

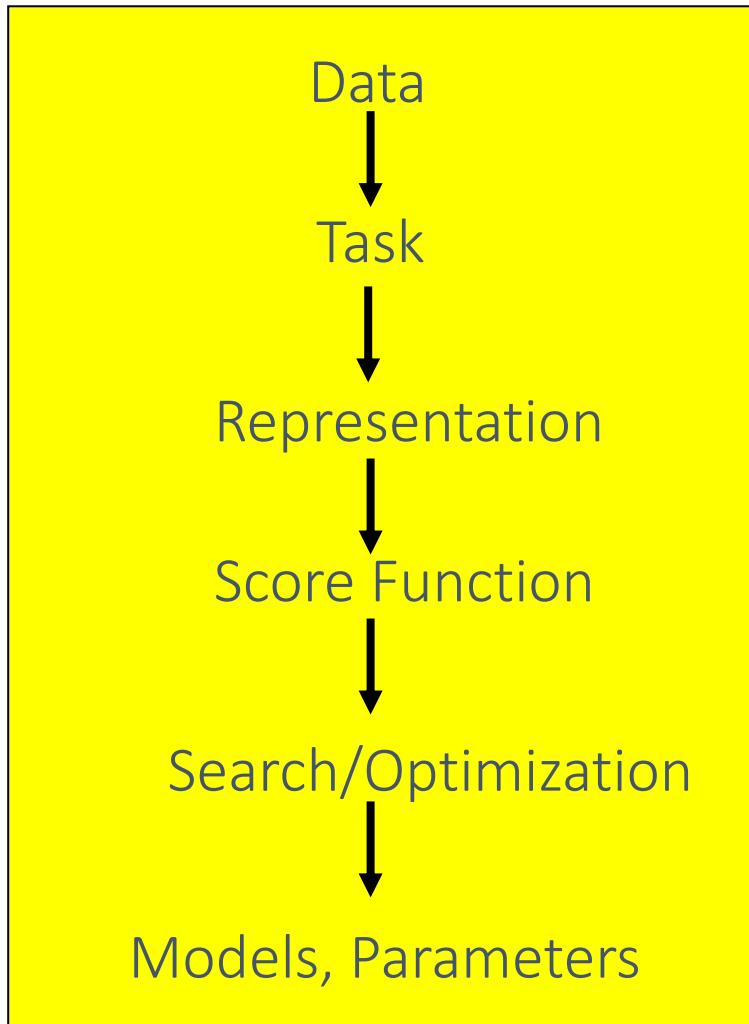
UVA CS 4774: Machine Learning

Lecture 3: Linear Regression Basics

Dr. Yanjun Qi

University of Virginia
Department of Computer Science

Machine Learning in a Nutshell



ML grew out of
work in AI

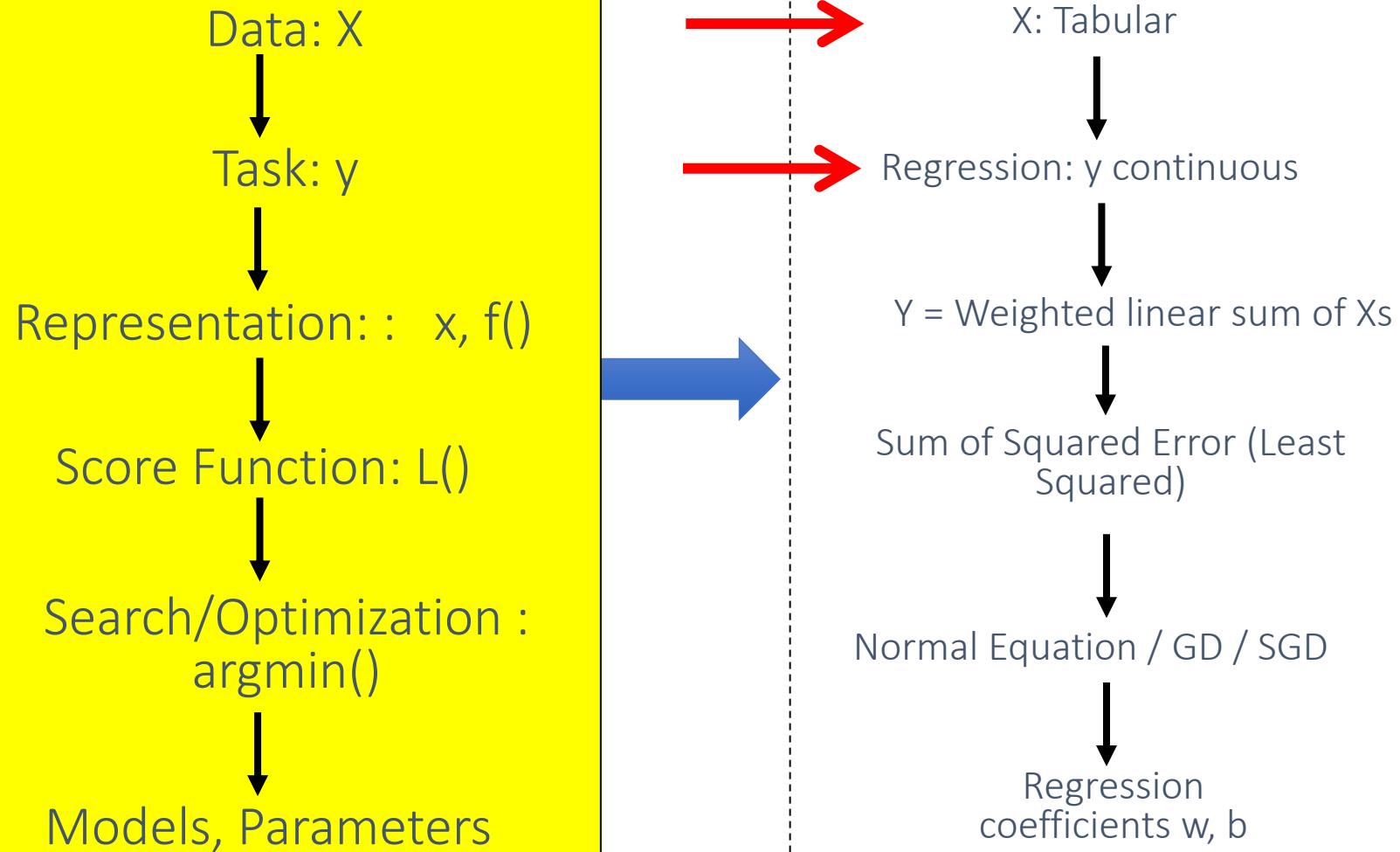
Optimize a
performance criterion
using example data or
past experience,

Aiming to generalize to
unseen data

Rough Sectioning of this Course

- 
- 1. Basic Supervised Regression + on Tabular Data
 - 2. Basic Deep Learning + on 2D Imaging Data
 - 3. Advanced Supervised learning + on Tabular Data
 - 4. Generative and Deep + on 1D Sequence Text Data
 - 5. Not Supervised + Mostly on Tabular Data

Today: Multivariate Linear Regression in a Nutshell



X_1	X_2	X_3	\vdots	Y
$s_1 \rightarrow$				
$s_2 \rightarrow$				
	\cancel{X}			
	$n \times p$			
$s_n \rightarrow$				
	x_1	$x_2 \dots x_p$		

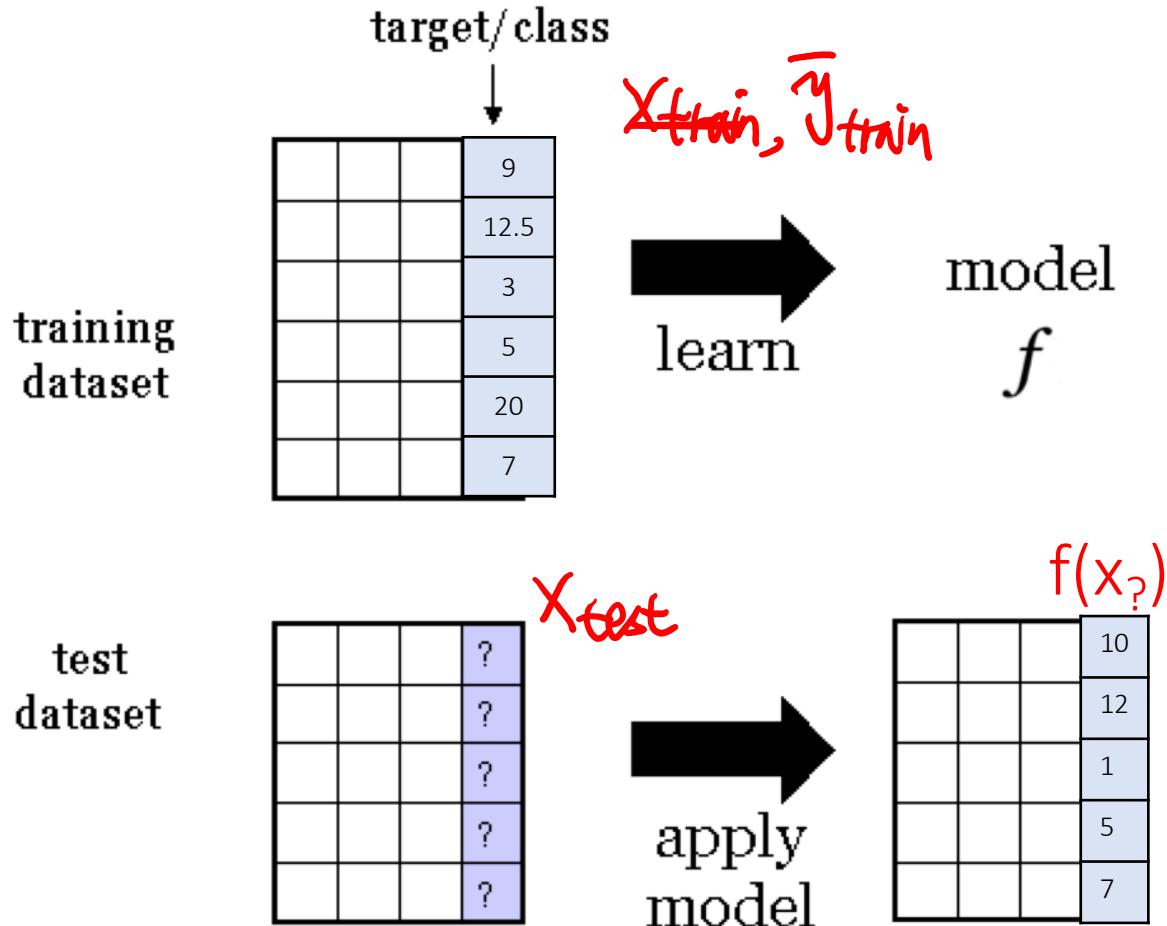
Tabular Dataset for regression

$$f : [X] \longrightarrow [Y]$$

continuous
valued
variable

- Data/points/instances/examples/samples/records: [rows]
- Features/attributes/dimensions/independent variables/covariates/predictors/regressors: [columns, except the last]
- Target/outcome/response/label/dependent variable: special column to be predicted [last column]

SUPERVISED Regression



Training dataset
consists of
input-output
pairs

- Target Y: **continuous** target variable

training dataset

$$\mathbf{X}_{train} = \begin{bmatrix} \cdots & \mathbf{x}_1^T & \cdots \\ \cdots & \mathbf{x}_2^T & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & \mathbf{x}_n^T & \cdots \end{bmatrix} \quad \vec{\mathbf{y}}_{train} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

test dataset

$$\mathbf{X}_{test} = \begin{bmatrix} \cdots & \mathbf{x}_{n+1}^T & \cdots \\ \cdots & \mathbf{x}_{n+2}^T & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & \mathbf{x}_{n+m}^T & \cdots \end{bmatrix} \quad \vec{\mathbf{y}}_{test} = \begin{bmatrix} y_{n+1} \\ y_{n+2} \\ \vdots \\ y_{n+m} \end{bmatrix}$$

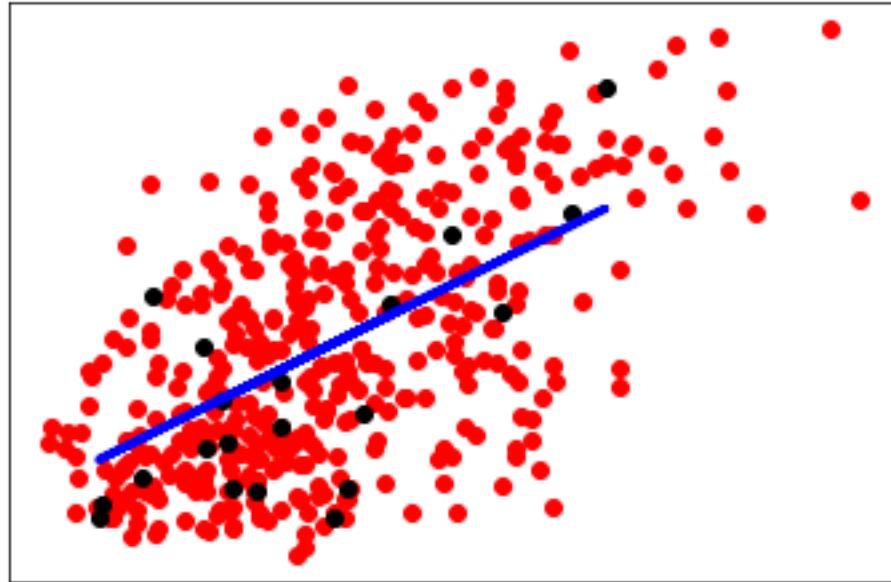
$f(\mathbf{x}_{test})$

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

<https://colab.research.google.com/drive/1ml-PFB2J1UVlp6JjLhQKEPirPZIQnDH7?usp=sharing>

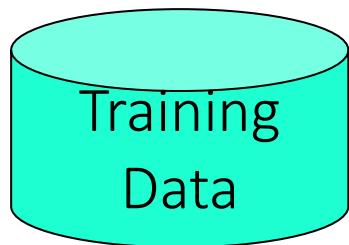
I will code-run through: Scikit-Learn Linear Regression Example

Adapted from: https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html



Two Modes of Machine Learning

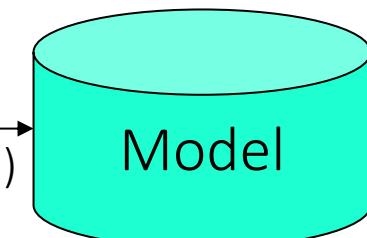
Consists of **input-output** pairs



(X, Y)

