UVA CS 6316: Machine Learning

Lecture 3: Linear Regression Basics

Dr. Yanjun Qi

University of Virginia

Department of Computer Science

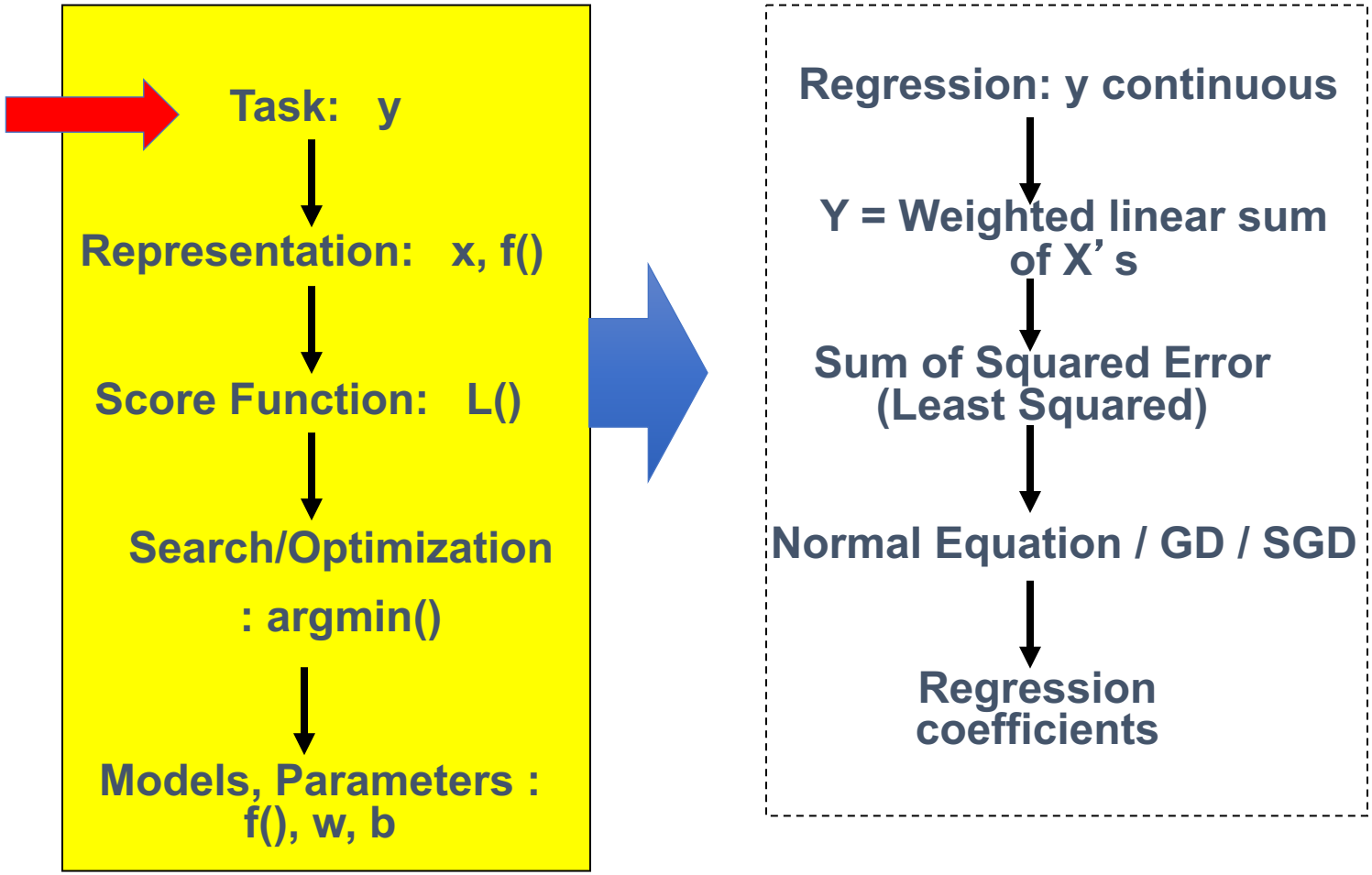
Course Content Plan ➔
Six major sections of this course

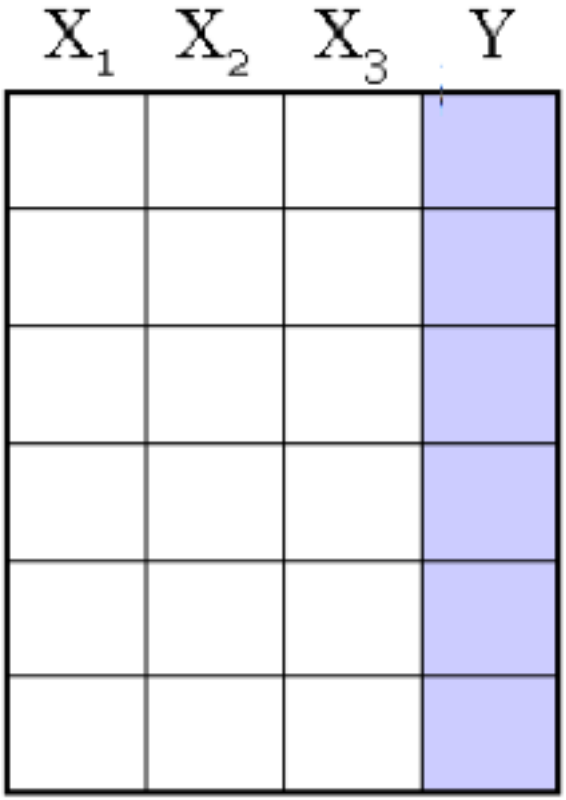
- ❑ Regression (supervised) — Y is a continuous
- ❑ Classification (supervised) — Y is a discrete
- ❑ Unsupervised models — NO Y
- ❑ Learning theory — About f()

