EECE695D: Efficient ML Systems

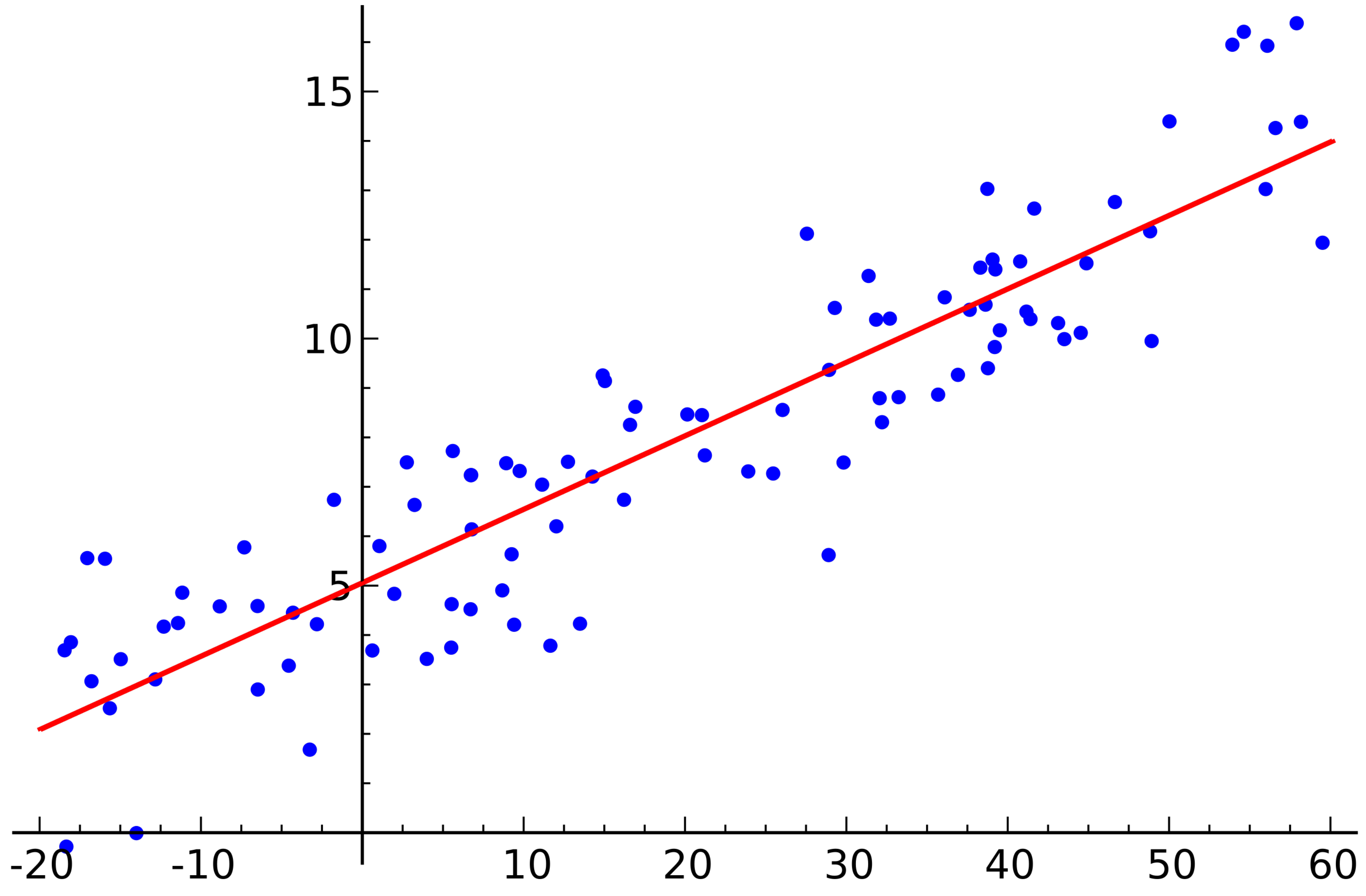
Compute / Memory: Linear Models

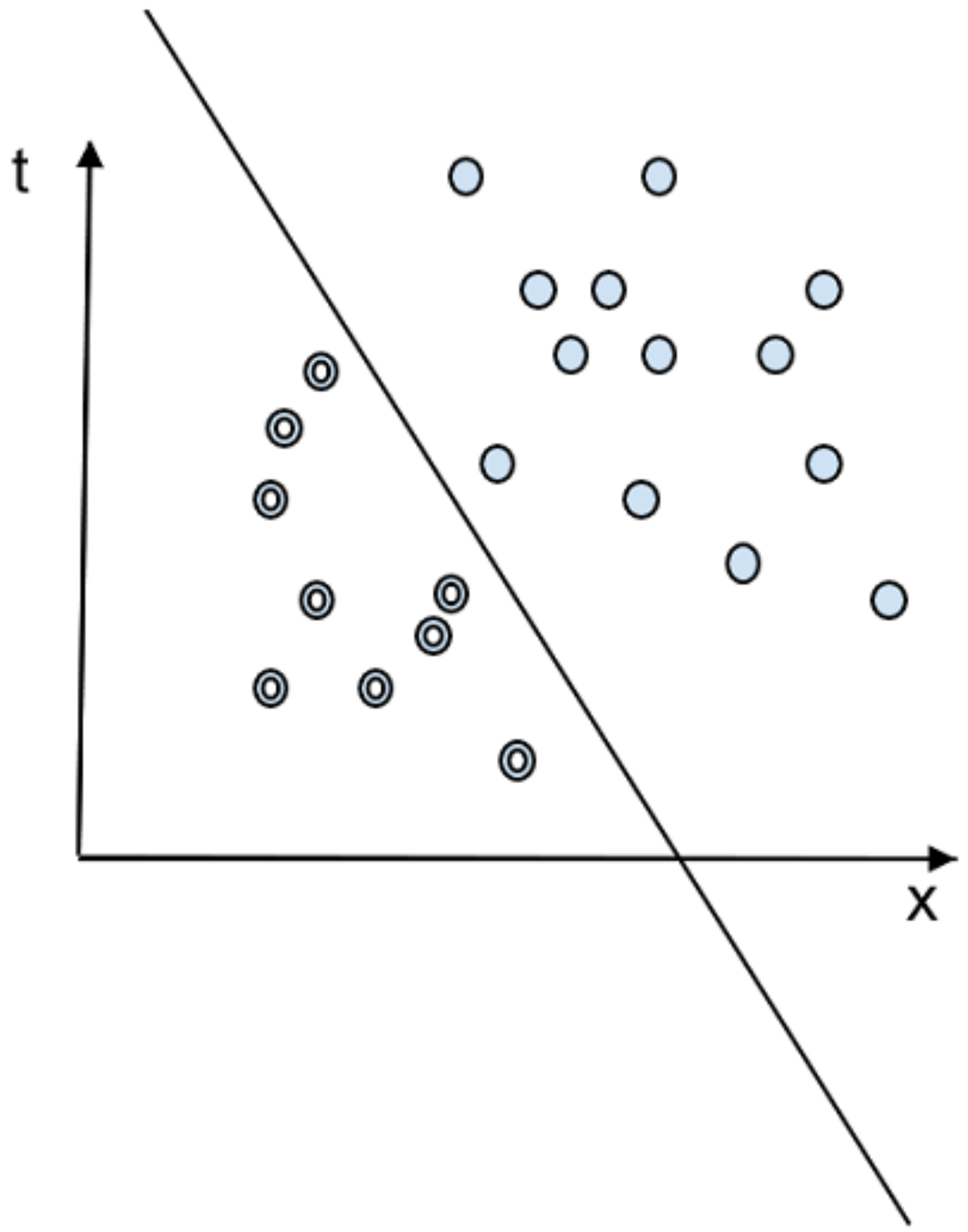
Pre-Class Announcements

- **Today's office hour.** 4:00PM-4:45PM, @ Terarosa coffee # Find me!
- **HW#0 due.** Sunday night (11/50+) # Do early and get ready for Chuseok!