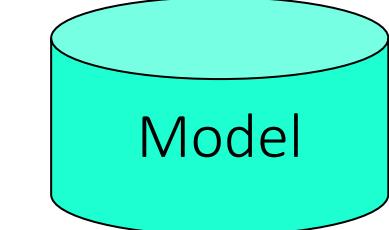
reg. fit

Computer



$f()$

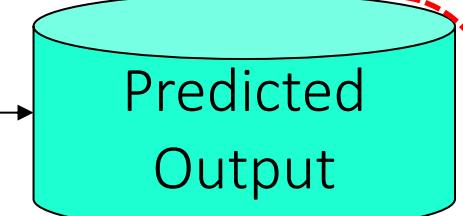
Training



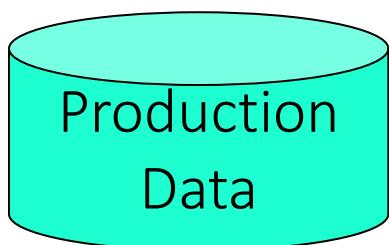
$f()$

reg. predict

Computer



$f(x?)$



$X?$

Testing

Non-Linear Regression: Example on COVID19

<https://www.kaggle.com/nitishabbarathi/the-story-of-covid-19-in-india-eda-and-prediction>

<https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge>

I will code-run through: The Story of COVID-19 in India EDA and Prediction
Adapted from:

<https://www.kaggle.com/nitishabbarathi/the-story-of-covid-19-in-india-eda-and-prediction>



The Story of COVID-19 in India EDA and Prediction

Python notebook using data from [multiple data sources](#) · 23,372 views · 4mo ago · beginner, data visualization, exploratory data analysis, +1 more



284

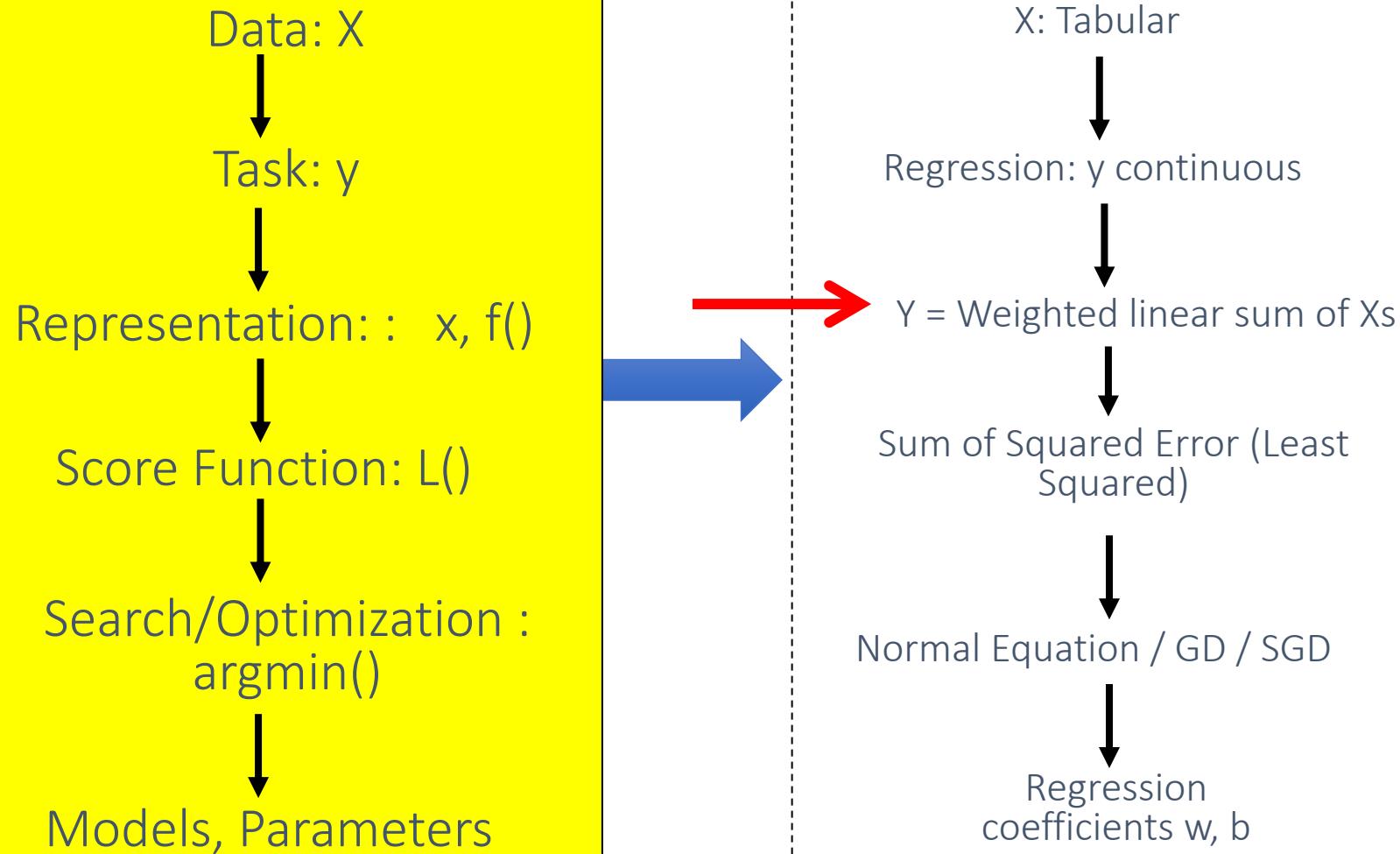
Table of Contents

- [World Updates](#)
- [India Updates](#)
- [Age Group/Gender Analysis](#)
- [The Spike in India](#)
- [Testing in India](#)
- [Statewise Insights](#)
- [Statewise Testing and Healthcare Insights](#)
- [Economic Impact of COVID-19 on India](#)
- [Prediction](#)
 - [Growth Factor](#)
 - [Prophet Model](#)
 - [ARIMA \(5,2,0\)](#)
 - [LightGBM](#)
 - [Random Forest Regressor](#)
 - [XGBoost Regressor](#)
- [Interesting Insights from other Sources](#)

Thank You



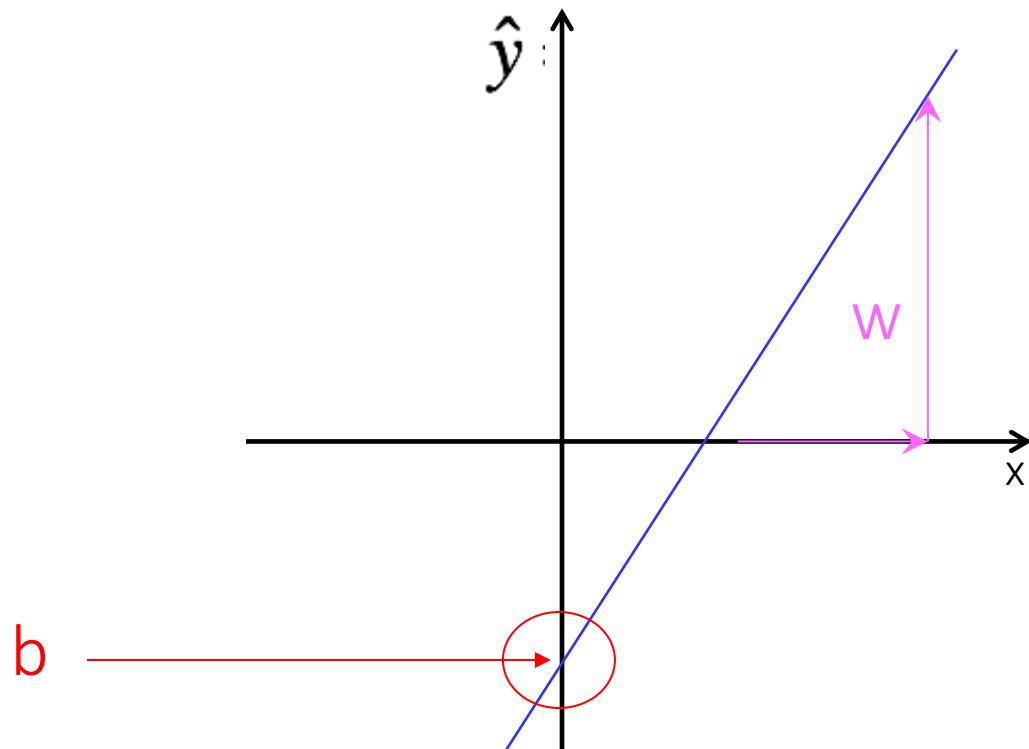
Today: Multivariate Linear Regression in a Nutshell



Review: $f(x)$ is Linear when X is single variable

- $f(x)=wx+b?$

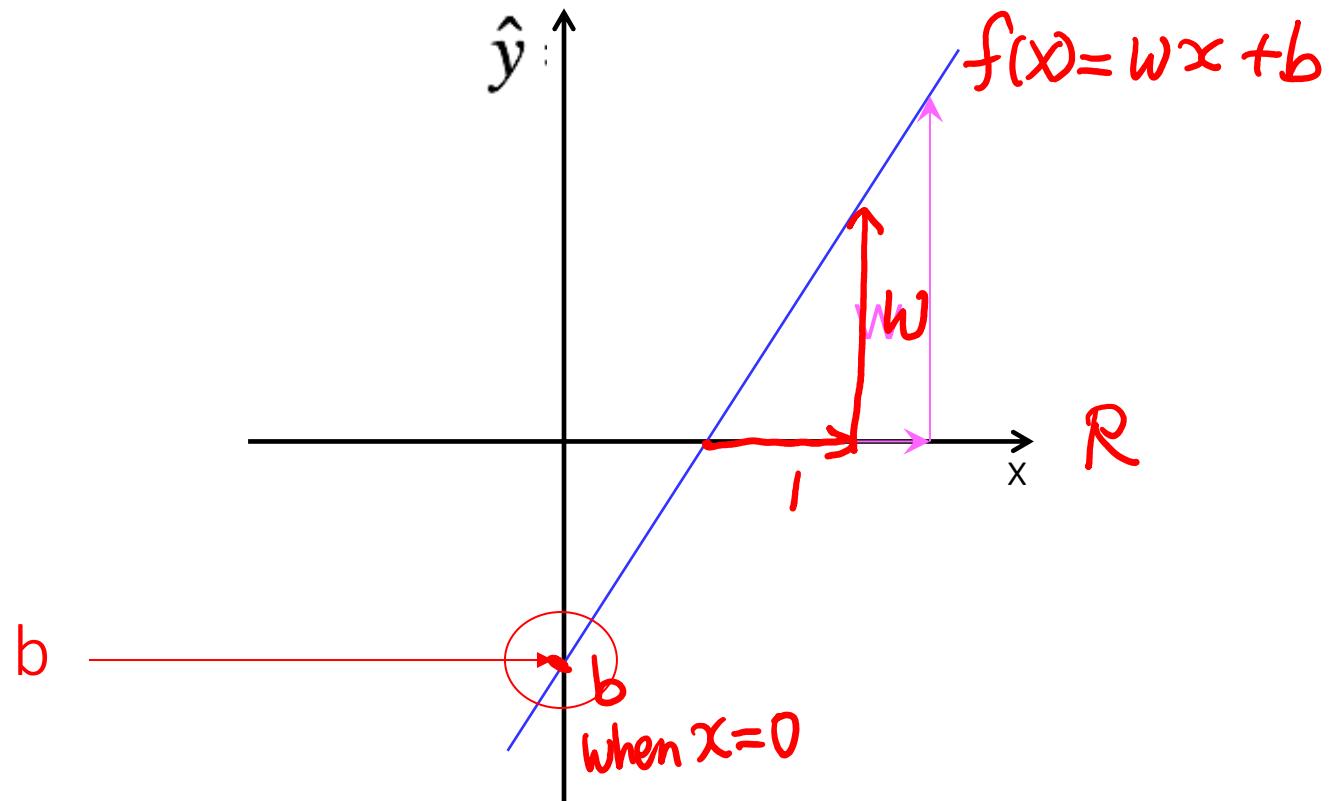
A slope of 2 (i.e. $w=2$) means that every 1-unit change in X yields a 2-unit change in Y.



Review: $f(x)$ is Linear when X is single variable

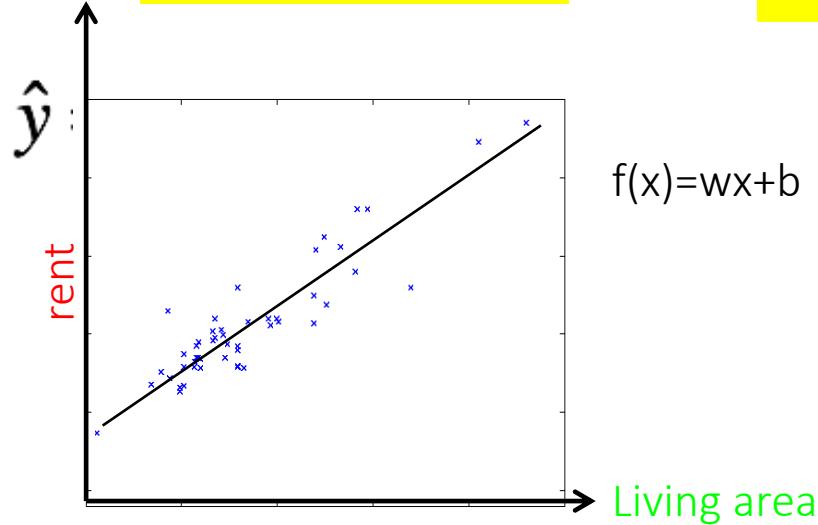
- $f(x)=wx+b?$

A slope of 2 (i.e. $w=2$) means that every 1-unit change in X yields a 2-unit change in Y.



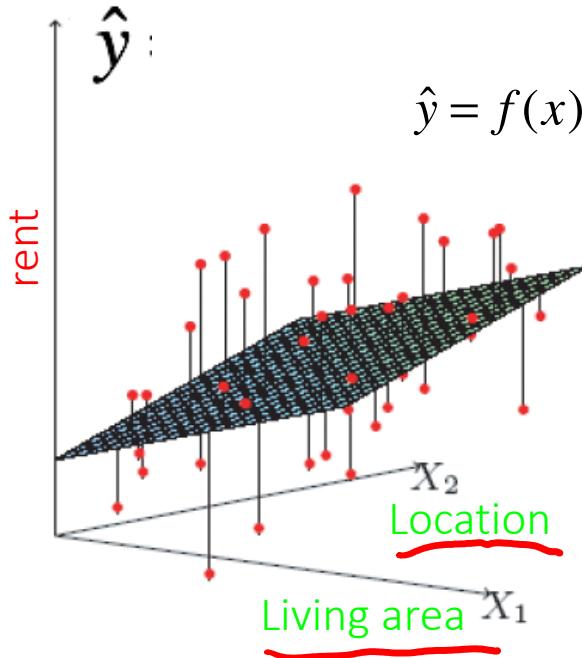
e.g., Living area as X

Linear Regression: Y as Weighted linear sum of Xs



1D case ($\mathcal{X} = \mathbb{R}$): a line

e.g., (Living area, Location) as X



Weighted sum of x_i

$\mathcal{X} = \mathbb{R}^2$: a plane

Linear Supervised Regression

$$f: X \rightarrow Y$$

$$f(x) = \theta_0 + \theta_1 x_1 + \dots + \theta_p x_p$$

Linear Regression Models

Linear Regression: Y as Weighted linear sum of Xs

$$\hat{y} = f(x) = \theta_0 + \theta_1 \underbrace{x_1}_{} + \theta_2 \underbrace{x_2}_{} + \dots$$

=> Features x_i : $\vec{x} \in \mathbb{R}^p$:

e.g., Living area, distance to campus, # bedroom ...