- ❑ Graphical models — About interactions among X1,… Xp

- ❑ Reinforcement Learning — Learn program to Interacts with its environment

Today: Multivariate Linear Regression in a Nutshell





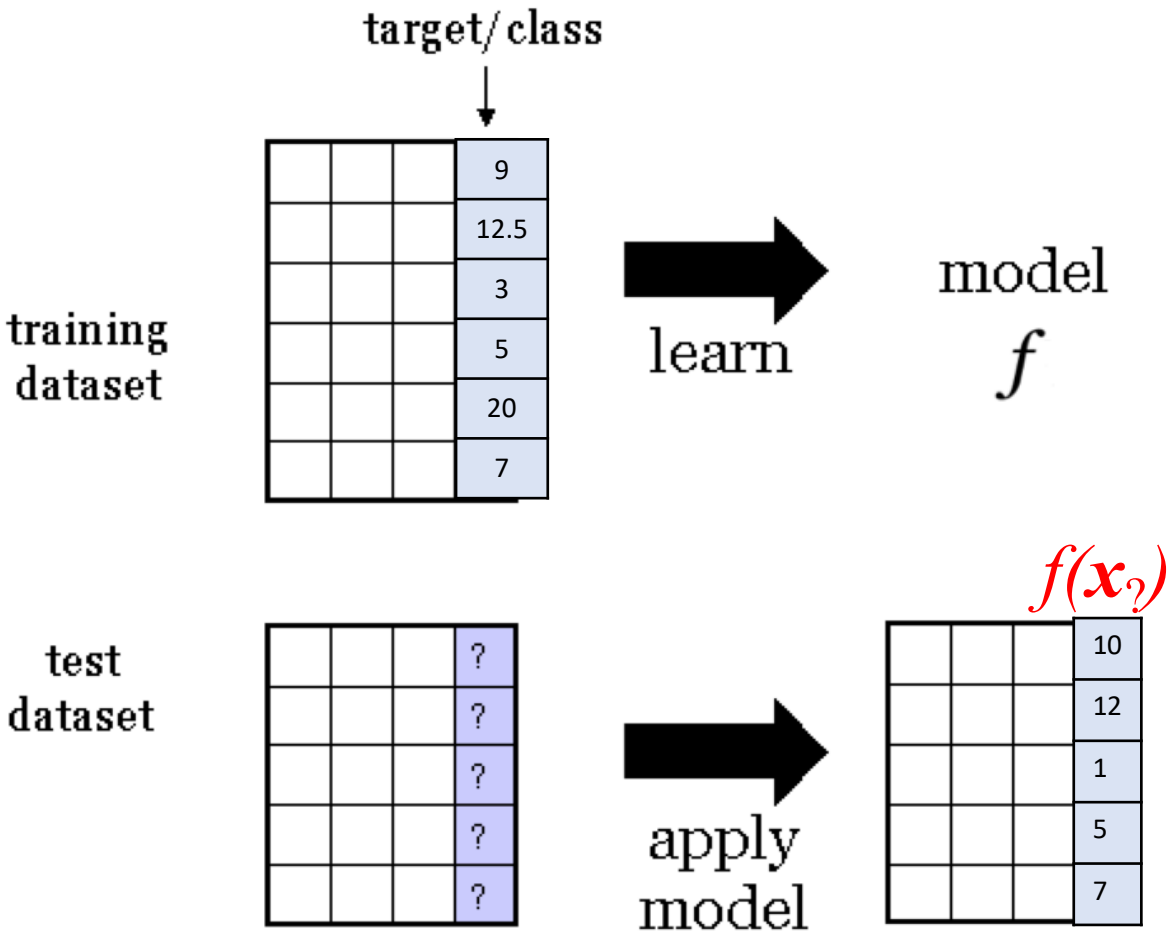
A Dataset for regression

$$f : X \longrightarrow Y$$

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

- e.g. target Y can be a continuous target variable

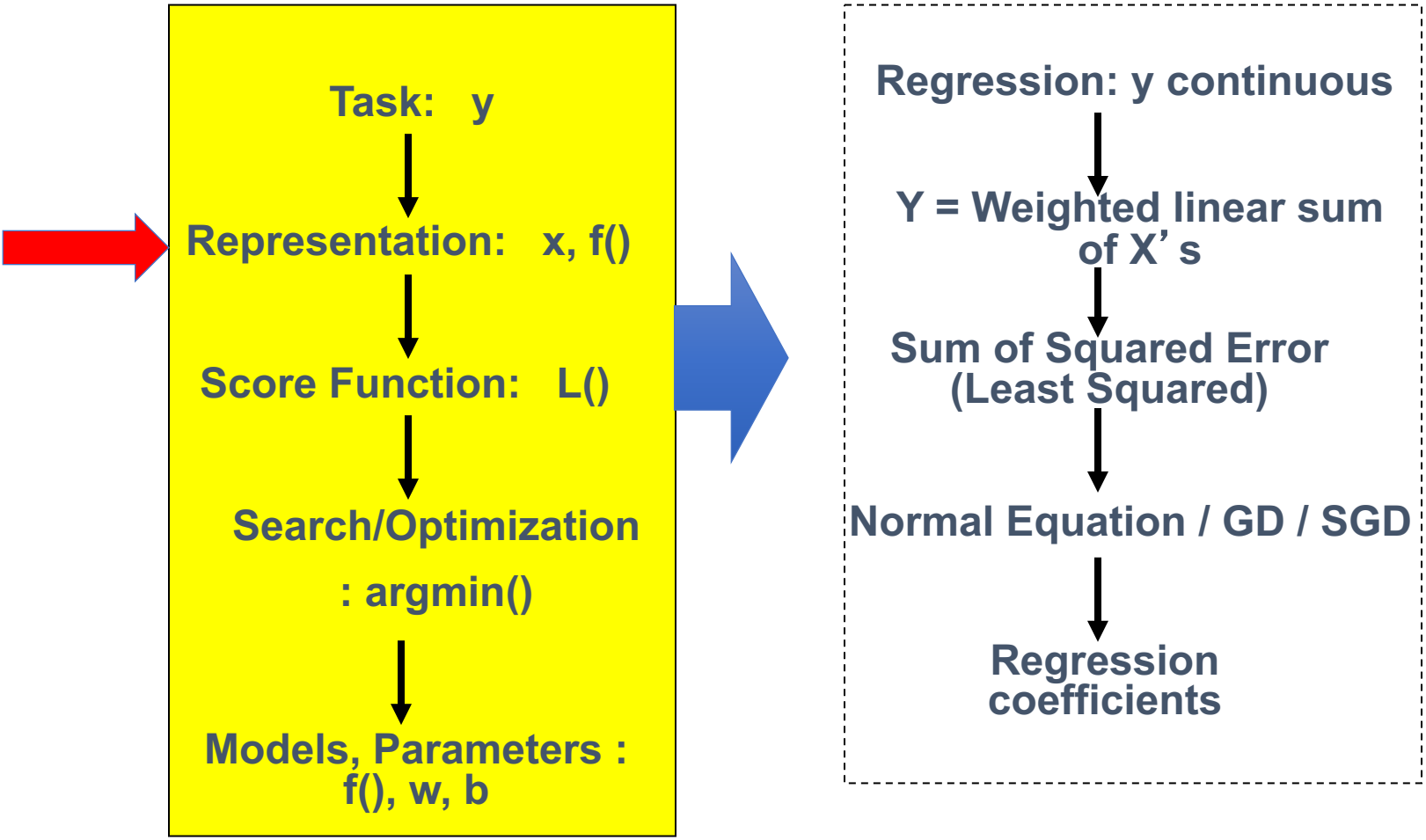
$$\mathbf{X}_{train} = \begin{bmatrix} - & - & \mathbf{x}_1^T & - & - \\ - & - & \mathbf{x}_2^T & - & - \\ & \vdots & \vdots & \vdots & \\ - & - & \mathbf{x}_n^T & - & - \end{bmatrix} \qquad \vec{y}_{train} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

training dataset

test dataset

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

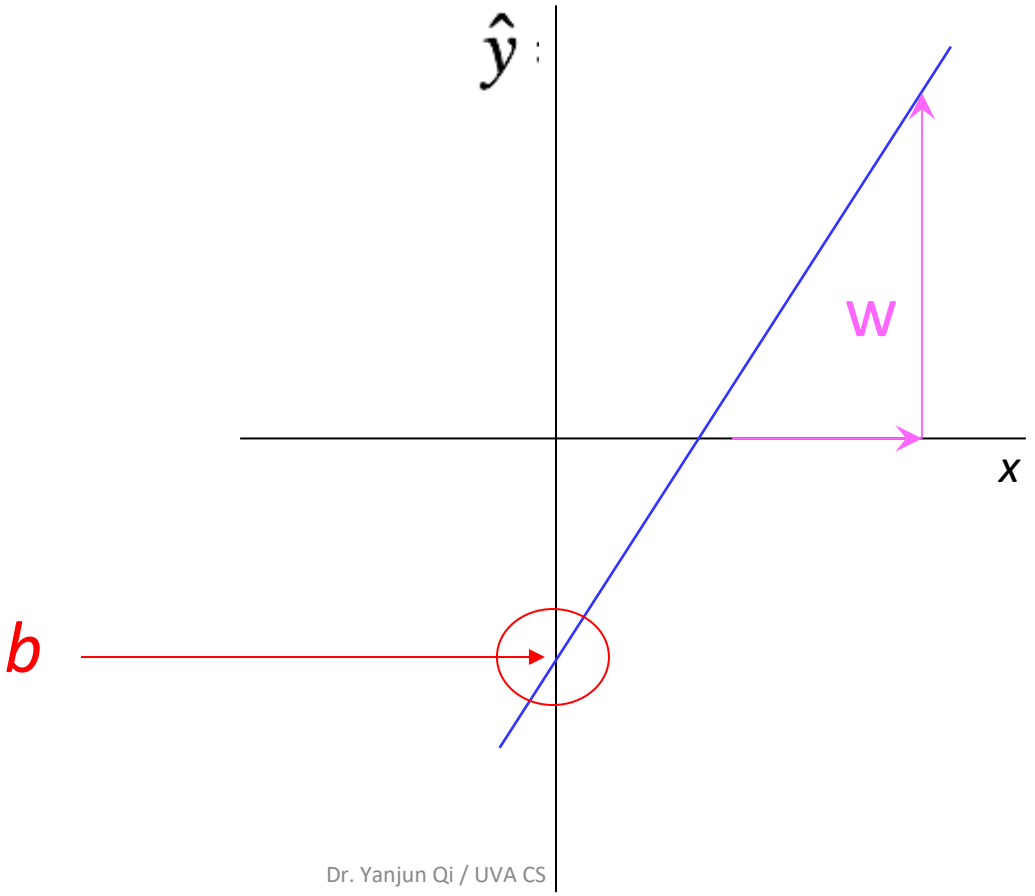
Today: Multivariate Linear Regression in a Nutshell



Review: f(x) is Linear when X is 1D

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

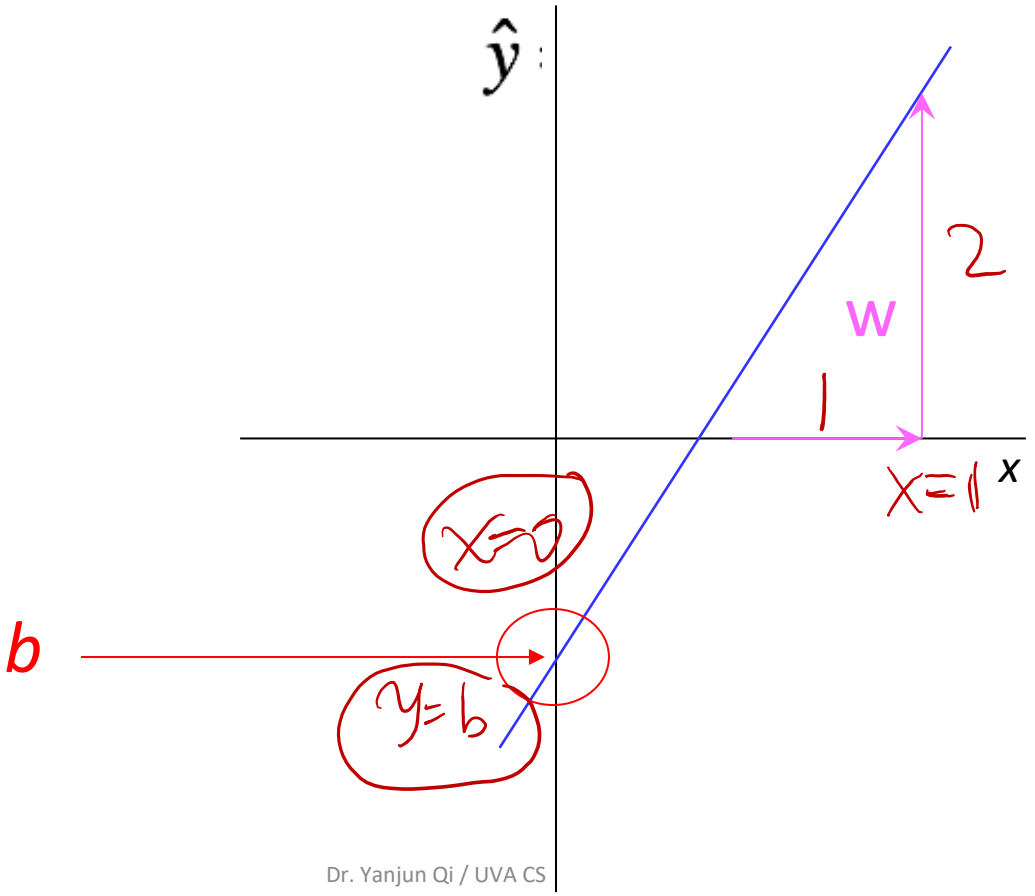
A slope of 2 (i.e. w=2) means that every 1-unit change in X yields a 2-unit change in Y.

