scale data processing)

## Parallel computing

- Multi-core processing – scale up
  - Data can fit on one machine
  - Requires high-end (expensive) hardware
  - Simpler algorithms that don’t necessarily scale well
- Distributed processing – scale out
  - Data stored across multiple machines
  - Scales to massive problems on commodity (inexpensive) hardware
  - Added complexity of network communication

## Big O Notation

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

$$f(n) = O\big(g(n)\big) \Leftrightarrow \exists c, n_0 : f(n) \leq cg(n) \forall n \geq n_0$$

  - $O(1)$= constant time/space
  - $O(\log(n))$ = logarithmic time/space
  - $O(n)$ = linear time/space
- An algorithm's time and space complexity can be different
  - Ex: multiplying an $m \times n$ matrix with an $n \times p$ matrix takes $O(mnp)$ time but the result uses $O(mn + np + mp)$ storage

# Empirical Risk Minimization – ERM

- ERM is a common framework for supervised learning
- Given:
  - some labelled training dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$
  - a loss function $l: Y \times Y \to R$
  - a hypothesis class or set of functions $F$
- The goal is to find

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

with the hope that

$$E_{p(x,y)}[l(f(x), y)] \approx \frac{1}{n} \sum_{i=1}^{n} l\left(f\left(x^{(i)}\right), y^{(i)}\right)$$

## Empirical Risk Minimization – ERM

- ERM is a common framework for supervised learning
- Given:
  - some labelled training dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$
  - a loss function $l: Y \times Y \rightarrow R$
  - a hypothesis class or set of functions $F$
- The goal is to find

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

- Depending on the choice of $F$ and $l$, this objective function may be convex (easy to optimize) or non-convex (hard)
- We nead to solve this problem for large $n$ and/or $d$

## Regression

- Regression is a type of supervised learning
- Given:
  - a labelled training dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$
  - a loss function $l: Y \times Y \to R$, where $Y = R$
  - a hypothesis class or set of functions $F$
- The goal is to find

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

- Depending on the choice of $F$ and $l$, this objective function may be convex (easy to optimize) or non-convex (hard)
- We need to solve this problem for large $n$ and/or $d$

## Linear Regression (Ordinary Least Squares)

- Linear regression is a simplest type of regression
- Given:
  - a labelled training dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$
  - a loss function $l(y, y') = (y - y')^2$
  - $F$ = all functions of the form $f(x) = \theta_0 + \sum_{i=1}^{k} \theta_i x_i = \theta^T x$
- The goal is to find

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

*or*

$$\hat{\theta} = \underset{\theta}{argmin} \|X\theta - y\|^2$$

*where:*

$$X = \begin{bmatrix} {x^{(1)}}^T \\ \vdots \\ {x^{(n)}}^T \end{bmatrix}, y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(1)} \end{bmatrix}, x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \\ \vdots \\ x_d^{(i)} \end{bmatrix}, \theta = \begin{bmatrix} 1 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix}$$

## Linear Regression (Ordinary Least Squares)

$$L_D(\theta) = \|X\theta - y\|^2 = (X\theta\ -\ y)^T(X\theta -\ y)$$

$$\vdots$$

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

## Regularizied Linear Regression

- Regularized empirical risk minimization that penalizes model complexityto deal with overfitting
- Given:
  - some labelled training dataset $D = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$
  - a loss function $l: Y \times Y \to R$, where $Y = R$
  - a hypothesis class or set of functions $F$
  - a regularizer function $r: \theta \to R$
  - a regularization parameter $\lambda$
- The goal is to find

$$\hat{\theta} = \underset{\theta}{argmim} \sum_{i=1}^{n} l\left(f_\theta\left(x^{(i)}\right), y^{(i)}\right) + \lambda r(\theta)$$