=> Target y :

e.g., Rent (a continuous variable)

A Concise Notation: via Vector/Matrix Product

- Represent each data sample \mathbf{x} as a column vector, plus a pseudo feature
 - We add a pseudo "feature" $x_0=1$ (this is the **intercept** term), and RE-define the feature vector to be:

$$\vec{\mathbf{x}} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}$$

- The parameter vector θ is also a [column vector]

$$\vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_p \end{bmatrix} \xrightarrow{(P+1) \times 1} \hat{y} = f(\mathbf{x}) = \vec{\mathbf{x}}^T \vec{\theta} = \theta^T \mathbf{x}$$

$$\hat{y} = f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = \begin{bmatrix} 1 & x_1 & x_2 & \dots & x_p \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_p \end{bmatrix} + \dots + \theta_p x_p$$

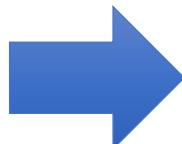
A Concise Notation: via Vector/Matrix Product

- Represent each data sample \mathbf{x} as a column vector, plus a pseudo feature
 - We add a pseudo "feature" $x_0=1$ (this is the intercept term), and RE-define the feature vector to be:

$$\mathbf{x}^T = [(\underline{x_0=1}), \underline{x_1}, \underline{x_2}, \dots, \underline{x_p}]$$

- The parameter vector θ is also a column vector

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_p \end{bmatrix}$$



$$\hat{y} = f(\mathbf{x}) \\ = \mathbf{x}^T \theta = \theta^T \mathbf{x}$$

$$\hat{y} = f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_p x_p$$

$$\hat{y} = f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Review:

- Dot (or Inner) Product of two vectors $\langle \mathbf{x}, \mathbf{a} \rangle$

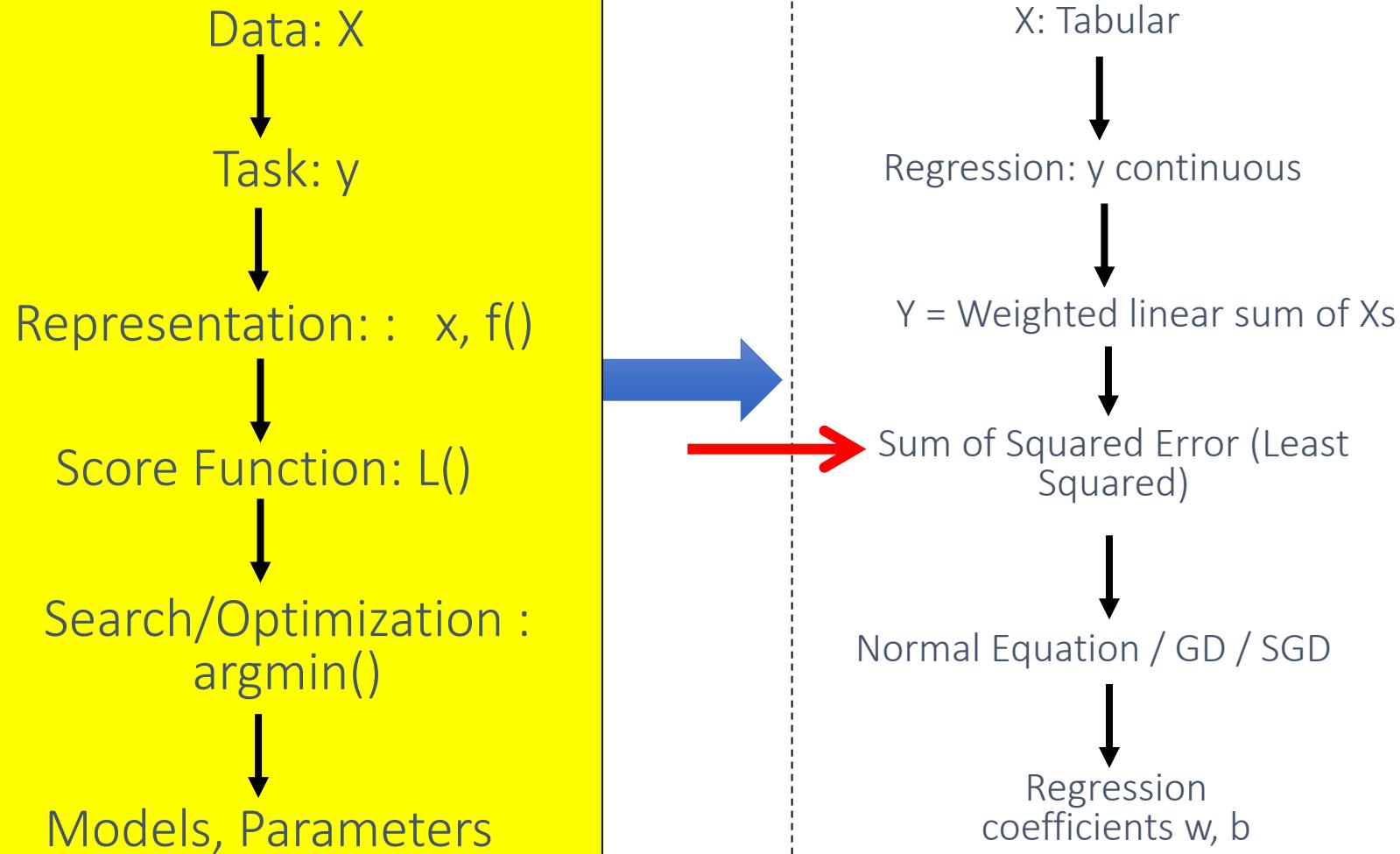
is the sum of products of elements in similar positions for the two vectors

$$\langle \mathbf{x}, \boldsymbol{\theta} \rangle = \langle \boldsymbol{\theta}, \mathbf{x} \rangle \quad \mathbf{x}^T \boldsymbol{\theta} = \boldsymbol{\theta}^T \mathbf{x}$$

Linear Regression: Y as Weighted linear sum of Xs

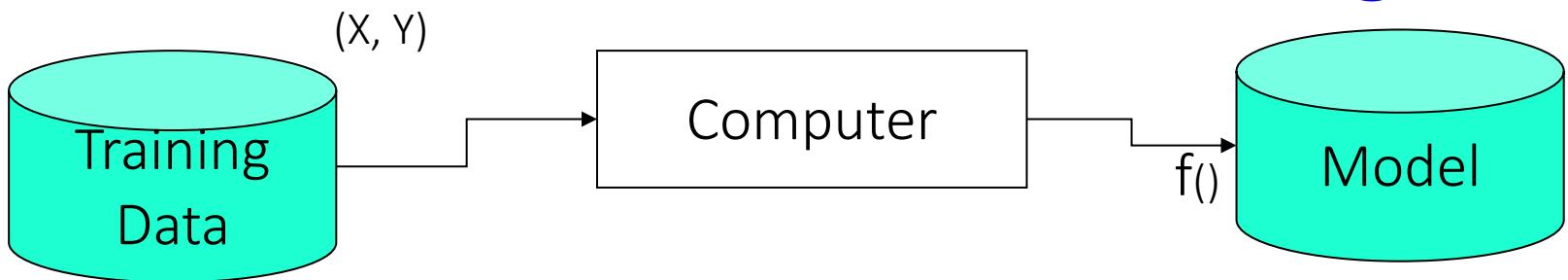
$$\begin{aligned}\hat{y} &= f(\mathbf{x}) \\ &= \mathbf{x}^T \boldsymbol{\theta} = \boldsymbol{\theta}^T \mathbf{x}\end{aligned}$$

Today: Multivariate Linear Regression in a Nutshell



Training Modes of Machine Learning

Consists of **input-output** pairs



```
# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

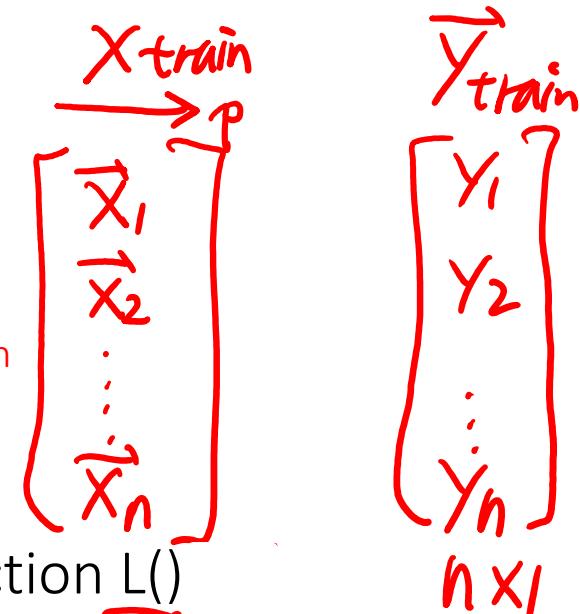
# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)
```

Basic Concepts

- Training (i.e. learning parameters w, b)
 - Training set includes
 - available examples x_1, \dots, x_n
 - available corresponding labels y_1, \dots, y_n



- Find (w, b) by minimizing loss / Cost function $\underline{\underline{L()}}$
 - (i.e. difference between y and $f(x)$ on available examples in training set)

$$(w, b) = \underset{w, b}{\operatorname{argmin}} \sum_{i=1}^n \ell(f(x_i), y_i)$$

A red bracket under the summation symbol is labeled n . A red bracket under the term $\ell(f(x_i), y_i)$ is labeled $??$.

Loss/Cost function for Regression L()

SSE: Sum of squared error

- Our goal is to search for the optimal θ that minimize the following lost /cost function:

training : $J(\theta) = \frac{1}{2} \sum_{i=1}^n (f(\mathbf{x}_i) - y_i)^2 \Rightarrow \theta^* = \operatorname{argmin}_{\theta} J(\theta)$

testing : $\hat{y} = f(\mathbf{x}) = \mathbf{x}^T \theta^* = \theta^{*T} \mathbf{x}$

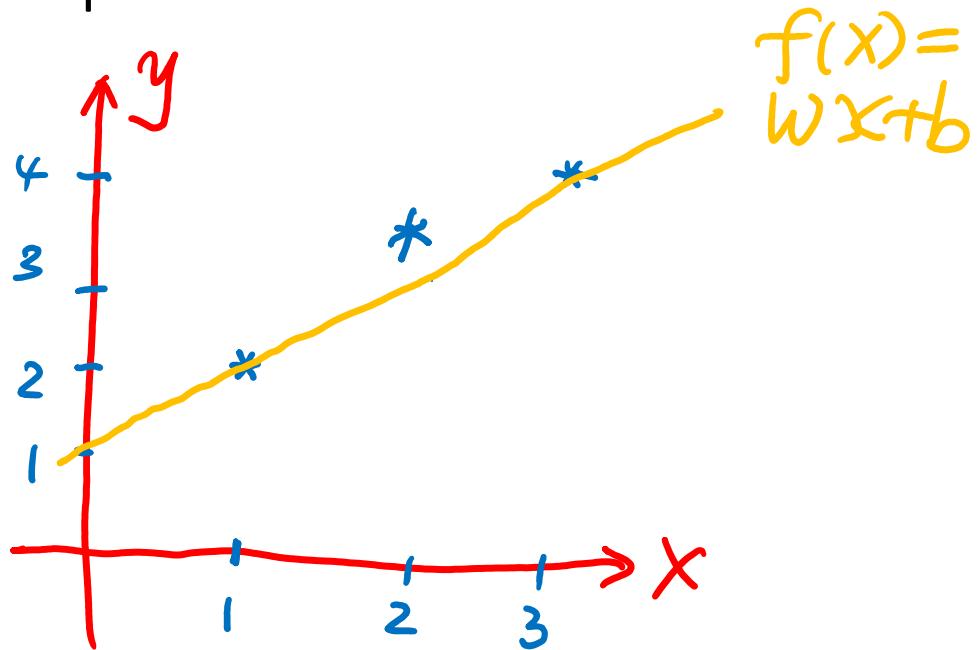
One concrete example

	x	y
s_1	1	2
s_2	2	3.3
s_3	3	4



$$f(x) = wx + b$$

$$(w^*, b^*) = \underset{w, b}{\operatorname{argmin}}$$



$$J(w, b) = \sum_{i=1}^3 (wx_i + b - y_i)^2$$

One concrete example

x_i	y_i	$f(x_i) = \hat{y}_i = wx + b$	$(wx_i + b - y_i)^2$
1	2	$w + b$	$(w + b - 2)^2$
2	3	$2w + b$	$(2w + b - 3)^2$
3	4	$3w + b$	$(3w + b - 4)^2$