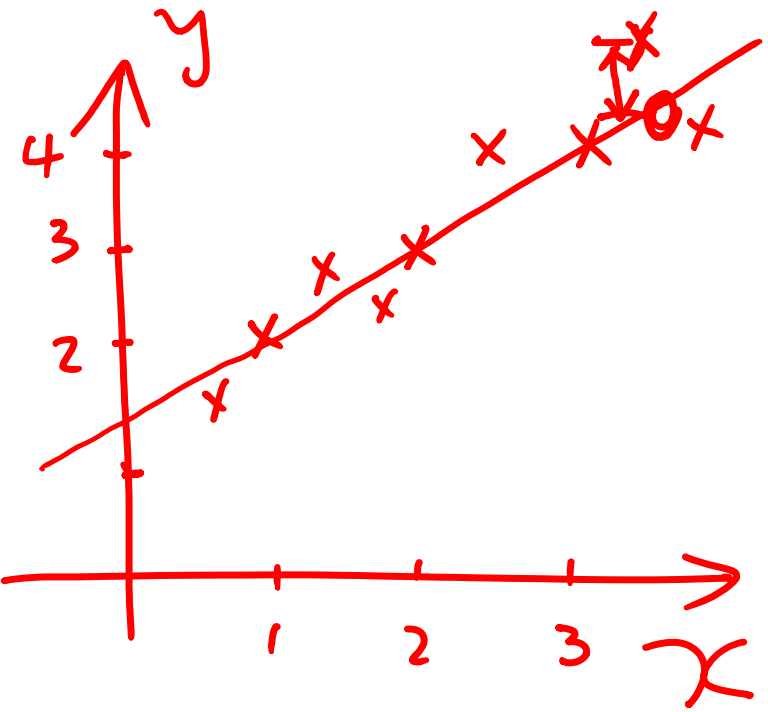


Review: f(x) is Linear when X is 1D

- *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.



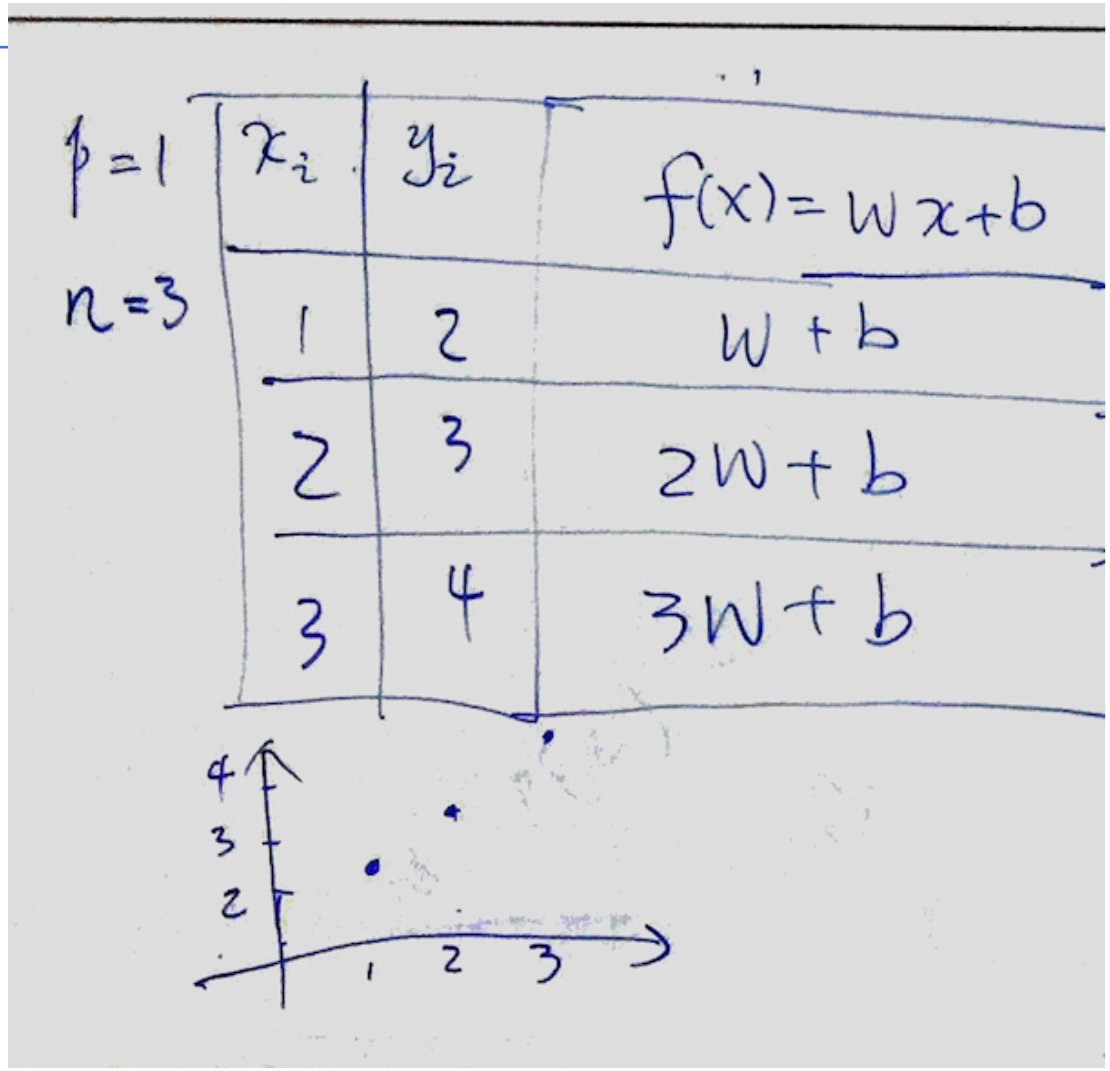


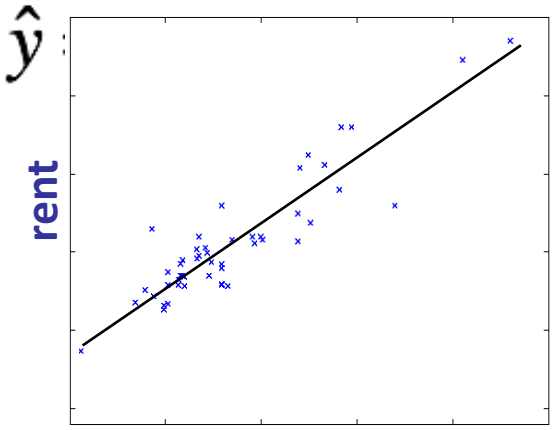
x	y	f(x)=wx+b
1	2	f(1) = w+b
2	3	f(2)=2w+b
3	4	f(3)=3w+b

$$loss(w,b) = (w+b-2)^2 + (2w+b-3)^2 + (3w+b-4)^2$$

$$(w,b) \Rightarrow = \underset{w,b}{\operatorname{argmin}}\ loss()$$

One concrete example

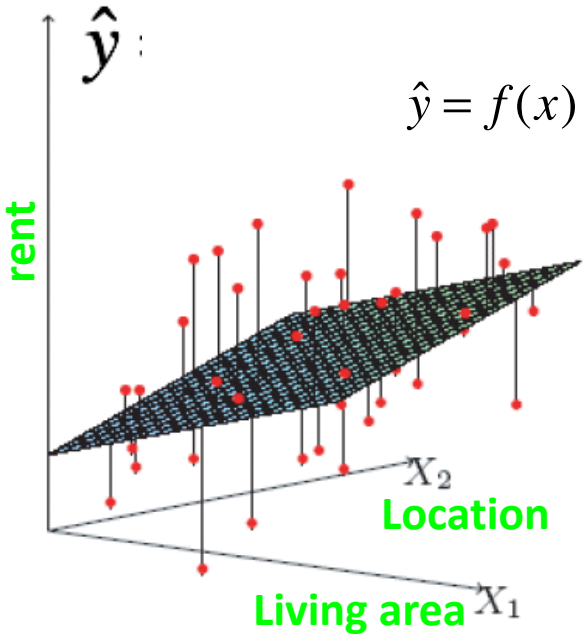




Living area as X

f(x)=wx+b

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



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

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

(Living area, Location) as X

Linear SUPERVISED Regression

$$f: X \longrightarrow Y$$

e.g. Linear Regression Models

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

=> Features **x**:

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

=> Target y:

e.g., Rent ➔ Continuous

Apply f(x) : A Concise Notation

- Represent each sample **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:

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

  - The parameter vector θ is also a column vector

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

$$\hat{y}=f(\mathbf{x})$$
$$=\mathbf{x}^T\theta=\theta^T\mathbf{x}$$
$$=\theta_0+\theta_1x_1+\theta_2x+\cdots\theta_px_p$$

Review:

- Dot (or Inner) Product of two Vectors <x, y>

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

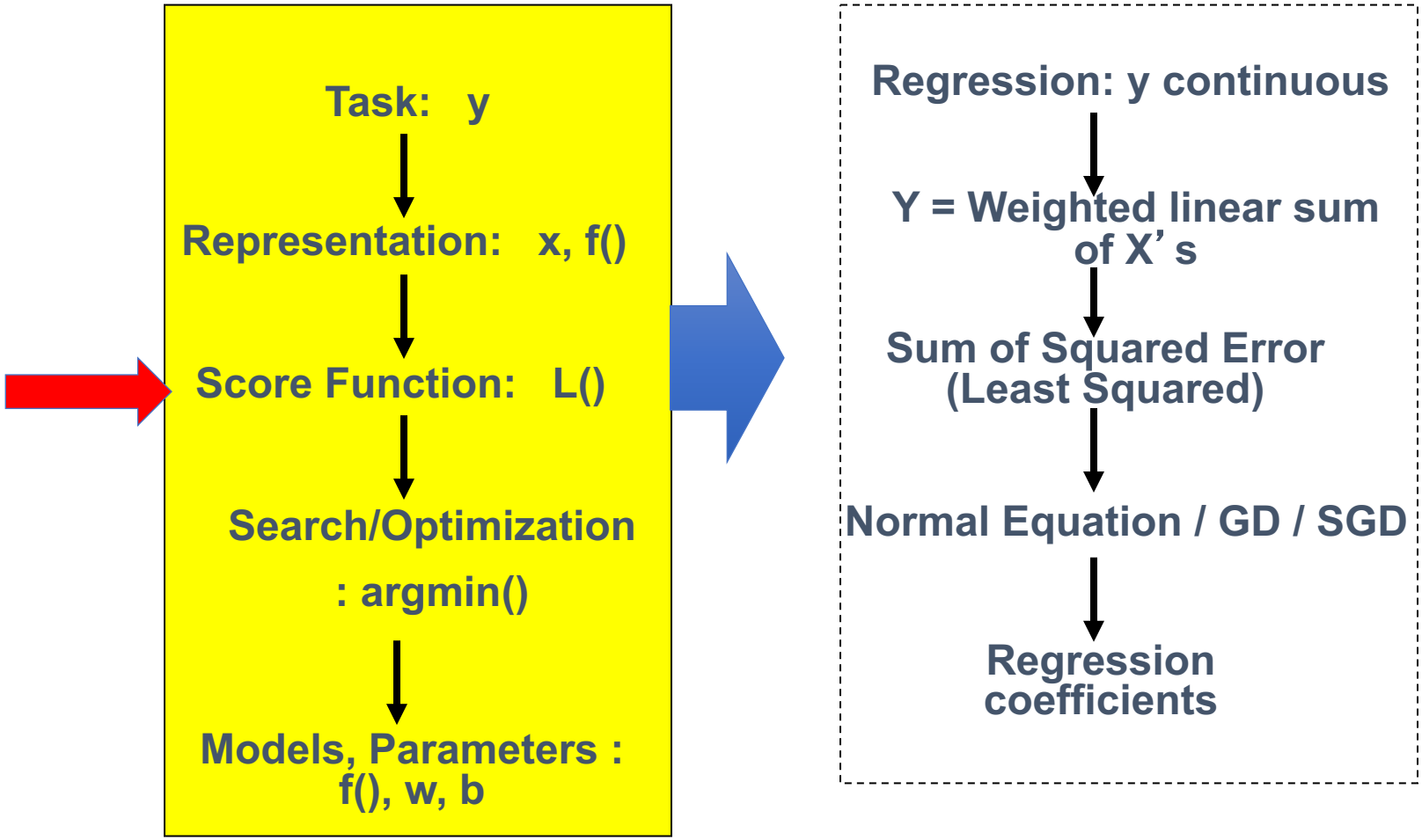
$$\langle x, y \rangle = \langle y, x \rangle$$

$$\underbrace{p \times 1 \quad p \times 1}_{1 \times 1} = x^T y = y^T x$$

We also always write

$\langle x, y \rangle = x^T y \in \mathbb{R} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ x_2 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^{n} x_i y_i.$

Today: Multivariate Linear Regression in a Nutshell

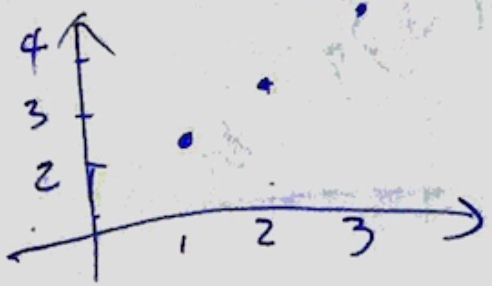


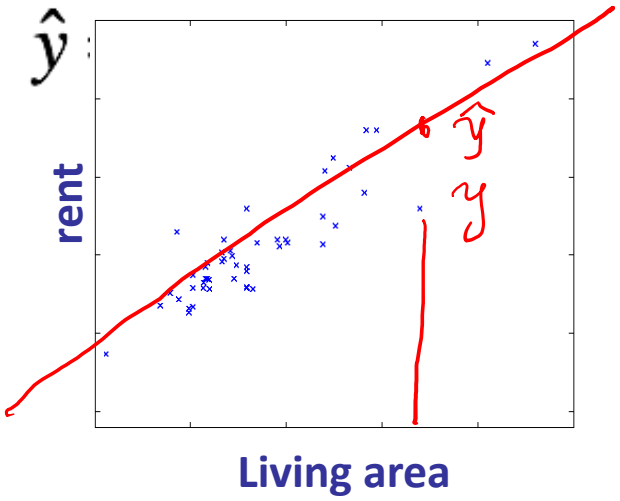
One concrete example

$p = 1$

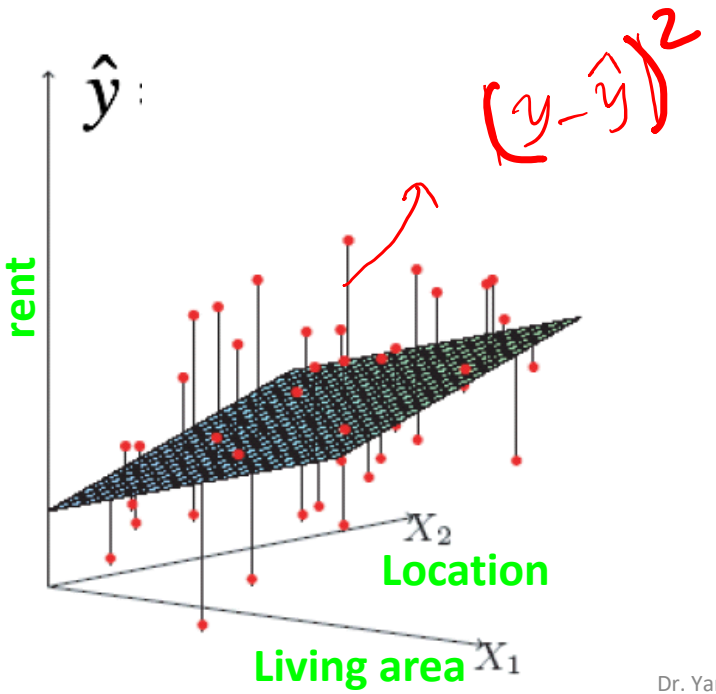
$n = 3$

x_i	y_i	$f(x) = wx + b$	$(f(x) - y)^2$
1	2	$w + b$	$(w + b - 2)^2$
2	3	$2w + b$	$(2w + b - 3)^2$
3	4	$3w + b$	$(3w + b - 4)^2$





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



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

Now the loss function:

SSE: Sum of squared error

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

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

$x_i^T\theta$

Now the loss function: via A Concise Notation

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

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

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

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

$$= \frac{1}{2}\left(\theta^T X^T X\theta - \theta^T X^T \vec{y} - \vec{y}^T X\theta + \vec{y}^T\vec{y}\right)$$

$$a = (X\theta - y) = \begin{bmatrix} x_1^T\theta - y_1 \\ x_2^T\theta - y_2 \\ \vdots \\ x_n^T\theta - y_n \end{bmatrix}$$

$n \times p \quad p \times 1 \quad n \times 1$

$n \times 1$

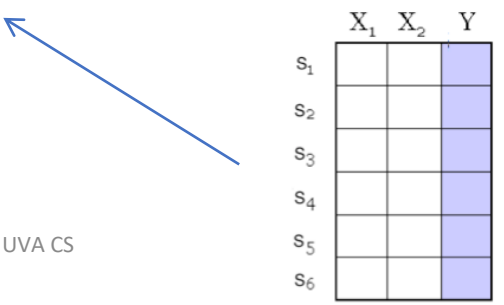
$$a^T a = \sum_{i=1}^{n} a_i^2$$

Review: Training Set in Matrix Form

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

$$X\theta = \begin{bmatrix} x_1^T\theta \\ \vdots \\ x_n^T\theta \end{bmatrix}$$

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



$$\vec{x}_1 = \begin{bmatrix} 1 \\ x_{1,1} \\ x_{1,2} \\ x_{1,3} \\ \vdots \\ x_{1,p} \end{bmatrix}$$

$$\vec{x}_1^T = \begin{bmatrix} 1, x_{1,1}, x_{1,2}, \cdots, x_{1,p} \end{bmatrix}$$