## Regularizied Linear Regression

$$L_D(\theta) = \|X\theta - y\|^2 + \lambda\|\theta\|^2 = (X\theta - y)^T(X\theta - y) + \lambda\theta^T\theta$$

$$\vdots$$

$$\hat{\theta} = (X^TX + \lambda I_{d+1})^{-1}X^Ty$$

$I_{d+1}$: is the $(d+1) \times (d+1)$ identity matrix

## Linear Regression – Large n, small d

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

- Time complexity for $(X^T X)^{-1}$ is $O(d^3)$
- Space complexity for $X^T X$ is $O(d^2)$
- Assume $O(d^3)$ computation and $O(d^2)$ storage is possible on a single machine
  - We can store and invert $X^T X$
  - We cannot compute $X^T X$ on a single machine
  - We cannot store $X$ on a single machine
- Idea: distribute storage of $X$ and computation of $X^T X$
  - Store the rows of $X$ across different machines
  - Compute $X^T X$ as the sum of outer products

## Matrix Multiplication via Outer Products

$$\begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} \begin{bmatrix} b_{11} & \cdots & b_{1k} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mk} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} a_{1i} b_{i1} & \cdots & \sum_{i=1}^{m} a_{1i} b_{ik} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^{m} a_{ni} b_{i1} & \cdots & \sum_{i=1}^{m} a_{ni} b_{ik} \end{bmatrix}$$

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

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

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

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

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

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

$O(nd)$
distributed
storage (total)

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

$O(d^2)$
local
storage

$O(d^3)$
local
work

$O(d^2)$
local
storage

## Linear Regression – Large n, large d

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

- Time complexity for $(X^T X)^{-1}$ is $O(d^3)$
- Space complexity for $X^T X$ is $O(d^2)$
- Assume $O(d^3)$ computation and $O(d^2)$ storage is possible on a single machine
    - We cannot can store and invert $X^T X$
    - We cannot compute $X^T X$ on a single machine
    - We cannot store $X$ on a single machine
- Idea: Use distributed version of gradient descent

## Gradient Descent

- We’re trying to minimize some function $L$
- Suppose at iteration $t$: we’re get $\theta^{(t)}$
- With learning rate $\eta$, we update $\theta^{(t)}$ (at iteration $t+1$) as follow

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta L(\theta^{(t)})$$

# Gradient Descent for Linear Regression

- Dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$

1. Initialize $\theta^{(0)} = 0$ (zero vector) and set $t = 0$
2. While not converged
   - Compute the gradient

$$\nabla_\theta L_D(\theta^{(t)}) = 2X^T X\theta^{(t)} - 2X^T y$$

   - Update the weights

$$\theta^{(t+1)} = \theta^{(\mathrm{t})} - \eta \nabla_\theta L_D(\theta^{(t)})$$

   - Increment $t$:

$$t = t + 1$$

- Output $\theta^{(\mathrm{t})}$

## Gradient Descent for Linear Regression – Change Step Size

- Dataset $D = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$

1. Initialize $\theta^{(0)} = 0$ (zero vector) and set $t = 0$
2. While not converged
    - Compute the gradient

$$\nabla_\theta L_D\left(\theta^{(t)}\right) = 2X^T X\theta^{(t)} - 2X^T y$$

    - Update the weights

$$\theta^{(t+1)} = \theta^{(\mathrm{t})} - \eta_t \nabla_\theta L_D\left(\theta^{(t)}\right) \qquad \eta_t = \frac{\eta_0}{n\sqrt{t+1}}$$

    - Increment $t$:

$$t = t + 1$$

- Output $\theta^{(\mathrm{t})}$

## Gradient Descent for Linear Regression – Change Step Size

- Dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$

1. Initialize $\theta^{(0)} = 0$ (zero vector) and set $t = 0$
2. While not converged
   - Compute the gradient
   $$\nabla_\theta L_D(\theta^{(t)}) = 2\sum_{i=1}^{n}\left({\theta^{(t)}}^T x^{(i)} - y^{(i)}\right)x^{(i)}$$
   - Update the weights
   $$\theta^{(t+1)} = \theta^{(t)} - \eta_t \sum_{i=1}^{n}\left({\theta^{(t)}}^T x^{(i)} - y^{(i)}\right)x^{(i)} \qquad \eta_t = \frac{\eta_0}{n\sqrt{t+1}}$$
   - Increment $t$:
   $$t = t + 1$$