Linear Models

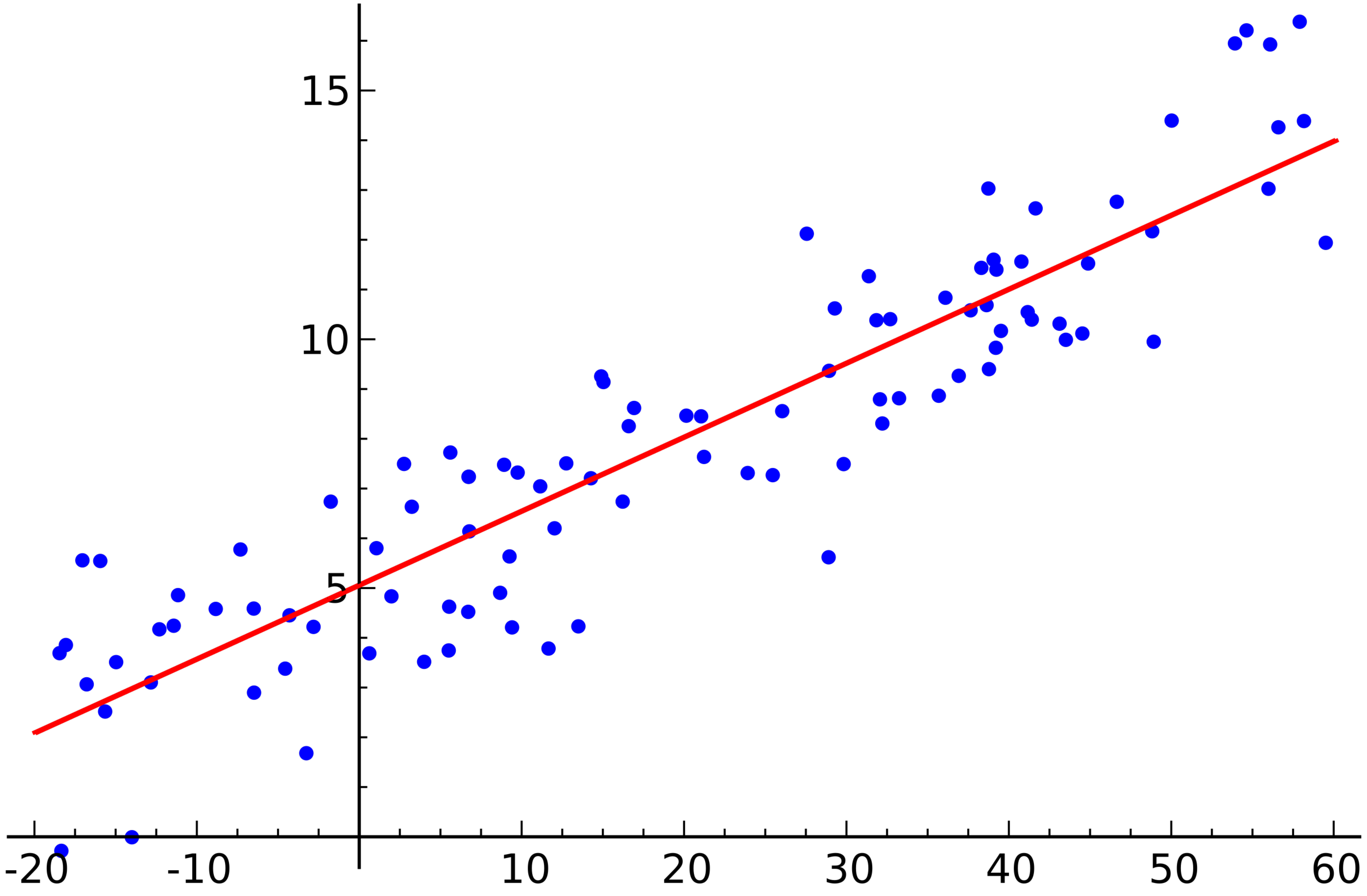
- As a warm-up, we revisit Linear Regression, and Perceptron.
  - **Linear models.** Overly simplified neural networks!
  - **Focus.** Compute / memory for training & inference of these models.
    (+ refresh your memory on basic stuffs)





https://en.wikipedia.org/wiki/Linear_regression
https://www.includehelp.com/python/perceptron-algorithm-and-its-implementation.aspx

Example 1: Linear Regression



Dataset. $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$,
given the features $\mathbf{x}_i \in \mathbb{R}^d$
and the labels $y_i \in \mathbb{R}$.

Model. $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$,
given the model params $\mathbf{w} \in \mathbb{R}^d$