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

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

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

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

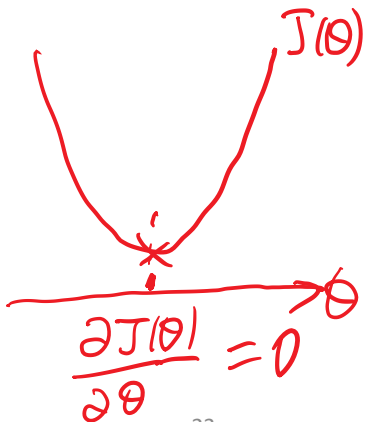
Since $\theta^T X^T y = y^T X\theta$

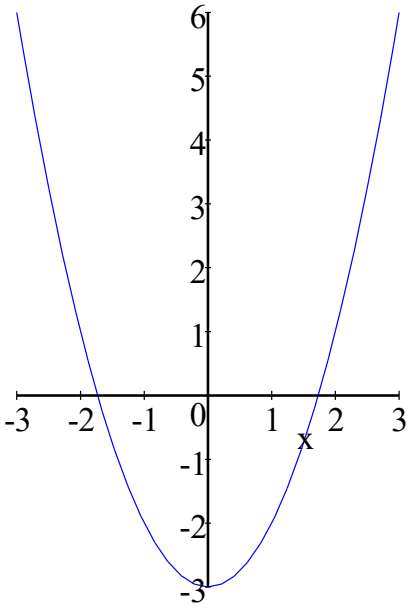
$\langle X\theta, y \rangle \quad \langle y, X\theta \rangle$

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

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

1d case





Review (IV): Derivative of a Quadratic Function

$$y = x^2 - 3$$

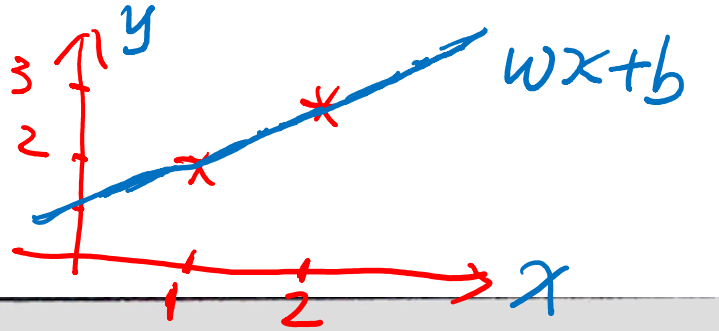
$$y' = 2x$$

$$y'' = 2$$

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

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

One concrete example



[a]

$$J(w,b) = (w+b-2)^2 + (2w+b-3)^2$$

[b]

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

[c]

$$\frac{\partial J(w,b)}{\partial w}\bigg|_{\theta_1} = 2(w+b-2) + 4(2w+b-3) = 0$$

$$\frac{\partial J(w,b)}{\partial b}\bigg|_{\theta_0} = 2(w+b-2) + 2(2w+b-3) = 0$$

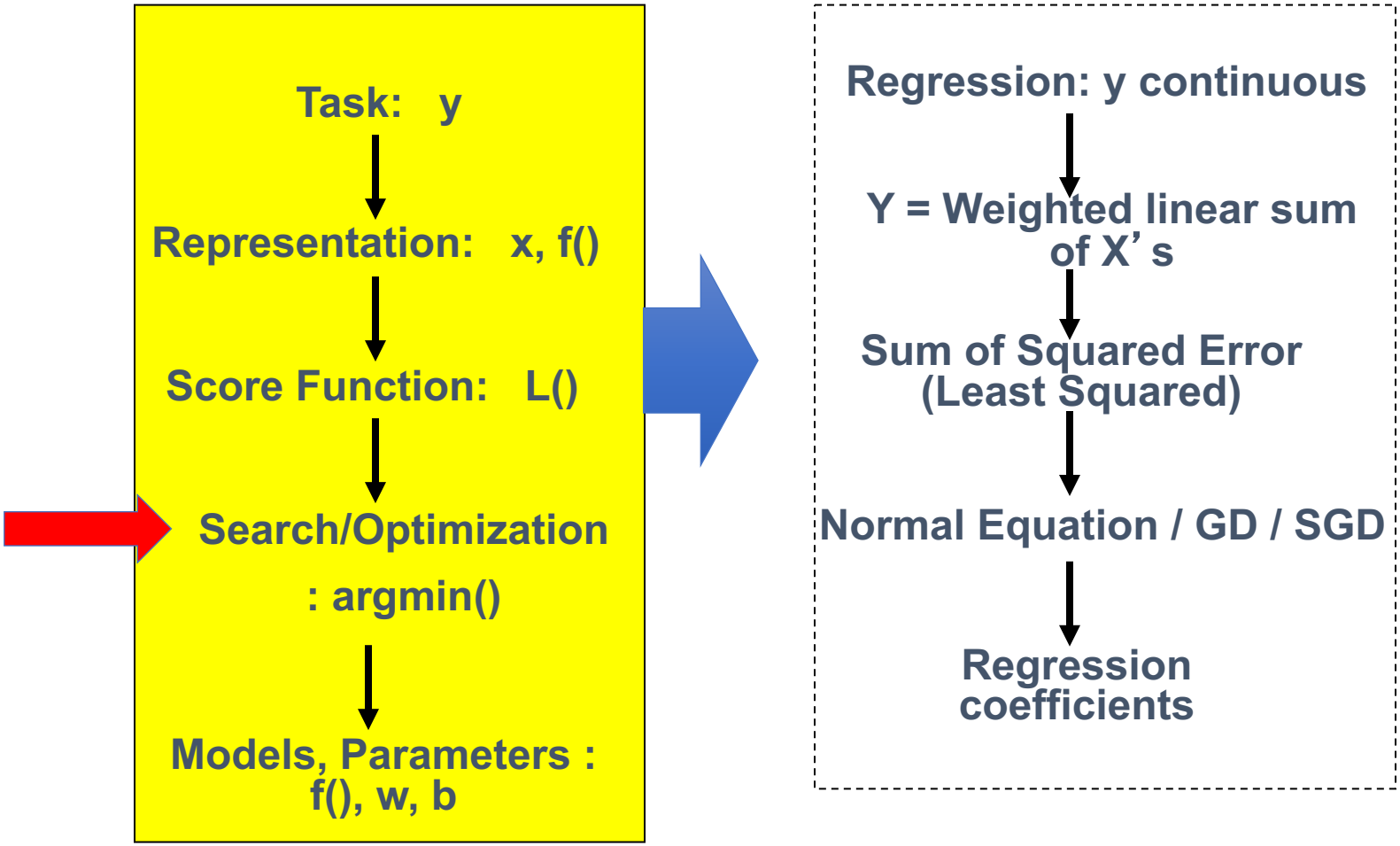
$$\Rightarrow \begin{cases} 10w + 6b - 16 = 0 \\ 6w + 4b - 10 = 0 \end{cases}$$

[D]

$$\Rightarrow 2b - 2 = 0 \Rightarrow b = 1 \Rightarrow w = 1$$

In Step [D], we solve the matrix equation via Gaussian Elimination

Today: Multivariate Linear Regression in a Nutshell



Method I: normal equations to minimize the loss

- Write the cost function in matrix form:

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

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

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

$$\nabla_\theta J(\theta) = \frac{\partial J(\theta)}{\partial \theta} = \frac{1}{2}(2X^TX\theta - X^T\vec{y} - X^T\vec{y})$$

$\theta^T X^T \vec{y} = \vec{y}^T X\theta$

$2X^TX\theta$

$1\times n \quad n\times 1$

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

$$\nabla_\theta J(\theta) = 0 \Rightarrow$$

$$X^TX\theta = X^T\vec{y}$$

The normal equations

WHY ??

$$\Downarrow$$

Closed form solution

$$\theta^* = (X^TX)^{-1}X^T\vec{y}$$

See handout 4.1 + 4.3 ⇒ matrix calculus, partial deri ⇒ Gradient

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

$$\nabla_\theta (-2 \theta^T X^T y) = -2 X^T y \quad (P24)$$

$$\nabla_\theta (y^T y) = 0$$

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

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

Gram matrix $X^T X$

Extra: Loss J() is Convex

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

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

$X^T X$ → Gram matrix

Gram matrix ⇓ PSD

⇓

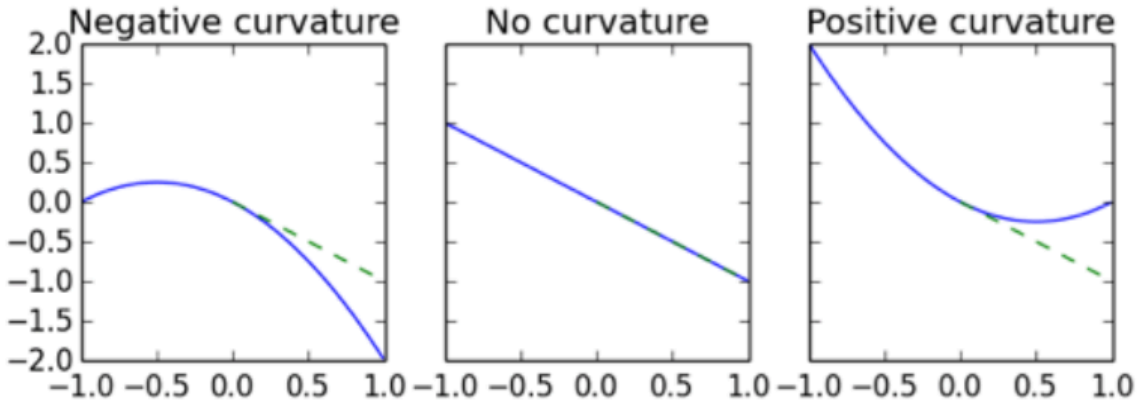
$J(\theta)$ is convex

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

Review: Hessian Matrix

Derivatives and Second Derivatives

Cost function	Gradient	Hessian
$J(\boldsymbol{\theta})$	$\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$	$\boldsymbol{H}$
	$g_i = \frac{\partial}{\partial \theta_i} J(\boldsymbol{\theta})$	$H_{i,j} = \frac{\partial}{\partial \theta_j} g_i$



H 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

$H = X^T X$ 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

Later: Comments on the normal equation

when X full rank

$$\theta^* = \left(X^T X\right)^{-1} X^T \vec{y}$$

- 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 $X^T X$ is necessarily invertible.