- Output $\theta^{(t)}$

# Distributed Computation of $\nabla_\theta L_D\left(\theta^{(t)}\right)$

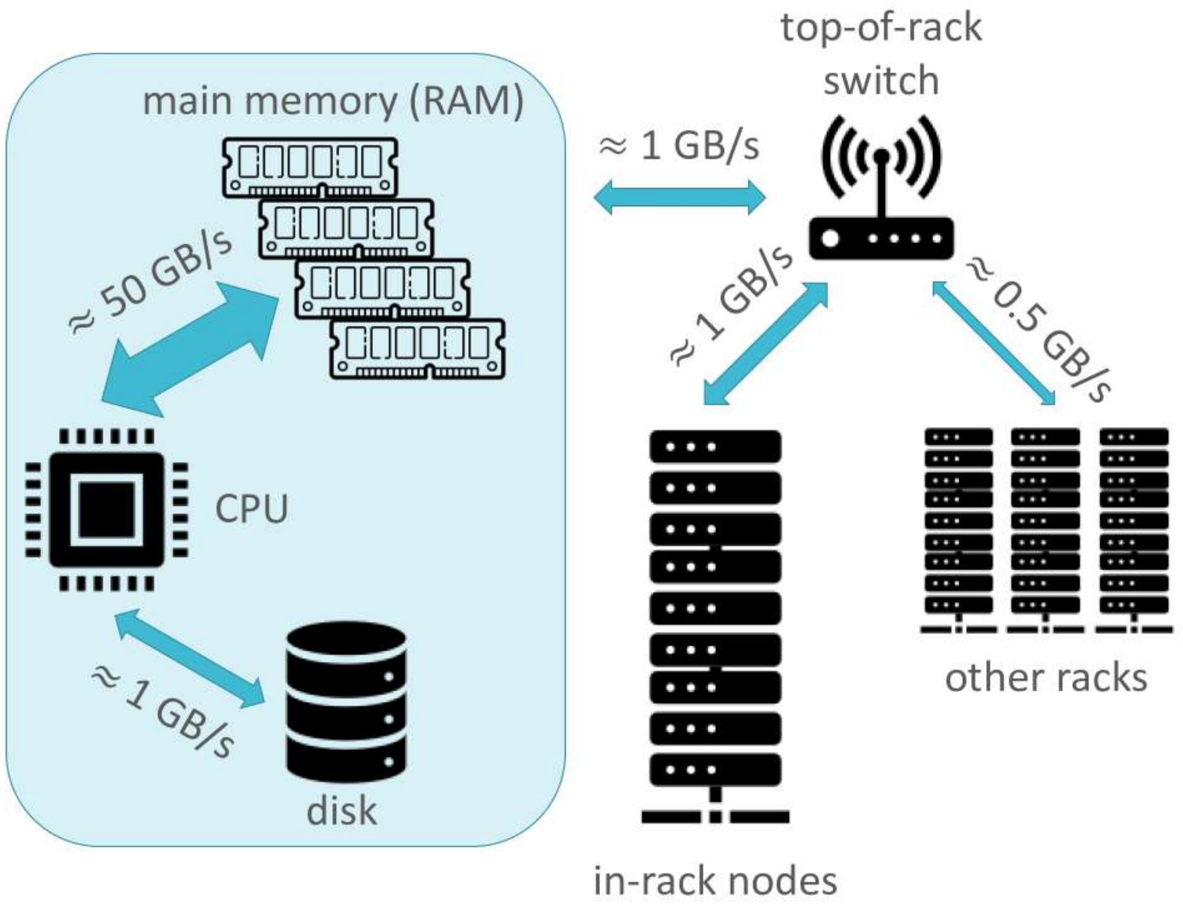
| | | | | |
|---|---|---|---|---|
| **Worker** | $\begin{bmatrix} \leftarrow & {x^{(1)}}^T & \rightarrow \\ \leftarrow & {x^{(4)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$ | $\begin{bmatrix} \leftarrow & {x^{(2)}}^T & \rightarrow \\ \leftarrow & {x^{(3)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$ | $\begin{bmatrix} \leftarrow & {x^{(5)}}^T & \rightarrow \\ \leftarrow & {x^{(7)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$ | $O(nd)$ distributed storage (total) |
| **Map** | $\left({\theta^{(t)}}^T x^{(i)} - y^{(i)}\right) x^{(i)}$ | $\left({\theta^{(t)}}^T x^{(i)} - y^{(i)}\right) x^{(i)}$ | $\left({\theta^{(t)}}^T x^{(i)} - y^{(i)}\right) x^{(i)}$ | $O(nd)$ distributed work (total); $O(d)$ local storage |
| **Reduce** | $\theta^{(t+1)} = \theta^{(\mathrm{t})} - \frac{\eta_0}{n\sqrt{t+1}} \sum_{i=1}^{n} \left({\theta^{(t)}}^T x^{(i)} - y^{(i)}\right) x^{(i)}$ | | | $O(d)$ local work; $O(d)$ local storage |