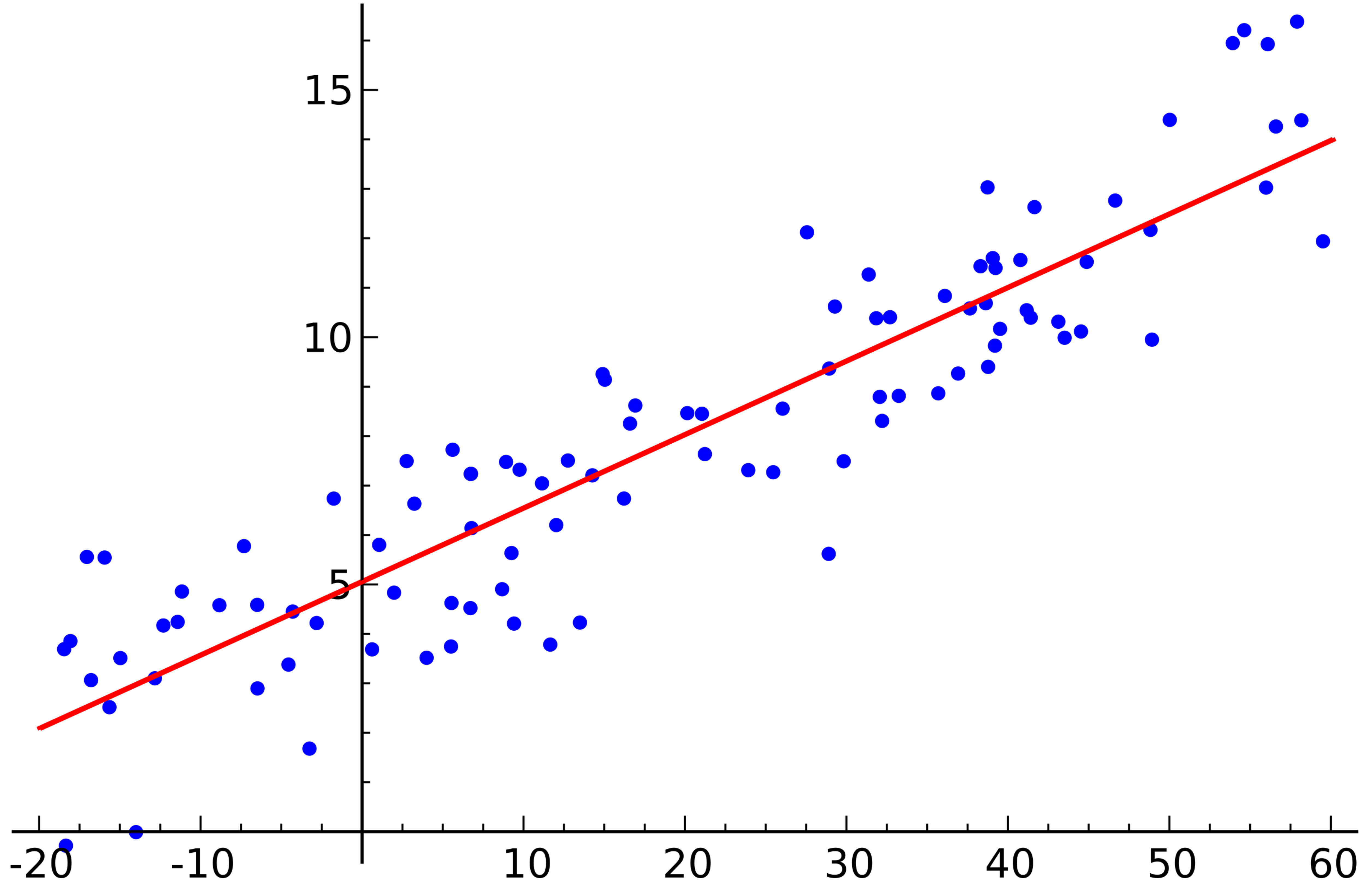
Note 1. Implicitly constrained capability for "overparameterization" (model dim. = data dim.)

Note 2. Also handles "bias terms:"

$$\tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}, \quad \tilde{\mathbf{w}} = \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}$$

https://en.wikipedia.org/wiki/Linear_regression

Example 1: Linear Regression



https://en.wikipedia.org/wiki/Linear_regression

Loss. (Rescaled) Mean-squared error

$$R(\mathbf{w}, D) = \sum_{i=1}^{N} (y - \mathbf{w}^\top \mathbf{x})^2$$

Note. One can rewrite this via "stacking."