$n >> p$

- The assumption that $X^T X$ 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? → regularization (later).

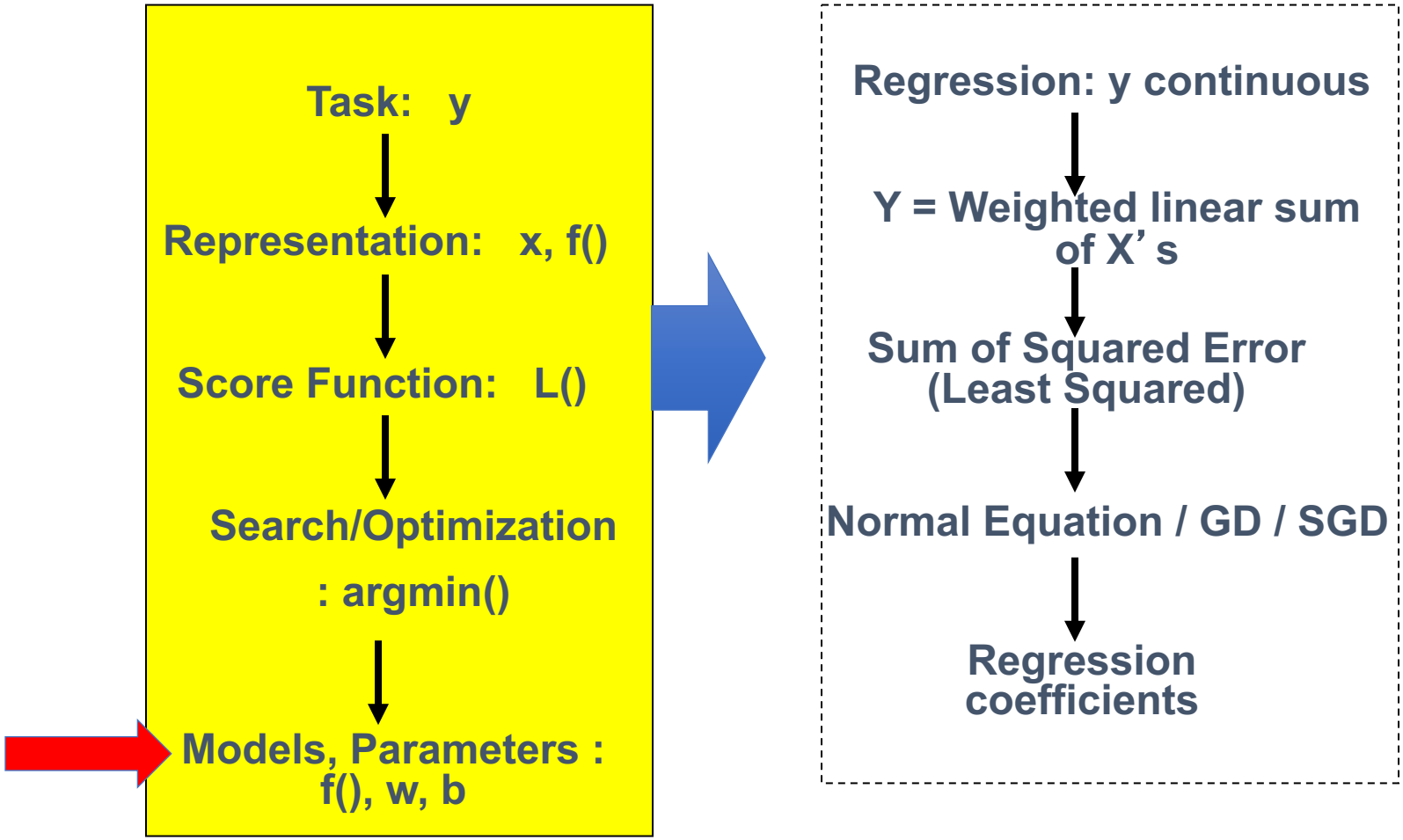
when X not full rank, e.g.

e.g. when $p > n \Rightarrow \text{rank}(X) < p \Rightarrow \text{rank}(X^T X) \leq \min\{\text{rank}(X)\} < p$

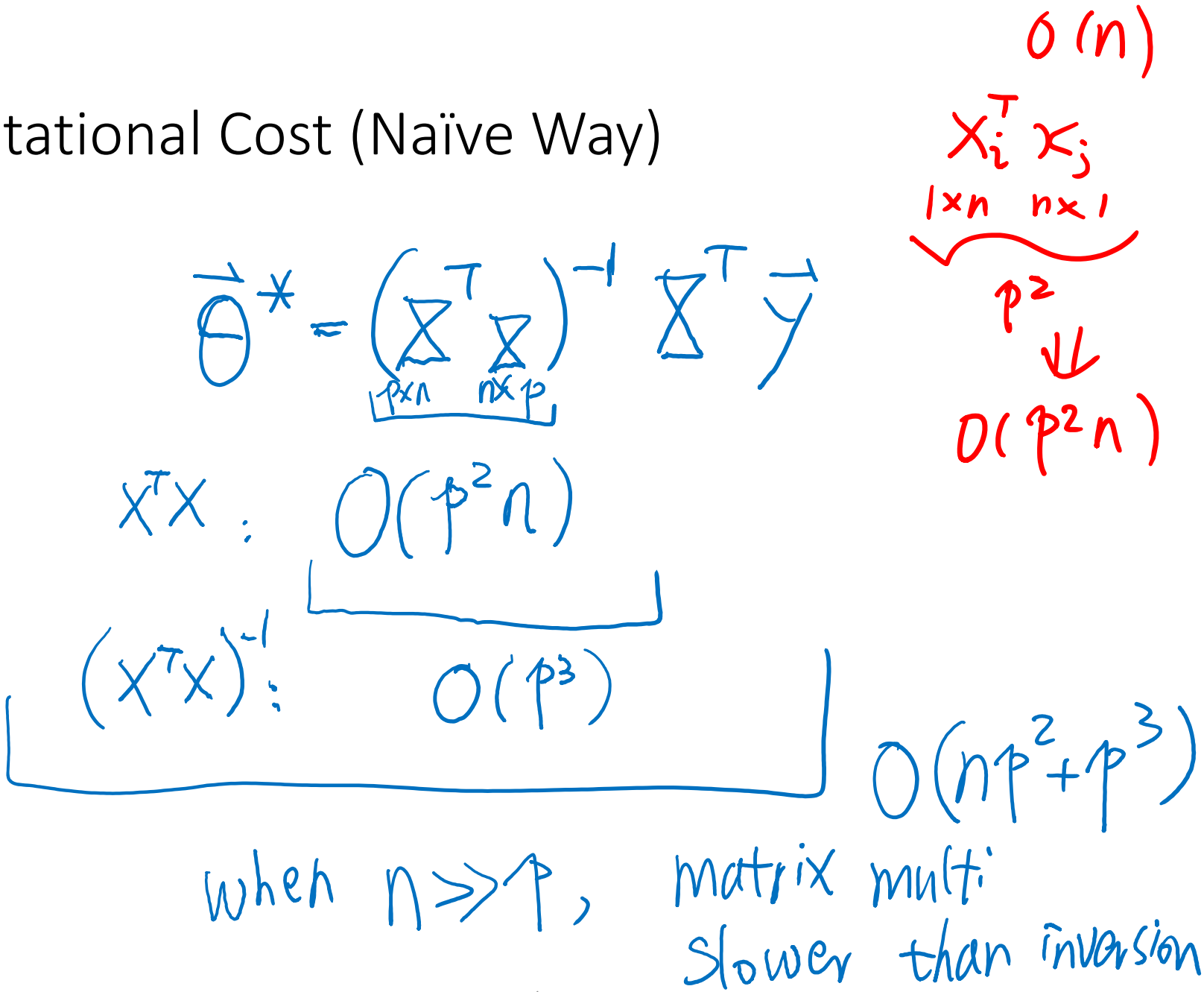
X: $n \times p$; $X^T X$: $p \times p$

$\Rightarrow$ not invertible

Today: Multivariate Linear Regression in a Nutshell



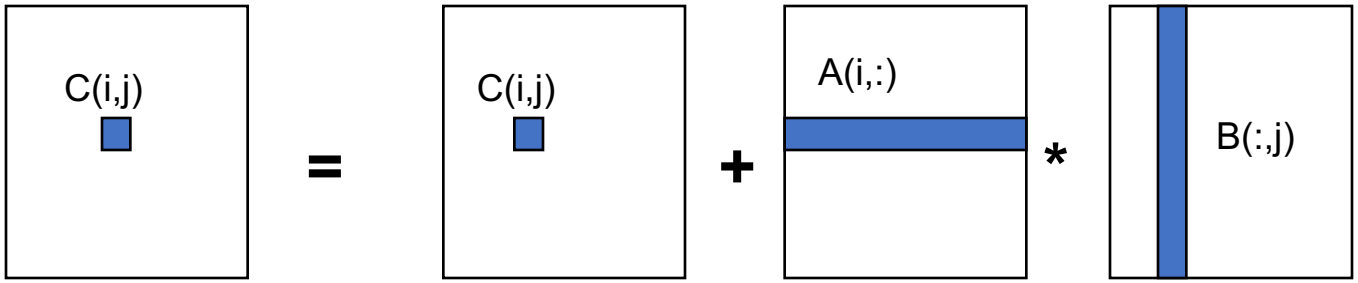
Computational Cost (Naïve Way)



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



The following complexity figures assume that arithmetic with individual elements has complexity $O(1)$, as is the case with fixed-precis 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: Scalability to big data?