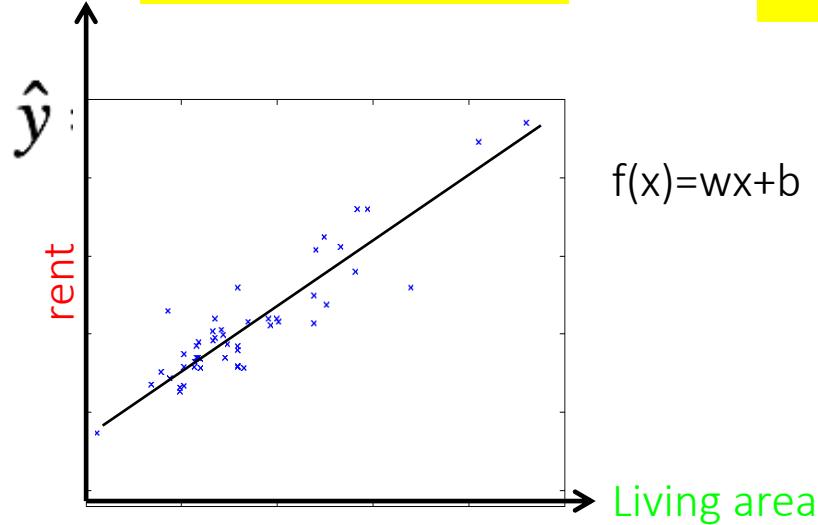
$n=3$

$$J(w, b) = \frac{1}{2} \left\{ (w+b-2)^2 + (2w+b-3)^2 + (3w+b-4)^2 \right\}$$

$$\Rightarrow ?? \quad \underset{w, b}{\operatorname{argmin}} \quad J(w, b)$$

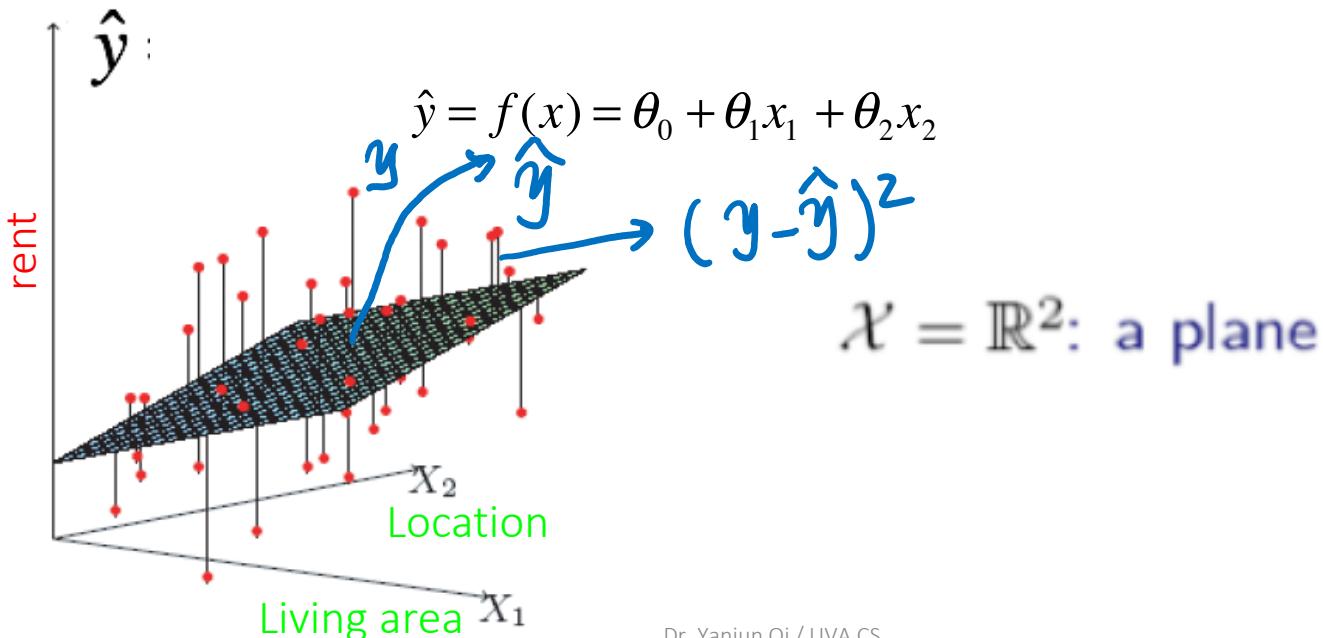
e.g., Living area as X

Linear Regression: Y as Weighted linear sum of Xs



1D case ($\mathcal{X} = \mathbb{R}$): a line

e.g., (Living area, Location) as X



$\mathcal{X} = \mathbb{R}^2$: a plane

Now the loss function: via A Concise Notation

- Using matrix form, we get the following general representation of the linear regression function:

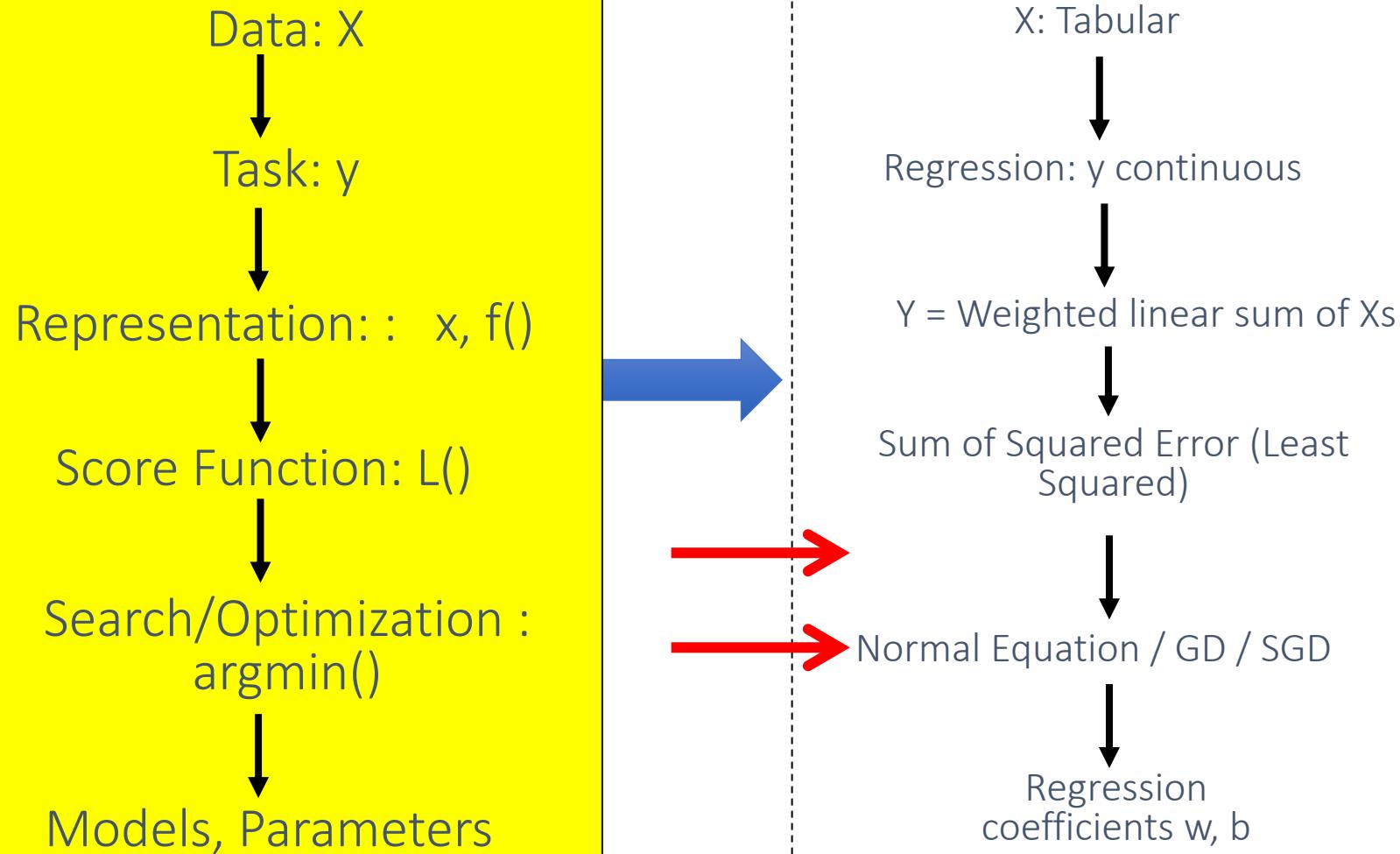
$$\mathbf{X} = \begin{bmatrix} \cdots & \mathbf{x}_1^T & \cdots \\ \cdots & \mathbf{x}_2^T & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & \mathbf{x}_n^T & \cdots \end{bmatrix}$$

$$\begin{aligned} J(\theta) &= \frac{1}{2} \sum_{i=1}^n (\mathbf{x}_i^T \theta - y_i)^2 \\ &= \frac{1}{2} (\mathbf{X}\theta - \bar{\mathbf{y}})^T (\mathbf{X}\theta - \bar{\mathbf{y}}) \\ &= \frac{1}{2} (\theta^T \mathbf{X}^T \mathbf{X}\theta - \theta^T \mathbf{X}^T \bar{\mathbf{y}} - \bar{\mathbf{y}}^T \mathbf{X}\theta + \bar{\mathbf{y}}^T \bar{\mathbf{y}}) \end{aligned}$$


Thank You



Today: Multivariate Linear Regression in a Nutshell



Default Vector Form is the Column Form

$$\vec{x}_i = \begin{bmatrix} 1 \\ x_{i,1} \\ x_{i,2} \\ x_{i,3} \\ \vdots \\ x_{i,p} \end{bmatrix} \quad (p+1) \times 1$$

$$\vec{x}_i^T = [1, x_{i,1}, x_{i,2}, \dots, x_{i,p}]$$

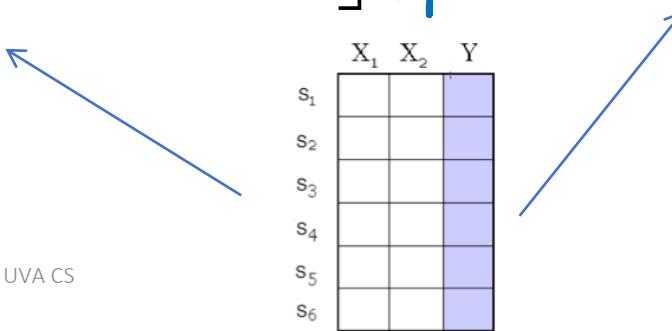
$$X = \begin{bmatrix} \cdots & \mathbf{x}_1^T & \cdots \\ \cdots & \mathbf{x}_2^T & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & \mathbf{x}_n^T & \cdots \end{bmatrix}$$

Training Set in Matrix Form

- the whole Training set (with n samples) as matrix form :

$$\mathbf{X} = \begin{bmatrix} \cdots & \mathbf{x}_1^T & \cdots \\ \cdots & \mathbf{x}_2^T & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & \mathbf{x}_n^T & \cdots \end{bmatrix} = \begin{bmatrix} x_{1,0} & x_{1,1} & \cdots & x_{1,p} \\ x_{2,0} & x_{2,1} & \cdots & x_{2,p} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n,0} & x_{n,1} & \cdots & x_{n,p} \end{bmatrix} \quad \bar{\mathbf{y}}_{train} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$\xrightarrow{\mathcal{P}}$
 $\downarrow h$
 $\downarrow h \times p$



	X_1	X_2	Y
s_1			
s_2			
s_3			
s_4			
s_5			
s_6			

Training Set in Matrix Form

- the whole Training set (with n samples) as matrix form :

$$\mathbf{X}\boldsymbol{\theta} = \begin{bmatrix} \cdots & \mathbf{x}_1^T & \cdots \\ \cdots & \mathbf{x}_2^T & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & \mathbf{x}_n^T & \cdots \end{bmatrix} = \begin{bmatrix} \boldsymbol{\theta} \\ \vdots \\ \boldsymbol{\theta} \end{bmatrix} = \begin{bmatrix} x_{1,0} & x_{1,1} & \cdots & x_{1,p} \\ x_{2,0} & x_{2,1} & \cdots & x_{2,p} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n,0} & x_{n,1} & \cdots & x_{n,p} \end{bmatrix} \quad \bar{y}_{train} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

~~$\sum_{n \times (p+1)} (\mathbf{x}^T \boldsymbol{\theta})$~~

	X ₁	X ₂	Y
s ₁			
s ₂			
s ₃			
s ₄			
s ₅			
s ₆			

Cont: Loss function: via A Concise Notation

- Using matrix form, we get the following general representation of the linear regression function:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^n (\mathbf{x}_i^T \theta - y_i)^2$$

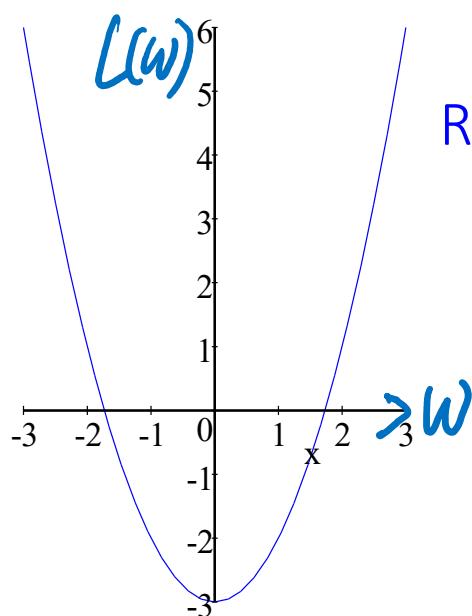
$$= \frac{1}{2} (\mathbf{X}\theta - \bar{\mathbf{y}})^T (\mathbf{X}\theta - \bar{\mathbf{y}})$$

$$= \frac{1}{2} (\theta^T \mathbf{X}^T \mathbf{X}\theta - \theta^T \mathbf{X}^T \bar{\mathbf{y}} - \bar{\mathbf{y}}^T \mathbf{X}\theta + \bar{\mathbf{y}}^T \bar{\mathbf{y}})$$

$$\mathbf{X} = \begin{bmatrix} \cdots & \mathbf{x}_1^T & \cdots \\ \cdots & \mathbf{x}_2^T & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & \mathbf{x}_n^T & \cdots \end{bmatrix} \theta$$

$$\mathbf{a} = (\mathbf{X}\theta - \bar{\mathbf{y}}) = \underbrace{\mathbf{X}\theta}_{n \times p} - \underbrace{\bar{\mathbf{y}}}_{p \times 1} = \begin{bmatrix} \mathbf{x}_1^T \theta - y_1 \\ \mathbf{x}_2^T \theta - y_2 \\ \vdots \\ \mathbf{x}_n^T \theta - y_n \end{bmatrix}$$

$$\bar{\mathbf{a}}^T \bar{\mathbf{a}} = \sum_{i=1}^n a_i^2$$



Review (IV): Derivative of a Quadratic Function

$$\underline{L(w)} = w^2 - 3$$

\downarrow

$$L'(w) = 2w = 0$$

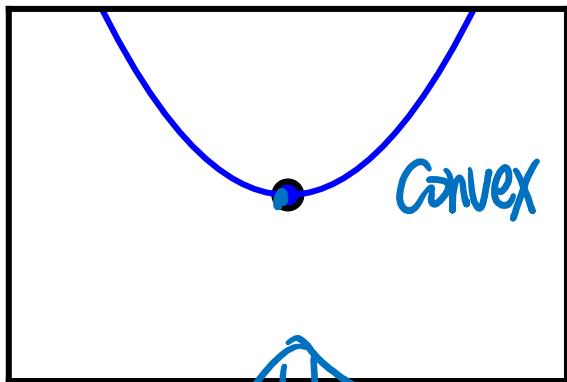
$$\operatorname{argmin}_w L(w) \quad = (L'(w)=0) \quad || \quad w^*$$

This quadrative (convex) function is minimized @ the unique point whose derivative (slope) is zero.

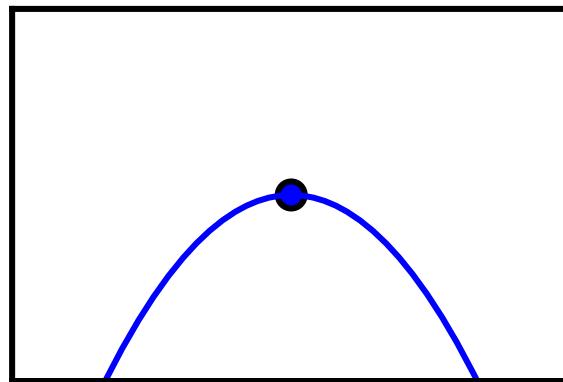
→ When we find zeros of the derivative of this function, we also find the minima (or maxima) of that function.