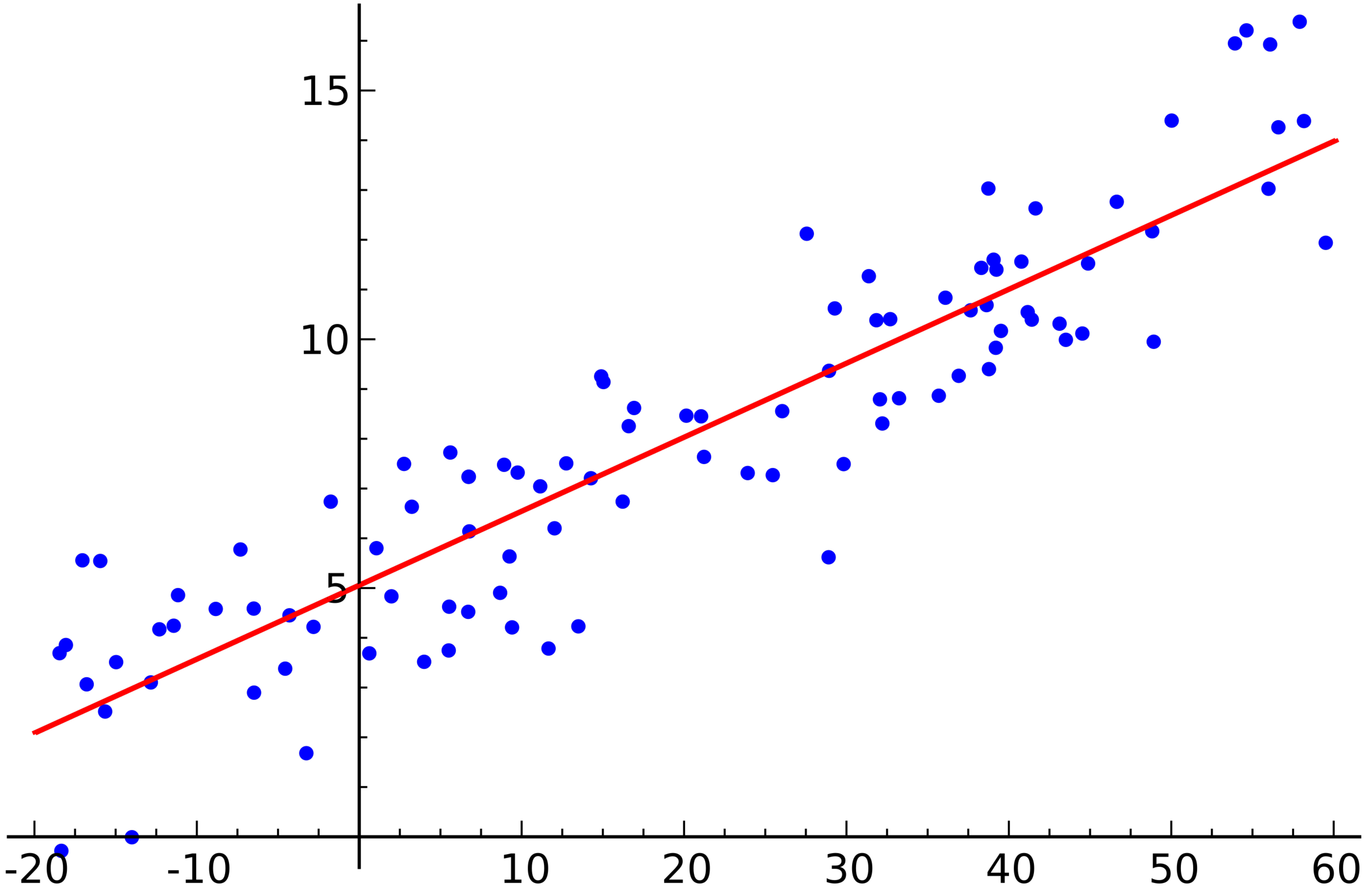
$$R(\mathbf{w}, D) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$$

where

$$\mathbf{X} := \begin{bmatrix} [\ \Leftarrow \mathbf{x}_1 \Rightarrow\ ] \\ [\ \Leftarrow \mathbf{x}_2 \Rightarrow\ ] \\ \cdots \\ [\ \Leftarrow \mathbf{x}_N \Rightarrow\ ] \end{bmatrix} \in \mathbb{R}^{N \times d}$$