Reducer send the latest weight vector to worker

## Gradient Descent

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

## Communication Hierarchy



Perform parallel and in-memory computation whenever possible

## Minimize network communication

- Need to tradeoff between parallelism and network communication
- Three types of objects that may need to be communicated
  - Data
  - Model
  - Intermediate objects
- Strategies
  - Keep large objects local
  - Reduce the number of iterations

## Data Parallel: Compute outer products locally

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

$O(nd)$ distributed storage (total)

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

$O(d^2)$ local storage

$O(d^3)$ local work

$O(d^2)$ local storage

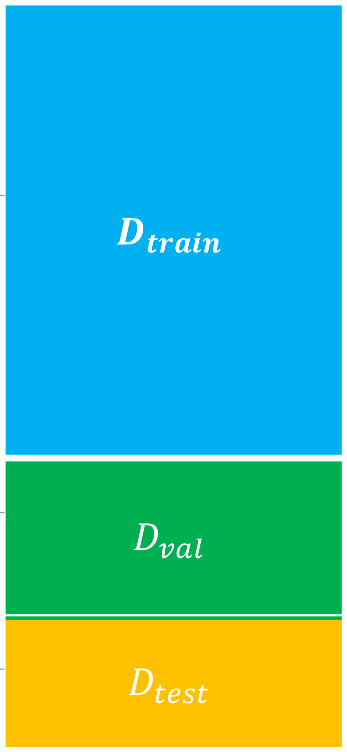
## Data Parallel: Compute pointwise gradients locally

|  |  |  |  |  |
|---|---|---|---|---|
| **Worker** | $\begin{bmatrix} \leftarrow & {x^{(1)}}^T & \rightarrow \\ \leftarrow & {x^{(4)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$ | $\begin{bmatrix} \leftarrow & {x^{(2)}}^T & \rightarrow \\ \leftarrow & {x^{(3)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$ | $\begin{bmatrix} \leftarrow & {x^{(5)}}^T & \rightarrow \\ \leftarrow & {x^{(7)}}^T & \rightarrow \\ \vdots & \vdots & \vdots \end{bmatrix}$ | $O(nd)$ distributed storage (total) |
| **Map** | $\left({\theta^{(t)}}^T x^{(i)} - y^{(i)}\right) x^{(i)}$ | $\left({\theta^{(t)}}^T x^{(i)} - y^{(i)}\right) x^{(i)}$ | $\left({\theta^{(t)}}^T x^{(i)} - y^{(i)}\right) x^{(i)}$ | $O(nd)$ distributed work (total); $O(d)$ local storage |
| **Reduce** | $\theta^{(t+1)} = \theta^{(\mathrm{t})} - \frac{\eta_0}{n\sqrt{t+1}} \sum_{i=1}^{n} \left({\theta^{(t)}}^T x^{(i)} - y^{(i)}\right) x^{(i)}$ |  |  | $O(d)$ local work; $O(d)$ local storage |

Reducer send the latest weight vector to worker

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

- Suppose we want to compare multiple hyperparameter settings $\lambda_1, \lambda_2, \cdots, \lambda_k$
- For $i = 1,2, \ldots, k$
  - Train a model on $D_{train}$ using $\lambda_i$
- Evaluate each model on $D_{val}$ and find the best hyperparameter setting, $\lambda_{i^*}$
- Compute the error of a model trained with $\lambda_{i^*}$ on $D_{test}$



## Summay

1. Computation and storage should be linear in $n$ and $k$
   - For linear regression
     - When *d* is small, distribute matrix computation using outer products
     - When *d* 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