Critical Points

Minimum



Maximum



Saddle point

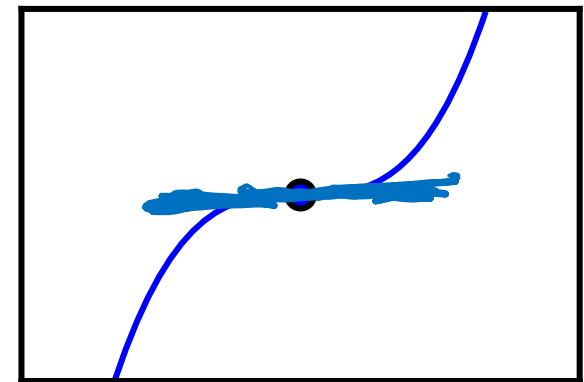


Figure 4.2

$$J(\theta) = \sum_{i=1}^n (\vec{x}_i^T \vec{\theta} - y_i)^2$$

Find best θ via Solving $\nabla_{\theta}J(\theta) = 0$

- Write the cost function in matrix form:

$$\begin{aligned} J(\theta) &= \frac{1}{2} \sum_{i=1}^n (\mathbf{x}_i^T \theta - y_i)^2 \\ &= \frac{1}{2} (\mathbf{X}\theta - \bar{\mathbf{y}})^T (\mathbf{X}\theta - \bar{\mathbf{y}}) \\ &= \frac{1}{2} (\theta^T \mathbf{X}^T \mathbf{X}\theta - \theta^T \mathbf{X}^T \bar{\mathbf{y}} - \bar{\mathbf{y}}^T \mathbf{X}\theta + \bar{\mathbf{y}}^T \bar{\mathbf{y}}) \end{aligned}$$

To minimize $J(\theta)$, take derivative (gradient) and set to zero:

$$\Rightarrow \boxed{\mathbf{X}^T \mathbf{X}\theta = \mathbf{X}^T \bar{\mathbf{y}}}$$

The normal equations

Closed form
solution

$$\theta^* = \boxed{\downarrow \left(\mathbf{X}^T \mathbf{X} \right)^{-1} \mathbf{X}^T \bar{\mathbf{y}}}$$

Method I: normal equations to minimize the loss

- Write the cost function in matrix form:

$$\begin{aligned} J(\theta) &= \frac{1}{2} \sum_{i=1}^n (\mathbf{x}_i^\top \theta - y_i)^2 \\ &= \frac{1}{2} (\mathbf{X}\theta - \bar{\mathbf{y}})^\top (\mathbf{X}\theta - \bar{\mathbf{y}}) \\ &= \frac{1}{2} \left(\theta^\top \mathbf{X}^\top \mathbf{X}\theta - \theta^\top \mathbf{X}^\top \bar{\mathbf{y}} - \bar{\mathbf{y}}^\top \mathbf{X}\theta + \bar{\mathbf{y}}^\top \bar{\mathbf{y}} \right) \end{aligned}$$

$\theta^\top \mathbf{X}^\top \bar{\mathbf{y}}$
//
 $2\mathbf{X}^\top \mathbf{X}\theta$ $\mathbf{1} \times \mathbf{n} \times \mathbf{1}$

$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{2} (2\mathbf{X}^\top \mathbf{X}\theta - 2\mathbf{X}^\top \bar{\mathbf{y}})$
 $= \mathbf{X}^\top \mathbf{X}\theta - \mathbf{X}^\top \bar{\mathbf{y}}$
 $= \mathbf{X}^\top \mathbf{X}\theta - \mathbf{X}^\top \bar{\mathbf{y}}$

To minimize $J(\theta)$, take derivative and set to zero:

$$\nabla_{\theta} J(\theta) = 0 \Rightarrow X^\top X\theta = X^\top \bar{y}$$

The normal equations

Closed form
solution

$$\theta^* = (X^\top X)^{-1} X^\top \bar{y}$$

Detailed
Derivation is
Optional /
Extra

See handout 4.1 + 4.3 \Rightarrow matrix calculus, partial deri \Rightarrow Gradient

$$\nabla_{\theta} (\theta^T X^T X \theta) = 2X^T X \theta \quad (\text{P24})$$

$$\nabla_{\theta} (-2\theta^T X^T Y) = -2X^T Y \quad (\text{P24})$$

$$\nabla_{\theta} (Y^T Y) = 0$$

$$\Rightarrow \nabla_{\theta} J(\theta) = \boxed{X^T X \theta - X^T Y}$$

This Slide is
Optional /
Extra

$$\Rightarrow \text{Hessian } H(J(\theta)) = \frac{\partial \nabla_{\theta} J(\theta)}{\partial \theta} = \frac{\partial (X^T X \theta)}{\partial \theta} =$$

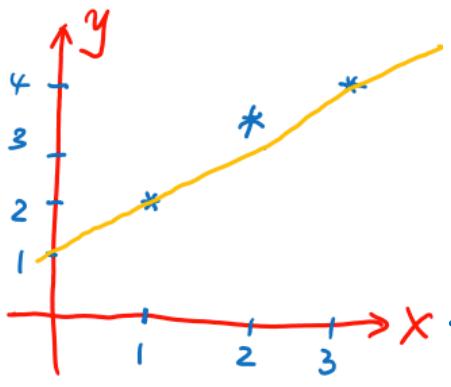
Gram matrix $X^T X$

Advanced / Optional / More in
Extra Slides

One concrete example

$$\Rightarrow J(w, b) = \frac{1}{2}((w+b-2)^2 + (2w+b-3)^2)$$

$$\Rightarrow \begin{cases} \frac{\partial J(w, b)}{\partial w} = (w+b-2) + (2w+b-3) \cdot 2 = 0 \\ \frac{\partial J(w, b)}{\partial b} = w+b-2 + (2w+b-3) = 0 \end{cases}$$



$$\begin{cases} 5w + 3b - 8 = 0 \\ 3w + 2b - 5 = 0 \end{cases} \Rightarrow \begin{cases} 10w + 6b - 16 = 0 \\ 9w + 6b - 15 = 0 \end{cases}$$

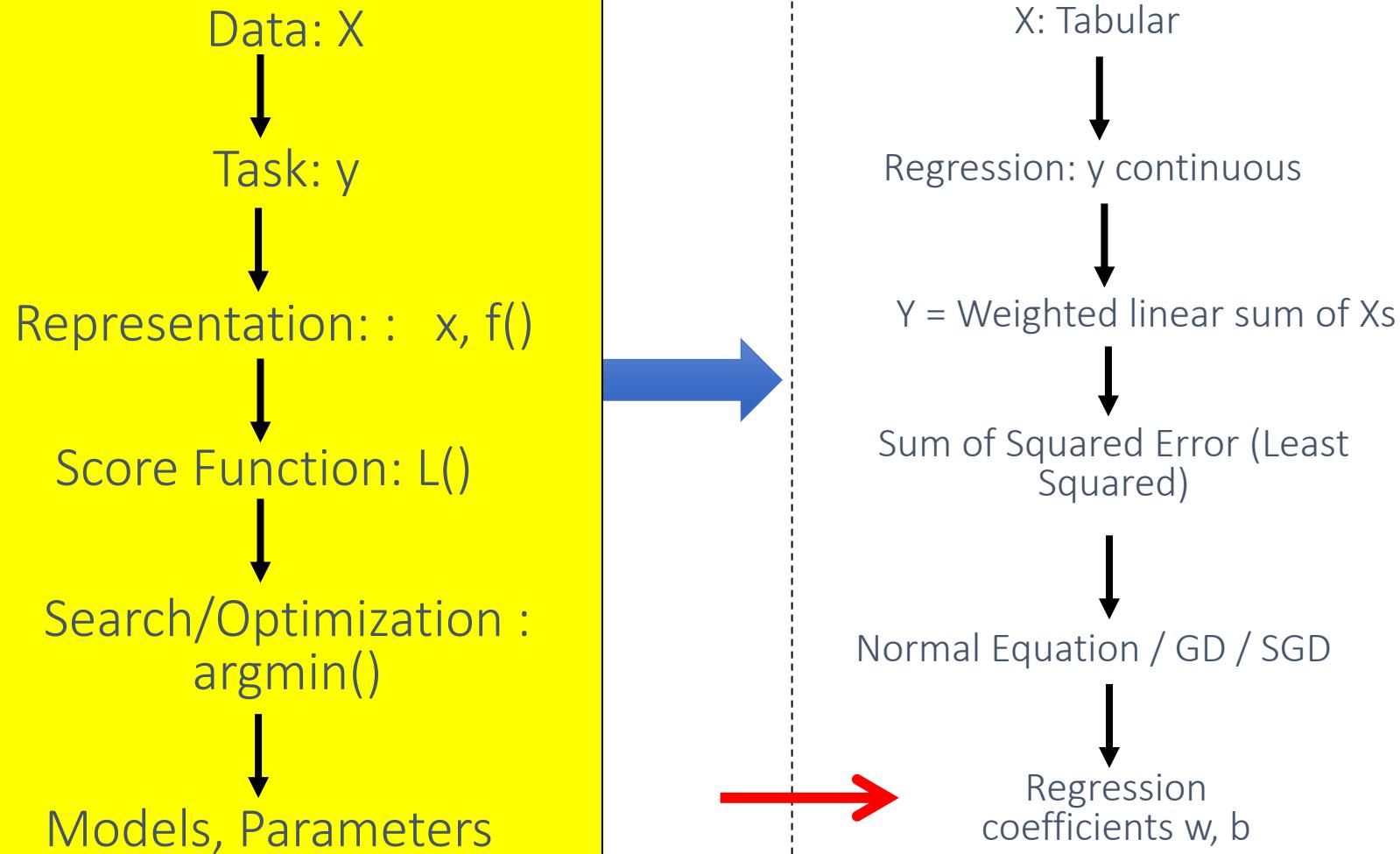
$$w = 1, \quad b = \frac{1}{2}(5 - 3w) = 1$$

we solve the matrix equation via Gaussian Elimination

Thank You



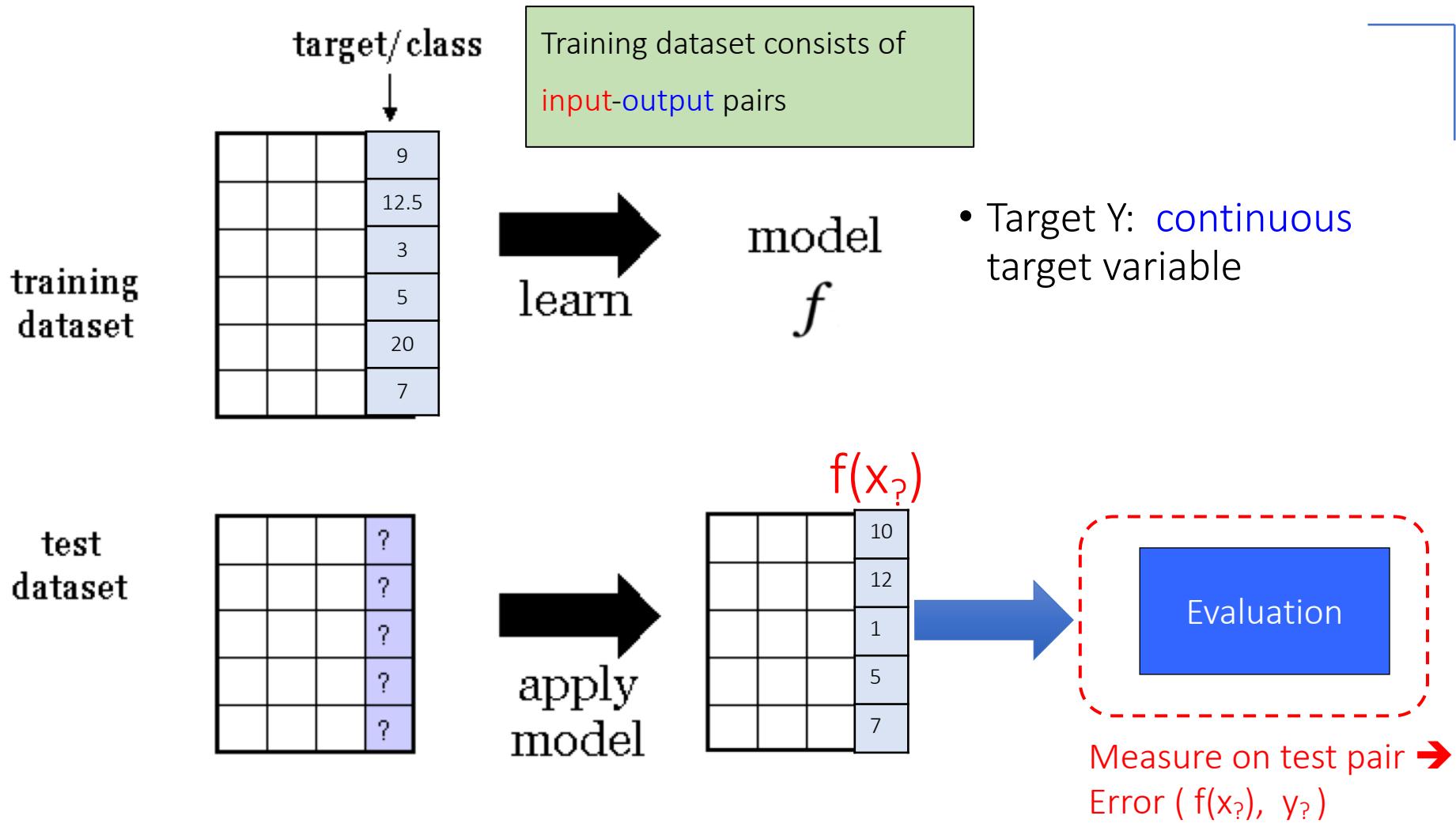
Today: Multivariate Linear Regression in a Nutshell



We aim to make the learned model

- 
- 1. Generalize Well
 - 2. Computational Scalable and Efficient
 - 3. Robust / Trustworthy / Interpretable
 - Especially for some domains, this is about trust!