$$\mathbf{y} := [y_1, y_2, \ldots, y_N]^\top \in \mathbb{R}^{N \times 1}$$

Example 1: Linear Regression



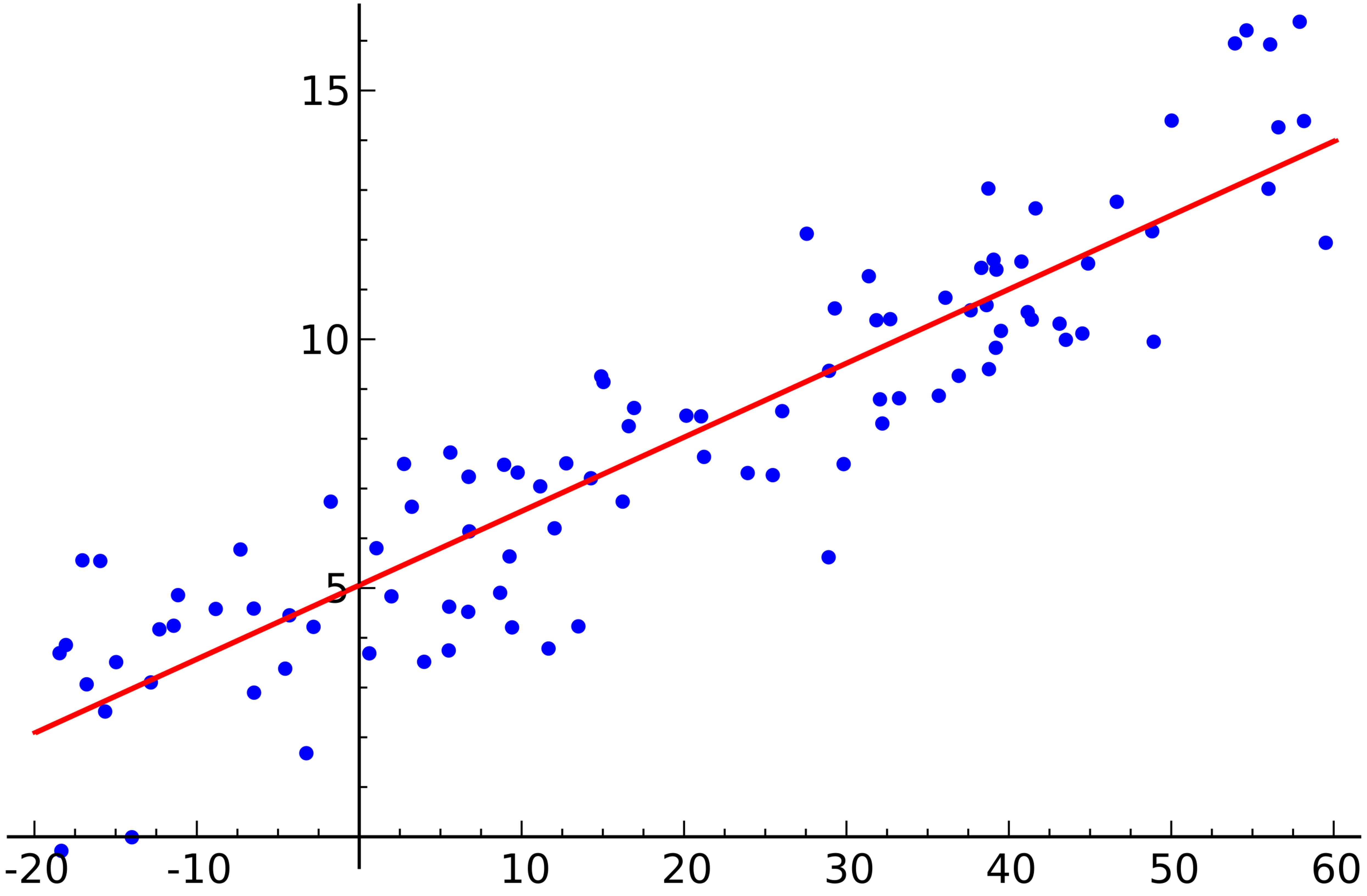
Q. Cost of <u>inference</u>?

$$f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$$

Compute? Memory?

https://en.wikipedia.org/wiki/Linear_regression

Example 1: Linear Regression



https://en.wikipedia.org/wiki/Linear_regression

Q. Cost of <u>inference</u>?

$$f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$$

Compute? Memory?