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

Basic Linear Algebra Subroutines (BLAS) ➔ numpy: a wrapper library of BLAS

- **Industry standard interface (evolving)**
  - **www.netlib.org/blas, www.netlib.org/blas/blast--forum**
- **Vendors, others supply optimized implementations**
- **History**
  - **BLAS1 (1970s):**
    - **vector operations: dot product, saxpy (y=α*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~2, less overhead**
    - **somewhat faster than BLAS1**
  - **BLAS3 (late 1980s)**
    - **matrix-matrix operations: matrix matrix multiply, etc**
    - **m <= 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}`

source: Stanford Optim EE course

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 produ norms, a generalized vector addition of the form

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

(called "axpy") and several other operations.

Level 2 [edit]

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

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

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

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

with $\boldsymbol{T}$ being triangular. Design of the Level 2 BLAS started in 1984, with re in 1988.[19] The Level 2 subroutines are especially intended to improve per programs using BLAS on vector processors, where Level 1 BLAS are subo "because they hide the matrix-vector nature of the operations from the com

Level 3 [edit]

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

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

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

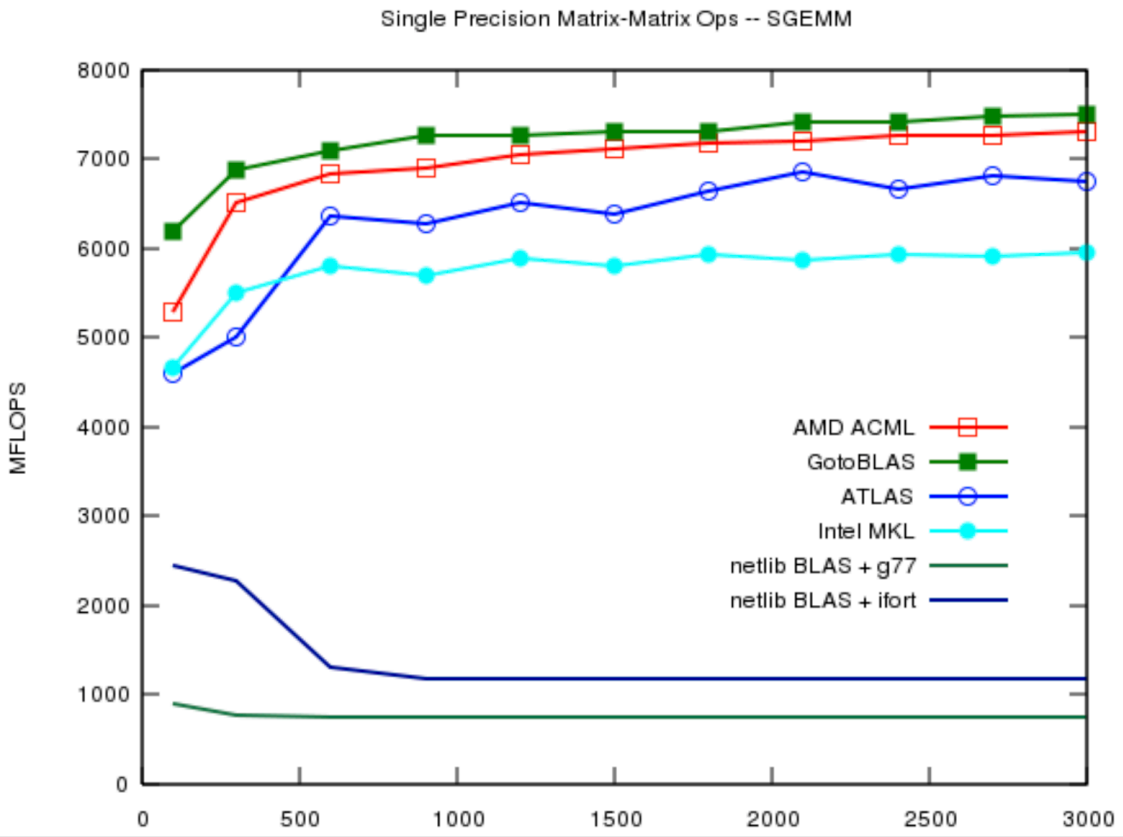
Also included in Level 3 are routines for solving

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

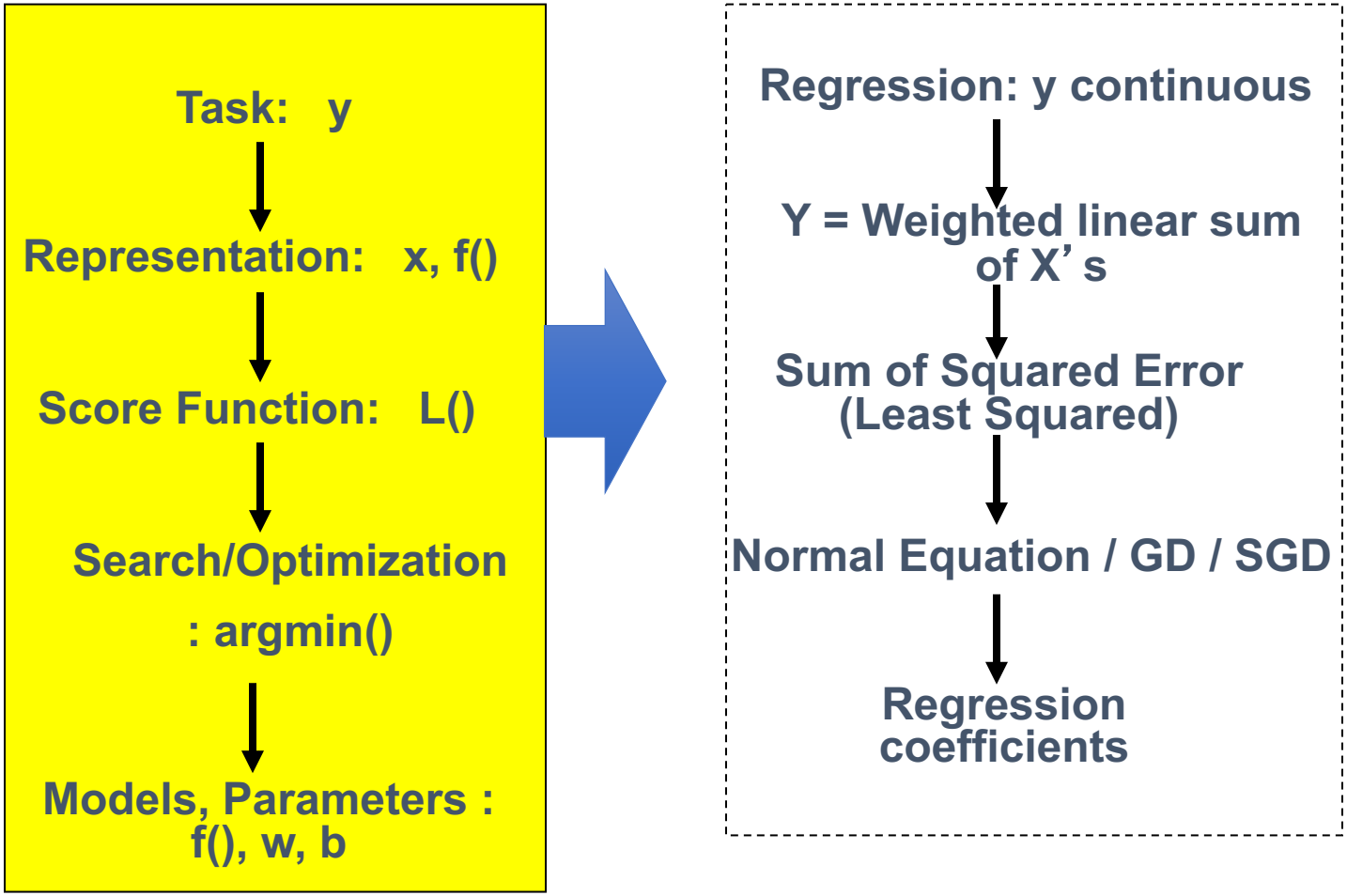
where $\boldsymbol{T}$ is a triangular matrix, among other functionality.

BLAS performance is very much system dependent, e.g., https://www.hoffman2.idre.ucla.edu/blas_benchmark/

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



Today: Multivariate Linear Regression in a Nutshell



Next: More Regression (supervised)

- Four ways to train / perform optimization for linear regression models
  - Normal Equation
  - Gradient Descent (GD)
  - Stochastic GD
  - Newton's method

} variations of $\arg\min_{\theta} L(\theta)$

- Supervised regression models
  - Linear regression (LR)
  - LR with non-linear basis functions
  - Locally weighted LR
  - LR with Regularizations

} variations of $f(x)$

→ variations of $L(\theta)$

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 (please read)

❑ Prof. Alexander Gray's slides

EXTRA

Review (I):

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

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

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

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

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

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

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

Review(I) :

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

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

Review (II): gradient of linear form

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

One Concrete Example

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

$$\frac{\partial f}{\partial w_1} = 1$$
$$\frac{\partial f}{\partial w_2} = 2$$
$$\frac{\partial f}{\partial w_3} = 3$$

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

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
	Gradient
Second-order derivative	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

- 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
  - rank(A)<= min(m,n)
  - If n=rank(A), then A has full row rank
  - If m=rank(A), then A has full column rank

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$ Rank=?

$$\begin{pmatrix} 2 & 1 \\ 4 & 2 \end{pmatrix}$$ Rank=?

If A is n*n, rank(A)=n iff A is invertible

rank(AB) <= min(rank(A), rank(B))

Extra: positive semi-definite!