How to know the regression program works well: Metrics on Prediction on Test Inputs



Ways for Measuring Regression Predictions: e.g., Correlation Coefficient

- Pearson correlation coefficient

$$r(x,y) = \frac{\sum_{i=1}^m (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^m (x_i - \bar{x})^2 \times \sum_{i=1}^m (y_i - \bar{y})^2}}$$

where $\bar{x} = \frac{1}{m} \sum_{i=1}^m x_i$ and $\bar{y} = \frac{1}{m} \sum_{i=1}^m y_i$.

$$|r(x,y)| \leq 1$$

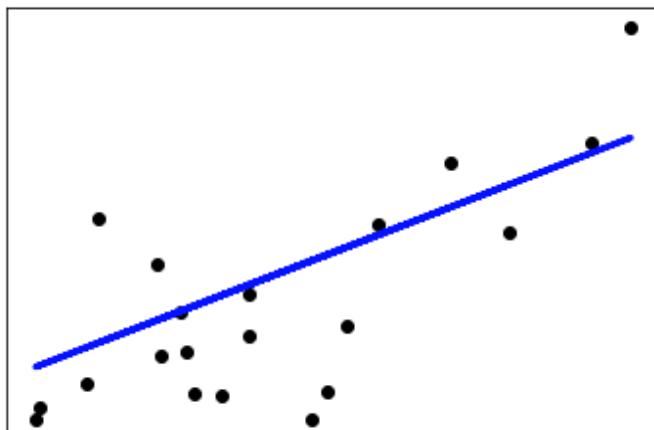
- Measuring the **linear correlation** between two sequences, x and y,
- Generating a value between +1 and -1 inclusive, where 1 is total positive **correlation**, 0 is no **correlation**, and -1 is total negative **correlation**.

- For regression: $r(\vec{y}_{\text{predicted}}, \vec{y}_{\text{known}})$

```
Mean squared error: 2548.07  
Coefficient of determination: 0.47
```

```
# Train the model using the training sets  
regr.fit(diabetes_X_train, diabetes_y_train)  
  
# Make predictions using the testing set  
diabetes_y_pred = regr.predict(diabetes_X_test)  
  
# The coefficients  
print('Coefficients: \n', regr.coef_)  
# The mean squared error  
print('Mean squared error: %.2f'  
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))  
# The coefficient of determination: 1 is perfect prediction  
print('Coefficient of determination: %.2f'  
      % r2_score(diabetes_y_test, diabetes_y_pred))
```

```
# Plot outputs  
plt.scatter(diabetes_X_test, diabetes_y_test, color='black')  
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)  
  
plt.xticks(())  
plt.yticks(())  
  
plt.show()
```



We aim to make the learned model

- 1. Generalize Well
- • 2. Computational Scalable and Efficient
- 3. Robust / Trustworthy / Interpretable
 - Especially for some domains, this is about trust!

Why we Prefer Concise Vector/Matrix Form?

Training: Closed form solution

$$\theta^* = (X_{train}^T X_{train})^{-1} (X_{train}^T \vec{y}_{train})$$

Testing: on multiple Test Inputs

$$\hat{\vec{y}}_{test} = X_{test} \theta^*$$

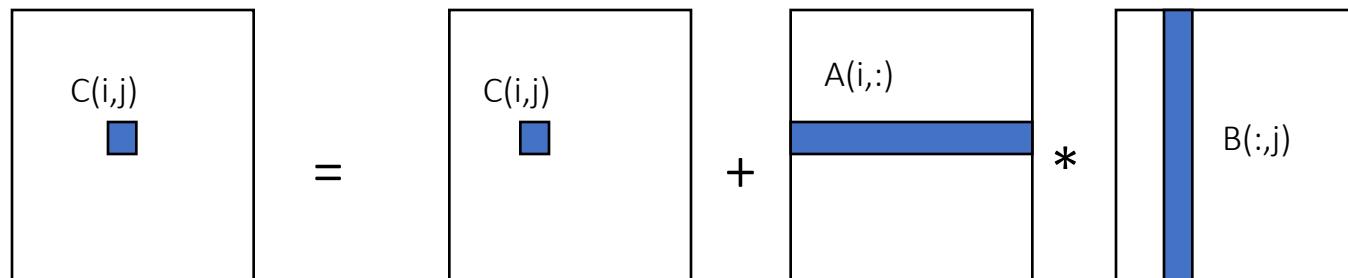
Extra: Naïve Matrix Multiply

{implements $C = C + A * B$ }

```
for i = 1 to n
    for j = 1 to n
        for k = 1 to n
            C(i,j) = C(i,j) + A(i,k) * B(k,j)
```

Algorithm has $2*n^3 = O(n^3)$ Flops and
operates on $3*n^2$ words of memory

Extra:



Computational Cost (Naïve Way)

$$\hat{\theta}^* = \left(\sum_{p \times n}^T \sum_{n \times p} \right)^{-1} \sum^T \vec{y}$$

$O(n)$
 $X_i^T X_j$
 $|x_n \ n \times 1|$
 $\curvearrowright p^2$
 $O(p^2 n)$

$$X^T X : O(p^2 n)$$

$$(X^T X)^{-1} : O(p^3)$$

$$O(np^2 + p^3)$$

When $n \gg p$,

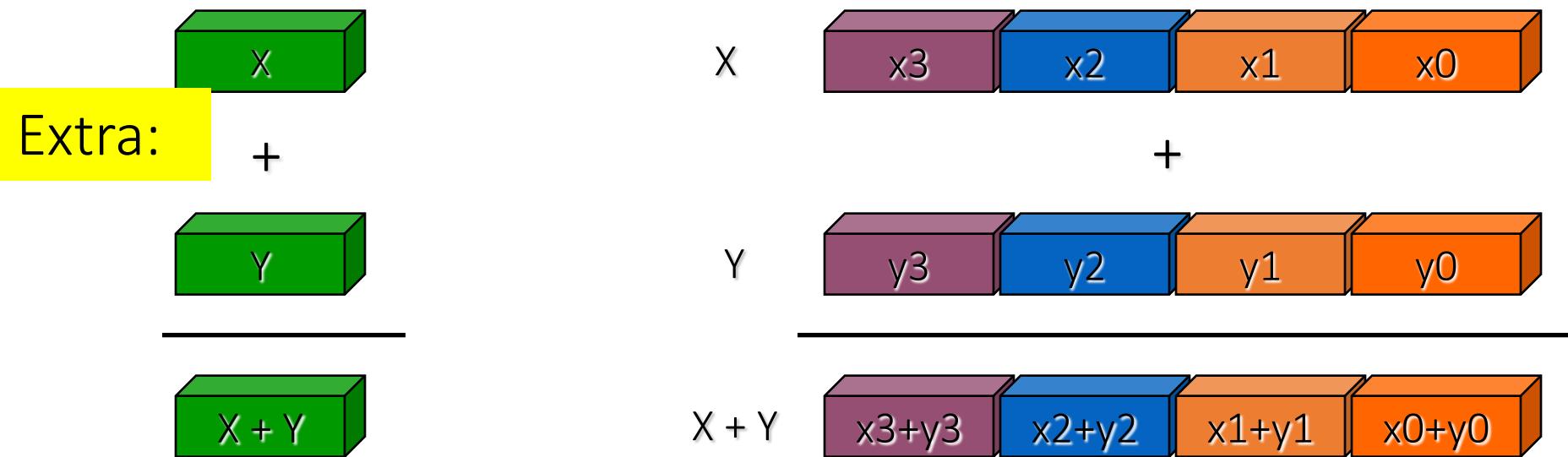
matrix multi:
slower than inversion

Many architecture details and Algorithm details to consider

- Data parallelization through CPU SIMD / Multithreading/ GPU parallelization /
- Memory hierarchical / locality
- Better algorithms, like Strassen's Matrix Multiply and many others

SIMD: Single Instruction, Multiple Data

- Scalar processing
 - traditional mode
 - one operation produces one result
- SIMD processing
 - with SSE / SSE2
 - SSE = streaming SIMD extensions
 - one operation produces multiple results

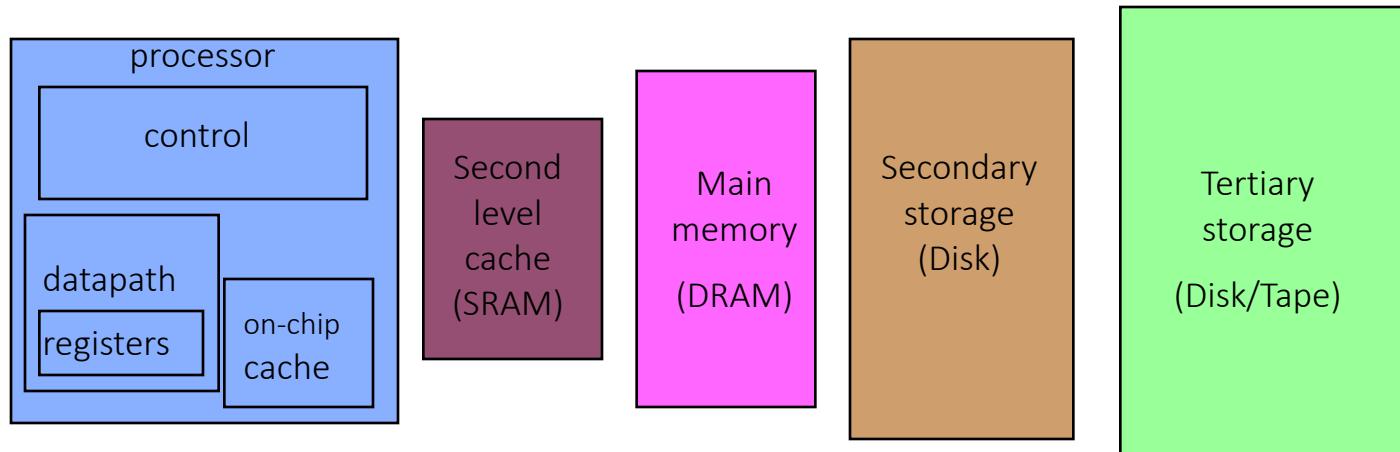


Slide Source: Alex Klimovitski & Dean Macri, Intel Corporation

Memory Hierarchy

- Most programs have a high degree of **locality** in their accesses
 - **spatial locality**: accessing things nearby previous accesses
 - **temporal locality**: reusing an item that was previously accessed
- Memory hierarchy tries to exploit locality to improve average

Extra:



Speed	1ns	10ns	100ns	10ms	10sec
Size	KB	MB	GB	TB	PB

The following complexity figures assume that arithmetic with individual elements has complexity $O(1)$, as is the case with fixed-precision operations on a finite field.

Operation	Input	Output	Algorithm	Complexity
Matrix multiplication	Two $n \times n$ matrices	One $n \times n$ matrix	Schoolbook matrix multiplication	$O(n^3)$
			Strassen algorithm	$O(n^{2.807})$
			Coppersmith–Winograd algorithm	$O(n^{2.376})$
			Optimized CW-like algorithms ^{[14][15][16]}	$O(n^{2.373})$
Matrix multiplication	One $n \times m$ matrix & one $m \times p$ matrix	One $n \times p$ matrix	Schoolbook matrix multiplication	$O(nmp)$
Matrix inversion*	One $n \times n$ matrix	One $n \times n$ matrix	Gauss–Jordan elimination	$O(n^3)$
			Strassen algorithm	$O(n^{2.807})$
			Coppersmith–Winograd algorithm	$O(n^{2.376})$
			Optimized CW-like algorithms	$O(n^{2.373})$
Singular value decomposition	One $m \times n$ matrix	One $m \times m$ matrix, one $m \times n$ matrix, & one $n \times n$ matrix		$O(mn^2)$ ($m \leq n$)
		One $m \times r$ matrix, one $r \times r$ matrix, & one $n \times r$ matrix		
Determinant	One $n \times n$ matrix	One number	Laplace expansion	$O(n!)$
			Division-free algorithm ^[17]	$O(n^4)$
			LU decomposition	$O(n^3)$
			Bareiss algorithm	$O(n^3)$
			Fast matrix multiplication ^[18]	$O(n^{2.373})$
Back substitution	Triangular matrix	n solutions	Back substitution ^[19]	$O(n^2)$

Extra:

Basic Linear Algebra Subroutines (BLAS) →

numpy: a wrapper library of BLAS

- Industry standard interface (evolving)
 - [www.netlib.org/blas](http://www.netlib.orgblas), [www.netlib.org/blasblast--forum](http://www.netlib.orgblasblast--forum)
- Vendors, others supply optimized implementations
- History
 - BLAS1 (1970s):
 - vector operations: dot product, saxpy ($y=a*x+y$), etc
 - $m=2*n$, $f=2*n$, $q = f/m$ = computational intensity ~ 1 or less
 - BLAS2 (mid 1980s)
 - matrix-vector operations: matrix vector multiply, etc
 - $m=n^2$, $f=2*n^2$, $q \sim 2$, less overhead
 - somewhat faster than BLAS1
 - BLAS3 (late 1980s)
 - matrix-matrix operations: matrix matrix multiply, etc
 - $m \leq 3n^2$, $f=O(n^3)$, so $q=f/m$ can possibly be as large as n , so BLAS3 is potentially much faster than BLAS2
- Good algorithms use BLAS3 when possible (LAPACK & ScaLAPACK)
 - See [www.netlib.org/{lapack,scalapack}](http://www.netlib.org/{lapack,scalapack)

Functionality [edit]

BLAS functionality is categorized into three sets of routines called "levels", which correspond to both the chronological order of definition and publication, as well as the degree of the polynomial in the complexities of algorithms; Level 1 BLAS operations typically take linear time, $O(n)$, Level 2 operations quadratic time and Level 3 operations cubic time.^[18] Modern BLAS implementations typically provide all three levels.

Level 1 [edit]

This level consists of all the routines described in the original presentation of BLAS (1979),^[1] which defined only *vector operations* on strided arrays: dot products, vector norms, a generalized vector addition of the form

$$\mathbf{y} \leftarrow \alpha \mathbf{x} + \mathbf{y}$$

(called "axpy") and several other operations.

Versions of BLAS compared: BLAS library from the Netlib Repository, ATLAS library, Intel-MKL library, AMD ACML Library and Goto BLAS.

Level 2 [edit]

This level contains *matrix-vector operations* including, among other things, matrix-vector multiplication (gemv):

$$\mathbf{y} \leftarrow \alpha \mathbf{A} \mathbf{x} + \beta \mathbf{y}$$

as well as a solver for \mathbf{x} in the linear equation

$$\mathbf{T} \mathbf{x} = \mathbf{y}$$

with \mathbf{T} being triangular. Design of the Level 2 BLAS started in 1984, with release in 1988.^[19] The Level 2 subroutines are especially intended to improve performance of programs using BLAS on *vector processors*, where Level 1 BLAS are suboptimal because they hide the matrix-vector nature of the operations from the compiler.