Compute.
We perform d multiplications and d additions.
(Total $2d$ FLOPs)

$$s \leftarrow 0$$
$$s \leftarrow s + x_1 \times w_1$$
$$s \leftarrow s + x_2 \times w_2$$
$$\cdots$$
$$s \leftarrow s + w_d \times x_d$$

Note. FLOPs = Floating Point Operations
1 addition of FP numbers = 1 FLOP
1 multiplication of FP numbers = 1 FLOP

Note. Often fused as **MAC** (**M**ultiply-**AC**cumulation)
+ Single rounding for better performance.
- Parallel computing

sign	exponent	fraction

$$\text{value} = (\text{sign}) \times 2^{(\text{exponent})} \times 1.(\text{fraction})$$

float

	8 bits	23 bits
0	01111110	.01010101010101010101011

FP32 Representation of $+\frac{2}{3}$

NVIDIA White Paper "Precision and Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs," fetched Aug. 2022

$$A = 2^{1} \times 1.00000000000000000000001$$
$$B = 2^{0} \times 1.00000000000000000000001$$
$$C = 2^{3} \times 1.00000000000000000000001$$

$$A + B + C = 2^{3} \times 1.0110000000000000000000101100\ldots$$
$$\mathrm{rn}(\mathrm{rn}(A + B) + C) = 2^{3} \times 1.01100000000000000000010$$
$$\mathrm{rn}(A + \mathrm{rn}(B + C)) = 2^{3} \times 1.01100000000000000000001$$

i.e., rounding matters!

NVIDIA White Paper "Precision and Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs," fetched Aug. 2022