L2-Note: Page 17

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

If $x^T A x \geq 0$

$1 \times n \quad n \times n \quad n \times 1$

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

If $X^T A X > 0$

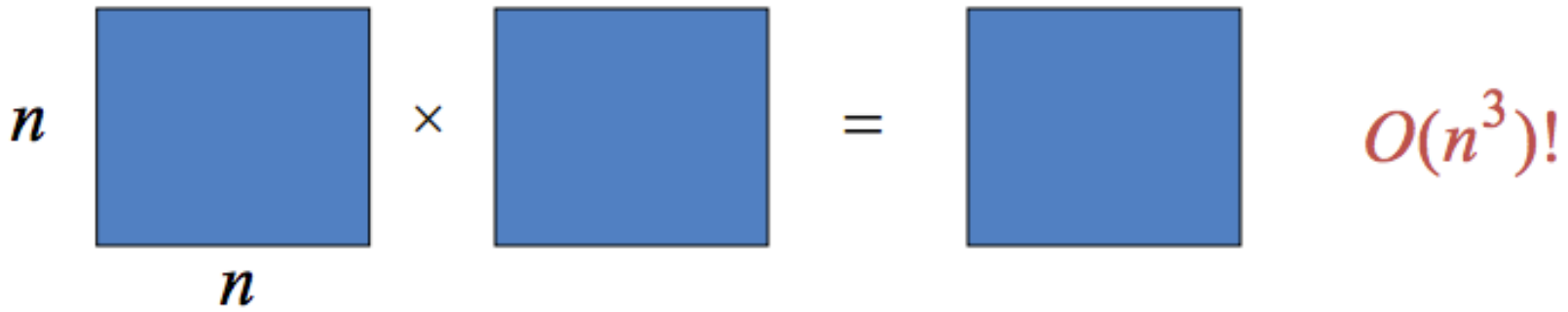
$\Rightarrow$ A is PD $\Rightarrow$ full rank / invertible

See proof on L2-Note: Page 18

Extra: Scalability to big data?

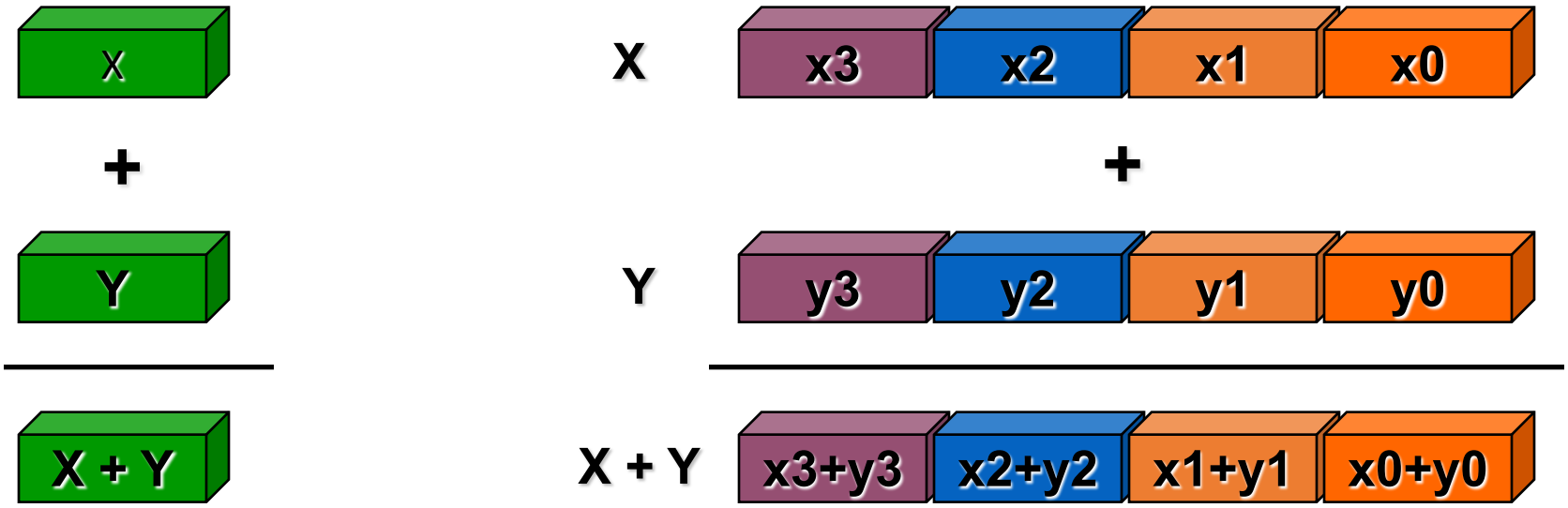
- Traditional CS view: Polynomial time algorithm, Wow!
- Large-scale learning: Sometimes even O(n) is bad! => Many state-of-the-art solutions (e.g., low rank, sparse, hardware, sampling, randomized…)

Simple example: Matrix multiplication



SIMD: Single Instruction, Multiple Data

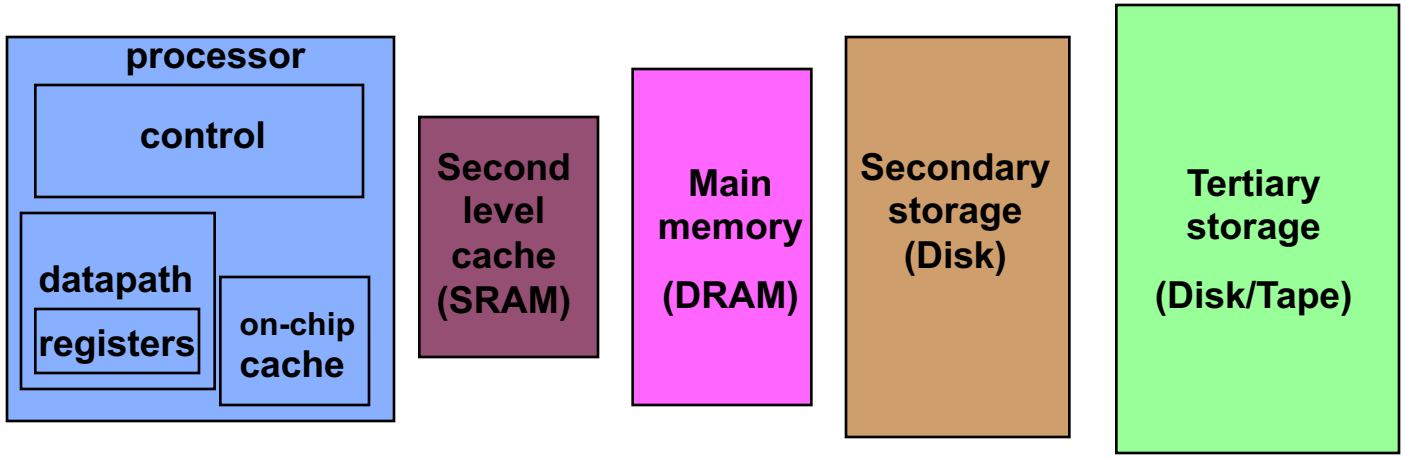
- Scalar processing
  - traditional mode
  - one operation produces one result
- SIMD processing
  - with SSE / SSE2
  - SSE = streaming SIMD extensions
  - one operation produces multiple results



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

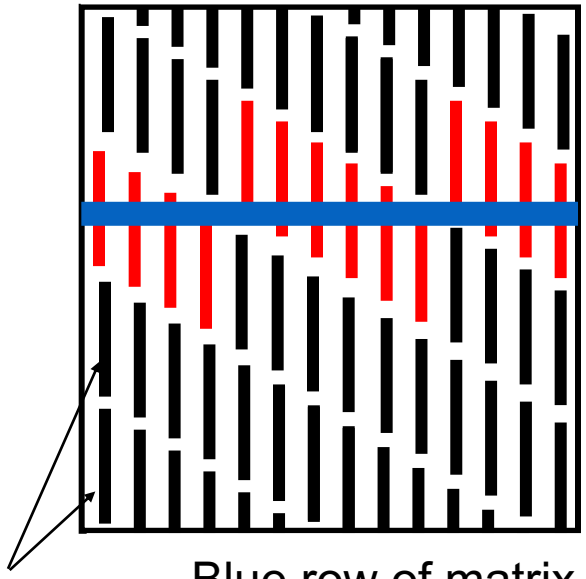
0	5	10	15
1	6	11	16
2	7	12	17
3	8	13	18
4	9	14	19

Row major

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

Column major (for now)

Column major matrix in memory



cachelines

Blue row of matrix is stored in red cachelines

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 2x2 matrix multiply, normally takes 8 multiplies, 4 adds
  - Strassen does it with 7 multiplies and 18 adds

Let $M = \begin{pmatrix} m11 & m12 \\ m21 & m22 \end{pmatrix} = \begin{pmatrix} a11 & a12 \\ a21 & a22 \end{pmatrix} \begin{pmatrix} b11 & b12 \\ b21 & b22 \end{pmatrix}$

Let p1 = (a12 - a22) * (b21 + b22)

p2 = (a11 + a22) * (b11 + b22)

p3 = (a11 - a21) * (b11 + b12)

p4 = (a11 + a12) * b22

p5 = a11 * (b12 - b22)

p6 = a22 * (b21 - b11)

p7 = (a21 + a22) * b11

Then m11 = p1 + p2 - p4 + p6

m12 = p4 + p5

m21 = p6 + p7

m22 = p2 - p3 + p5 - p7

Extends to nxn by divide&conquer

Strassen (continued)

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

- 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 = $\Omega(n^{\log_2 7} / M^{(\log_2 7)/2 - 1}) \sim \Omega(n^{2.81} / M^{0.4})$
  - (Ballard, D., Holtz, Schwartz, 2011)
  - Attainable too