Level 3 [edit]

This level, formally published in 1990,^[18] contains *matrix-matrix operations* "general matrix multiplication" (gemm), of the form

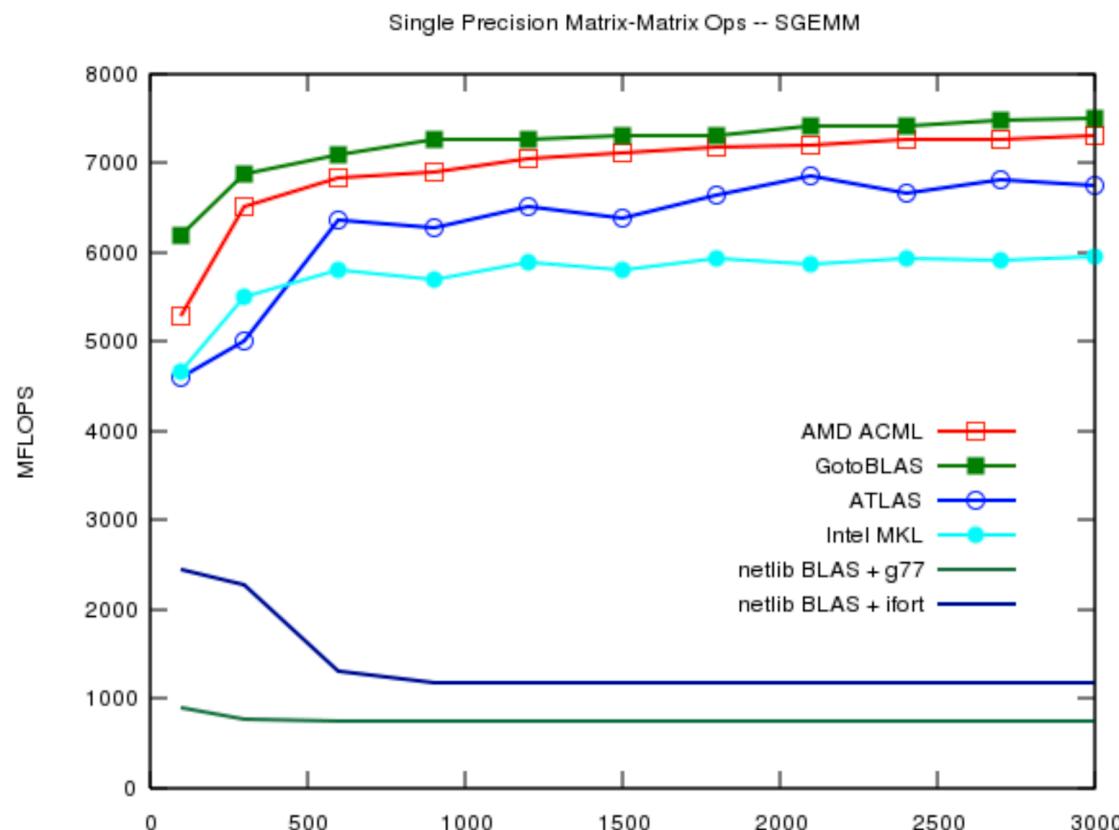
$$\mathbf{C} \leftarrow \alpha \mathbf{A} \mathbf{B} + \beta \mathbf{C}$$

where \mathbf{A} and \mathbf{B} can optionally be transposed or hermitian-conjugated inside and all three matrices may be strided. The ordinary matrix multiplication $\mathbf{A} \mathbf{B}$ is performed by setting α to one and \mathbf{C} to an all-zeros matrix of the appropriate size.

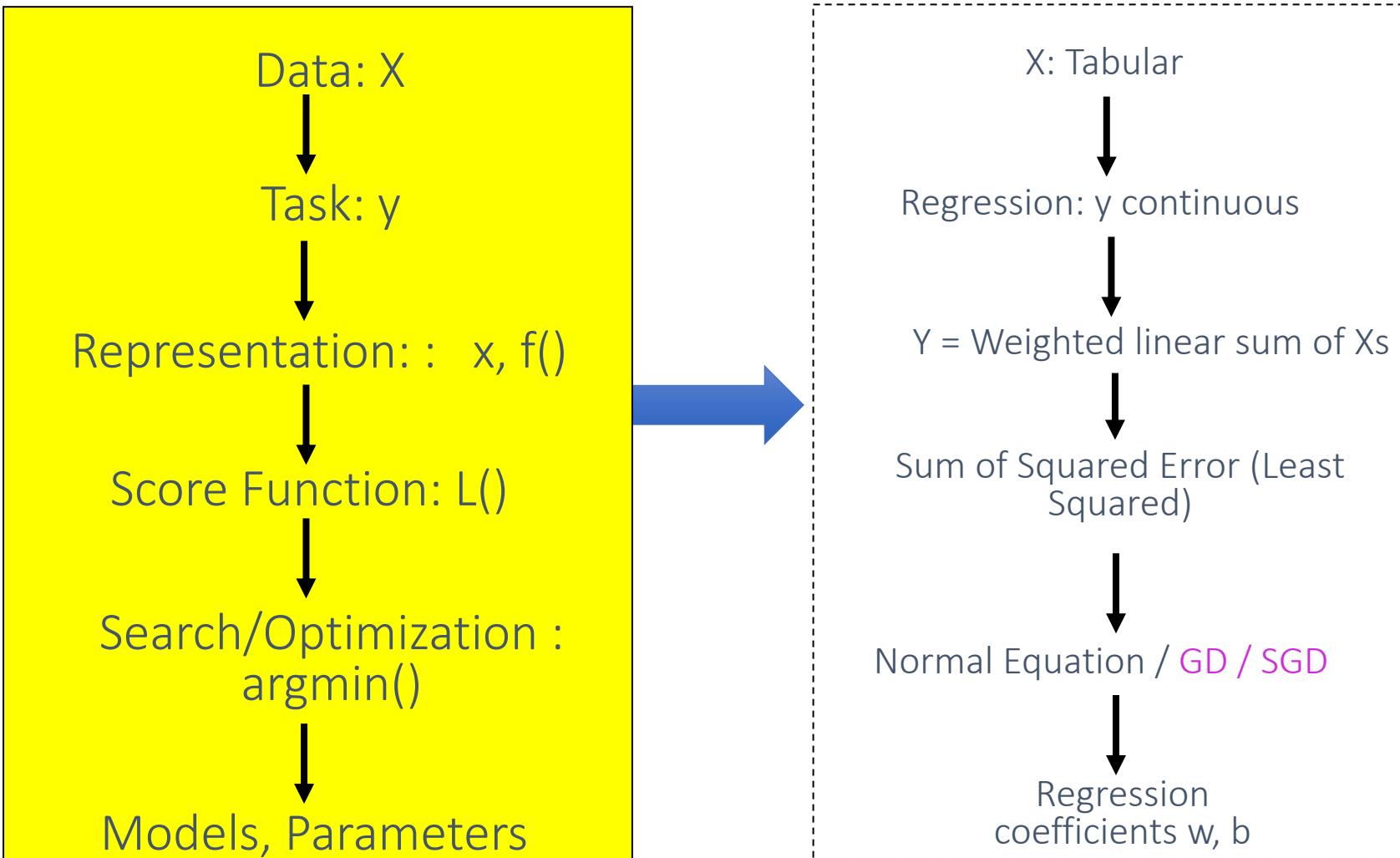
Also included in Level 3 are routines for solving

$$\mathbf{B} \leftarrow \alpha \mathbf{T}^{-1} \mathbf{B}$$

where \mathbf{T} is a triangular matrix, among other functionality.



Recap : Multivariate Linear Regression **in a Nutshell**



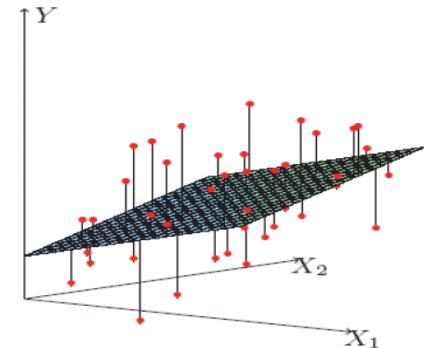
References

- Big thanks to Prof. Eric Xing @ CMU for allowing me to reuse some of his slides
 - <http://www.cs.cmu.edu/~zkolter/course/15-884/linalg-review.pdf>
 - Prof. Alexander Gray's slides

EXTRA

In Case you are interested in more advanced details!

Probabilistic Interpretation of Linear Regression (Extra)



- Let us assume that the target variable and the inputs are related by the equation:

$$y_i = \theta^T \mathbf{x}_i + \varepsilon_i$$

error data on each point

where ε is an error term of unmodeled effects or random noise

- Now assume that ε follows a Gaussian $N(0, \sigma)$, then we have:

$$p(y_i | x_i; \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \theta^T \mathbf{x}_i)^2}{2\sigma^2}\right)$$

- By iid (among samples) assumption:

Many more variations of LinearR from this perspective, e.g. binomial / poisson (LATER)

$$L(\theta) = \prod_{i=1}^n p(y_i | x_i; \theta) = \left(\frac{1}{\sqrt{2\pi}\sigma} \right)^n \exp\left(-\frac{\sum_{i=1}^n (y_i - \theta^T \mathbf{x}_i)^2}{2\sigma^2}\right)$$

Review (I):

- Sum the Squared Elements of a Vector equals
Vector dot product to itself

$$\vec{a} = \sum \theta - y$$

$$\mathbf{a} = \begin{bmatrix} 5 \\ 2 \\ 8 \end{bmatrix}$$

$$J(\theta) = \frac{1}{2} \sum_{i=1}^n (\mathbf{x}_i^T \theta - y_i)^2$$

$$\mathbf{a}^T = \begin{bmatrix} 5 & 2 & 8 \end{bmatrix}$$

$$\vec{a}^T \vec{a} = \sum_{i=1}^n a_i^2$$

$$\mathbf{a}^T \mathbf{a} = \begin{bmatrix} 5 & 2 & 8 \end{bmatrix} \begin{bmatrix} 5 \\ 2 \\ 8 \end{bmatrix} = 5^2 + 2^2 + 8^2 = 93$$

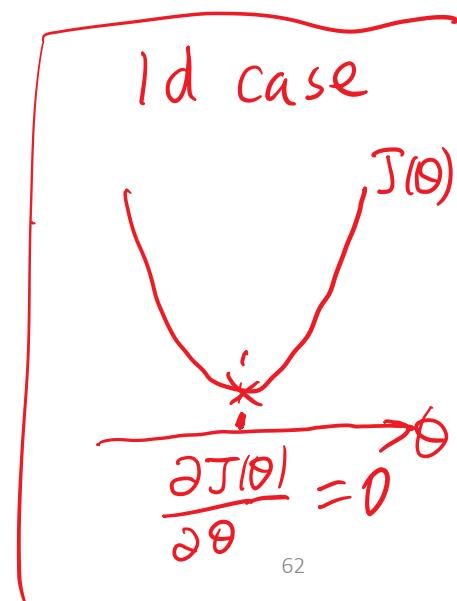
Review(I) :

$$a = \begin{bmatrix} \mathbf{x}_1^T \theta - y_1 \\ \mathbf{x}_2^T \theta - y_2 \\ \vdots \\ \mathbf{x}_n^T \theta - y_n \end{bmatrix} = X\theta - \vec{y}$$

$$\mathbf{a}^\top \mathbf{a} = 2J(\theta) = \sum_{i=1}^n (\mathbf{x}_i^T \theta - y_i)^2$$

$$\begin{aligned}
 J(\theta) &= (\underline{x}\theta - y)^T (\underline{x}\theta - y) \frac{1}{2} \\
 &= ((x\theta)^T - y^T)(\underline{x}\theta - y) \frac{1}{2} \\
 &= (\theta^T \underline{x}^T - y^T)(\underline{x}\theta - y) \frac{1}{2} \\
 &= (\underbrace{\theta^T \underline{x}^T \underline{x}\theta - \theta^T \underline{x}^T y - y^T \underline{x}\theta + y^T y}_{\text{since } \theta^T \underline{x}^T y = y^T \underline{x}\theta}) \frac{1}{2} \\
 &= (\underbrace{\theta^T \underline{x}^T \underline{x}\theta}_{\langle x\theta, y \rangle} - \underbrace{2\theta^T \underline{x}^T y}_{\langle y, \underline{x}\theta \rangle} + \underbrace{y^T y}_{J(\theta)}) \frac{1}{2}
 \end{aligned}$$

$\Rightarrow J(\theta)$ quadratic func of θ ;



Review (II): gradient of linear form

$$\frac{\partial(\theta^T X^T y)}{\partial \theta} = X^T y$$

One
Concrete
Example

$$f(w) = w^T a = [w_1, w_2, w_3] \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = w_1 + 2w_2 + 3w_3$$

$$\frac{\partial f}{\partial w_1} = 1$$

$$\frac{\partial f}{\partial w_2} = 2$$

$$\frac{\partial f}{\partial w_3} = 3$$



$$\frac{\partial f}{\partial w} = \frac{\partial w^T a}{\partial w} = a = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

Review (III): Gradient of Quadratic Form

- See L2-note.pdf -> Page 17, Page 23-24
- See white board

$$\frac{\partial(\theta^T X^T X \theta)}{\partial \theta} = \frac{\partial(\theta^T G \theta)}{\partial \theta} = 2G\theta = 2X^T X \theta$$

Review (III): Single Var-Func to Multivariate

Single Var- Function	Multivariate Calculus
Derivative	Partial Derivative
Second-order derivative	Gradient
	Directional Partial Derivative
	Vector Field
	Contour map of a function
	Surface map of a function
	Hessian matrix
	Jacobian matrix (vector in / vector out)

Review (IV) : Definitions of gradient (Matrix_calculus / Scalar-by-vector)

- Size of gradient is always the same as the size of variable

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix} \in \mathbb{R}^n \quad \text{if } x \in \mathbb{R}^n$$

In principle, gradients are a natural extension of partial derivatives to functions of multiple variables.

Review (V): Rank of a Matrix

- $\text{rank}(A)$ (the rank of a m -by- n matrix A) is
 - = The maximal number of linearly independent columns
 - = The maximal number of linearly independent rows

- If A is n by m , then
 - $\text{rank}(A) \leq \min(m, n)$
 - If $n = \text{rank}(A)$, then A has full row rank
 - If $m = \text{rank}(A)$, then A has full column rank

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 2 & 1 \\ 4 & 2 \end{pmatrix}$$

Rank=? Rank=?

If A is $n \times n$, $\text{rank}(A) = n$ iff A is invertible

$\text{rank}(AB) \leq \min(\text{rank}(A), \text{rank}(B))$

Extra: Loss $J()$ is Convex

$$\Rightarrow J(\theta) = \frac{1}{2} (\theta^T X^T X \theta - 2\theta^T X^T y + y^T y)$$

$$\Rightarrow \text{Hessian } (J(\theta)) = X^T X$$

Gram matrix
PSD

$J(\theta)$ is convex

If $\nabla J(\theta^*) = 0$, $J(\theta)$ is minimized @ θ^*

Review: Hessian Matrix

Derivatives and Second Derivatives

Cost function

$$J(\boldsymbol{\theta})$$

Gradient

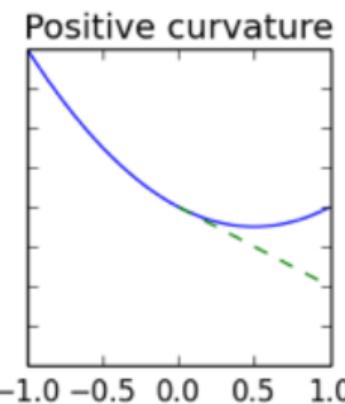
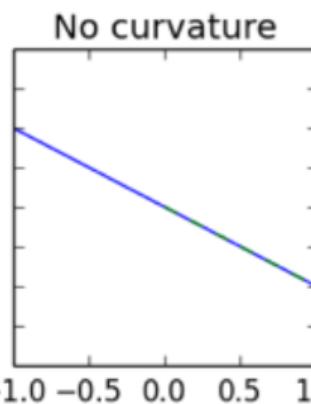
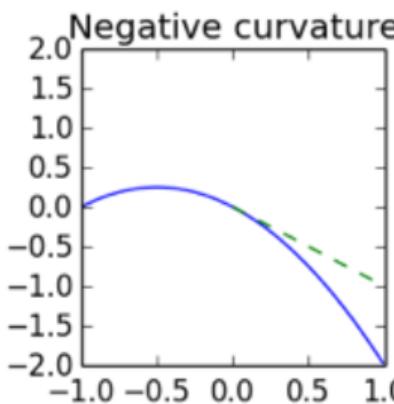
$$\mathbf{g} = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$g_i = \frac{\partial}{\partial \theta_i} J(\boldsymbol{\theta})$$

Hessian

$$\mathbf{H}$$

$$H_{i,j} = \frac{\partial}{\partial \theta_j} g_i$$



$\mathbf{H} \text{ PD}$
for positive
curvature

Positive Definite Hessian

Extra: Convex function

- Intuitively, a convex function (1D case) has a single point at which the derivative goes to zero, and this point is a minimum.
- Intuitively, a function f (1D case) is convex on the range $[a,b]$ if a function's second derivative is positive every-where in that range.
- Intuitively, if a multivariate function's Hessians is pd (positive definite!), this (multivariate) function is Convex
 - Intuitively, we can think “Positive definite” matrices as analogy to positive numbers in matrix case

Our loss function $J()$'s Hessian is
Positive Semi-definite - PSD

$H = \mathbf{X}^T \mathbf{X}$
is always
PSD

Intuitively, a convex function with PD hessian is minimized @ point whose

- ✓ derivative (slope) is zero
- ✓ gradient is zero vector (multivariate case)

Gram Matrix $H = X^T X$

when X is full rank, H is positive definite!

Extra: Gram Matrix $H = X^T X$
is always positive semi-definite!

Because for any vector a

$$a^T X^T X a = \|Xa\|_2^2 \geq 0$$



Besides, when X is full rank,
 H is Positive Definite (PD) and invertible

Comments on the normal equation

When \mathbf{X} full rank $\theta^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \vec{\mathbf{y}}$

- In most situations of practical interest, the number of data points n is larger than the dimensionality p of the input space and the matrix \mathbf{X} is of full column rank. If this condition holds, then it is easy to verify that $\mathbf{X}^T \mathbf{X}$ is necessarily invertible.

$$n \gg p$$

- The assumption that $\mathbf{X}^T \mathbf{X}$ is invertible implies that it is positive definite, thus the critical point (by solving gradient to zero) we have found is a minimum.
- What if \mathbf{X} has less than full column rank? \rightarrow regularization (later).

when \mathbf{X} not full rank, e.g.

e.g. when $p > n \Rightarrow \text{rank}(\mathbf{X}) < p \Rightarrow \text{rank}(\underbrace{\mathbf{X}^T \mathbf{X}}_{p \times p}) \leq \min\{\text{rank}(\mathbf{X})\} < p \Rightarrow \text{not invertible}$

Extra: positive semi-definite!

$A \in \mathbb{R}^{n \times n}, \forall x \in \mathbb{R}^n$

L2-Note: Page 17

If $x^T A x \geq 0$

($x \in \mathbb{R}^n$)

$\Rightarrow A$ is positive semi-definite (PSD)

If $x^T A x > 0$

$\Rightarrow A$ is PD \Rightarrow full rank / invertible

See proof on L2-Note: Page 18

Extra: Scalability to big data?

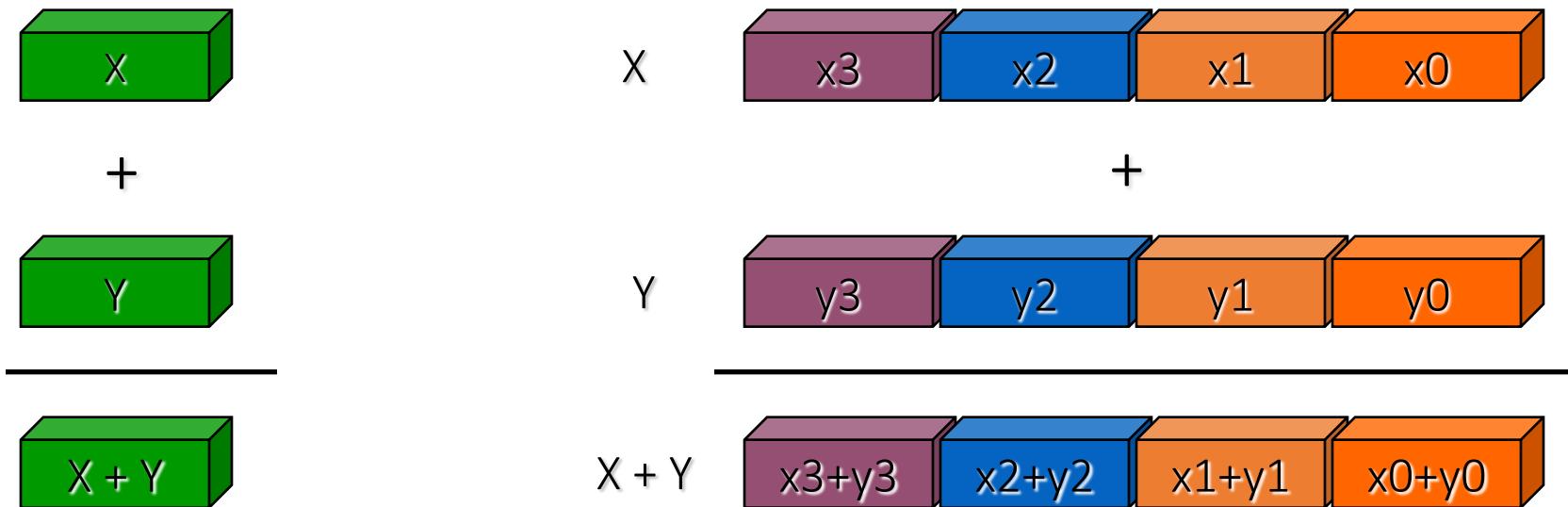
- Traditional CS view: Polynomial time algorithm, Wow!
- Large-scale learning: Sometimes even $O(n)$ is bad! => Many state-of-the-art solutions (e.g., low rank, sparse, hardware, sampling, randomized...)

Simple example: Matrix multiplication

$$n \begin{matrix} n \\ \times \\ n \end{matrix} = \begin{matrix} n^3 \end{matrix}$$

SIMD: Single Instruction, Multiple Data

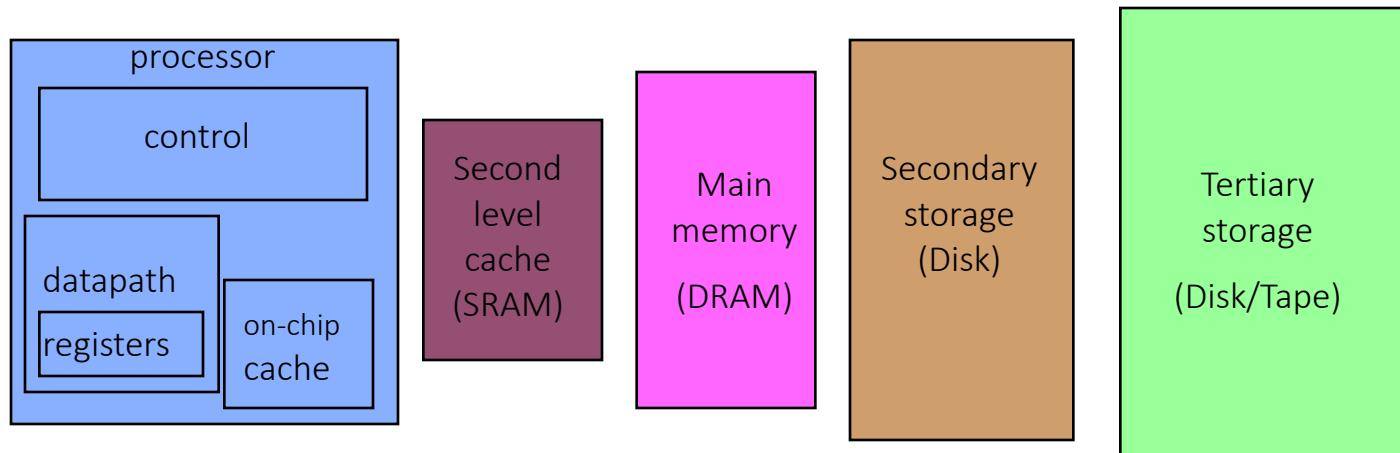
- Scalar processing
 - traditional mode
 - one operation produces one result
- SIMD processing
 - with SSE / SSE2
 - SSE = streaming SIMD extensions
 - one operation produces multiple results



Slide Source: Alex Klimovitski & Dean Macri, Intel Corporation

Memory Hierarchy

- Most programs have a high degree of **locality** in their accesses
 - **spatial locality**: accessing things nearby previous accesses
 - **temporal locality**: reusing an item that was previously accessed
- Memory hierarchy tries to exploit locality to improve average



Speed	1ns	10ns	100ns	10ms	10sec
Size	KB	MB	GB	TB	PB

Note on Matrix Storage

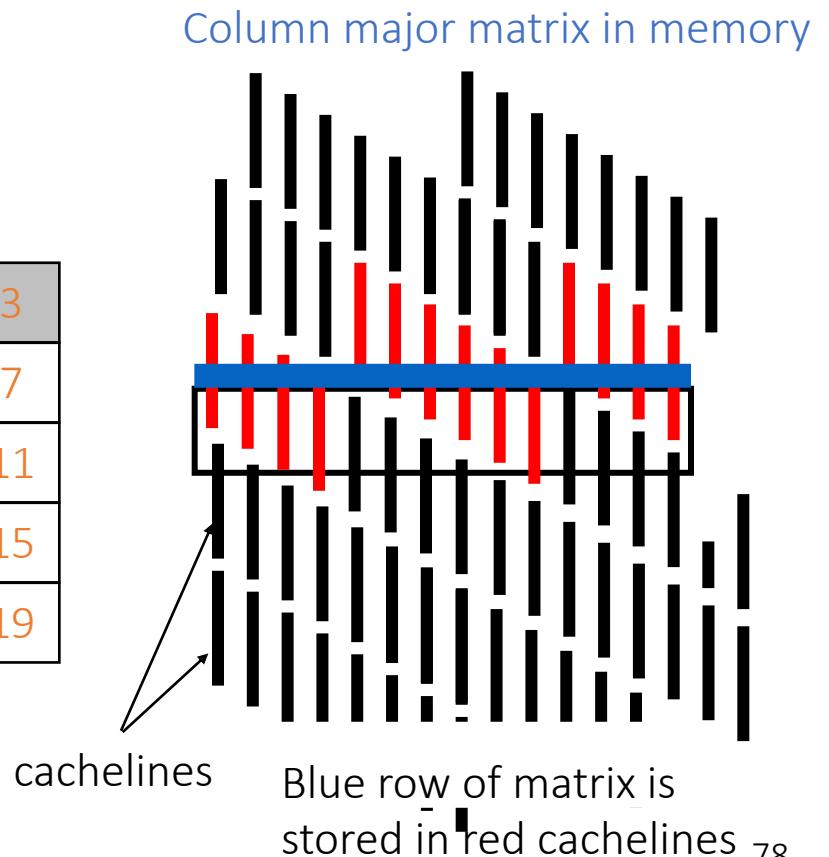
- A matrix is a 2-D array of elements, but memory addresses are “1-D”
- Conventions for matrix layout
 - by column, or “column major” (Fortran default); $A(i,j)$ at $A+i+j*n$
 - by row, or “row major” (C default) $A(i,j)$ at $A+i*n+j$
 - recursive (later)

Column major

0	5	10	15
1	6	11	16
2	7	12	17
3	8	13	18
4	9	14	19

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19

- Column major (for now)



Strassen's Matrix Multiply

- The traditional algorithm (with or without tiling) has $O(n^3)$ flops
- Strassen discovered an algorithm with asymptotically lower flops
 - $O(n^{2.81})$
- Consider a 2×2 matrix multiply, normally takes 8 multiplies, 4 adds
 - Strassen does it with 7 multiplies and 18 adds

Let $M = \begin{matrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{matrix} = \begin{matrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{matrix} \begin{matrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{matrix}$

$$\begin{aligned} p_1 &= (a_{12} - a_{22}) * (b_{21} + b_{22}) & p_5 &= a_{11} * (b_{12} - b_{22}) \\ p_2 &= (a_{11} + a_{22}) * (b_{11} + b_{22}) & p_6 &= a_{22} * (b_{21} - b_{11}) \\ p_3 &= (a_{11} - a_{21}) * (b_{11} + b_{12}) & p_7 &= (a_{21} + a_{22}) * b_{11} \\ p_4 &= (a_{11} + a_{12}) * b_{22} \end{aligned}$$

$$\text{Then } m_{11} = p_1 + p_2 - p_4 + p_6$$

$$m_{12} = p_4 + p_5$$

$$m_{21} = p_6 + p_7$$

$$m_{22} = p_2 - p_3 + p_5 - p_7$$

Extends to $n \times n$ by divide&conquer

Strassen (continued)

$$\begin{aligned} T(n) &= \text{Cost of multiplying } nxn \text{ matrices} \\ &= 7*T(n/2) + 18*(n/2)^2 \\ &= O(n \log_2 7) \\ &= O(n^{2.81}) \end{aligned}$$

- Asymptotically faster
 - Several times faster for large n in practice
 - Cross-over depends on machine
 - “Tuning Strassen's Matrix Multiplication for Memory Efficiency”, M. S. Thottethodi, S. Chatterjee, and A. Lebeck, in Proceedings of Supercomputing '98
- Possible to extend communication lower bound to Strassen
 - #words moved between fast and slow memory $= \Omega(n^{\log 2})$
 $\frac{7}{M} / M^{(\log 2)/2 - 1} \sim \Omega(n^{2.81} / M^{0.4})$
 - (Ballard, D., Holtz, Schwartz, 2011)
 - Attainable too