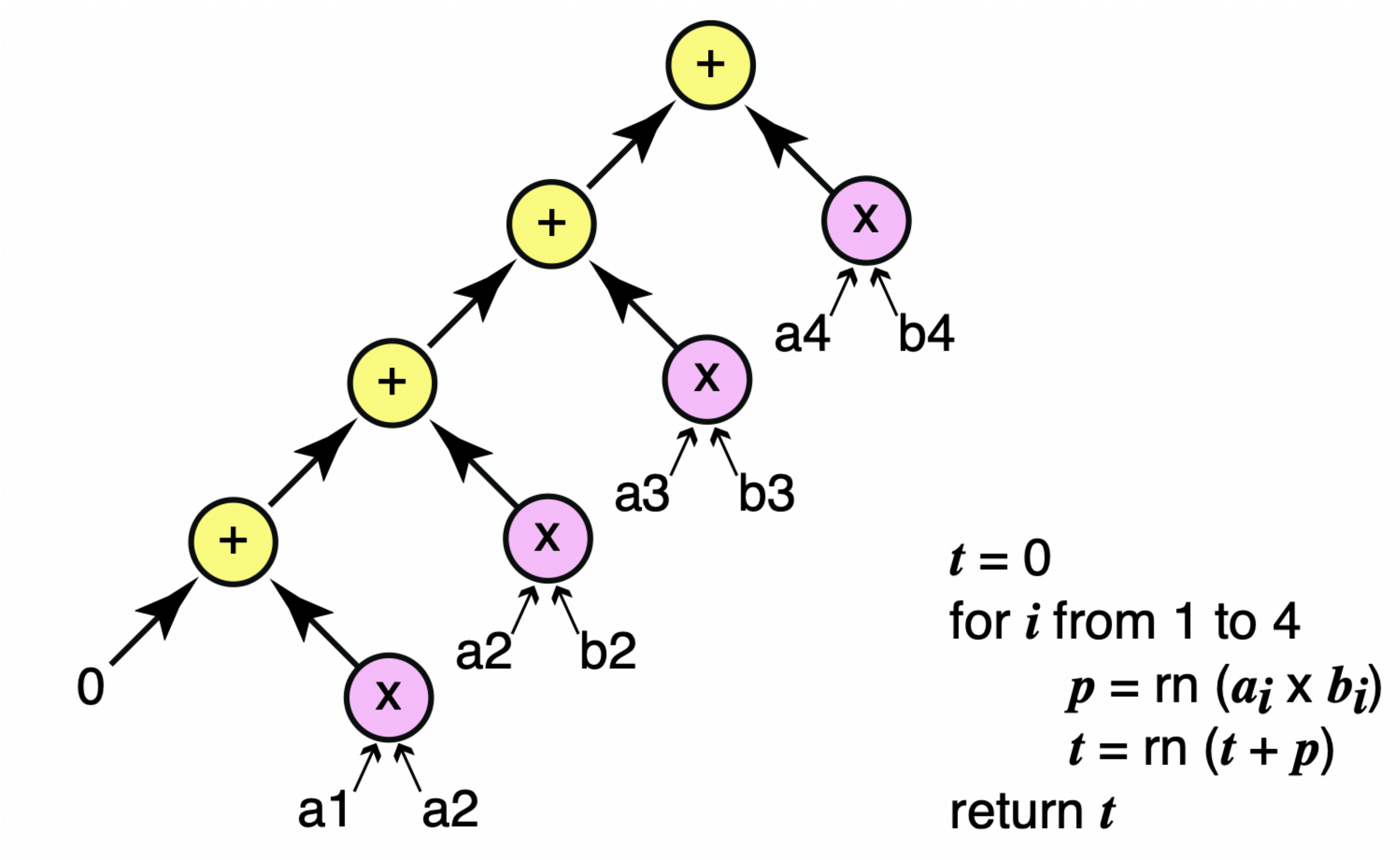


Fig 1. Vanilla Multiply-Add

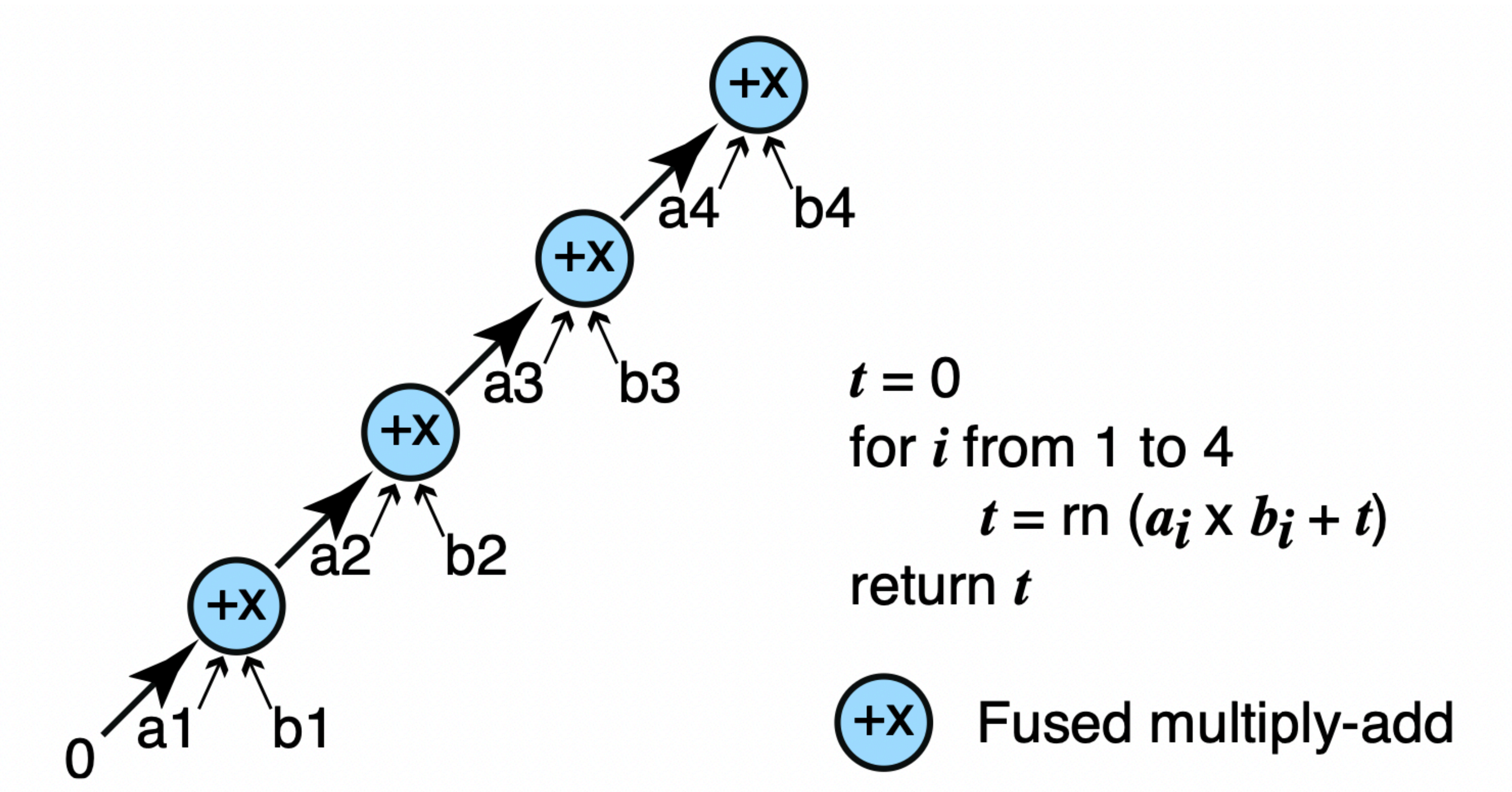


Fig 2. Fused Multiply-Add

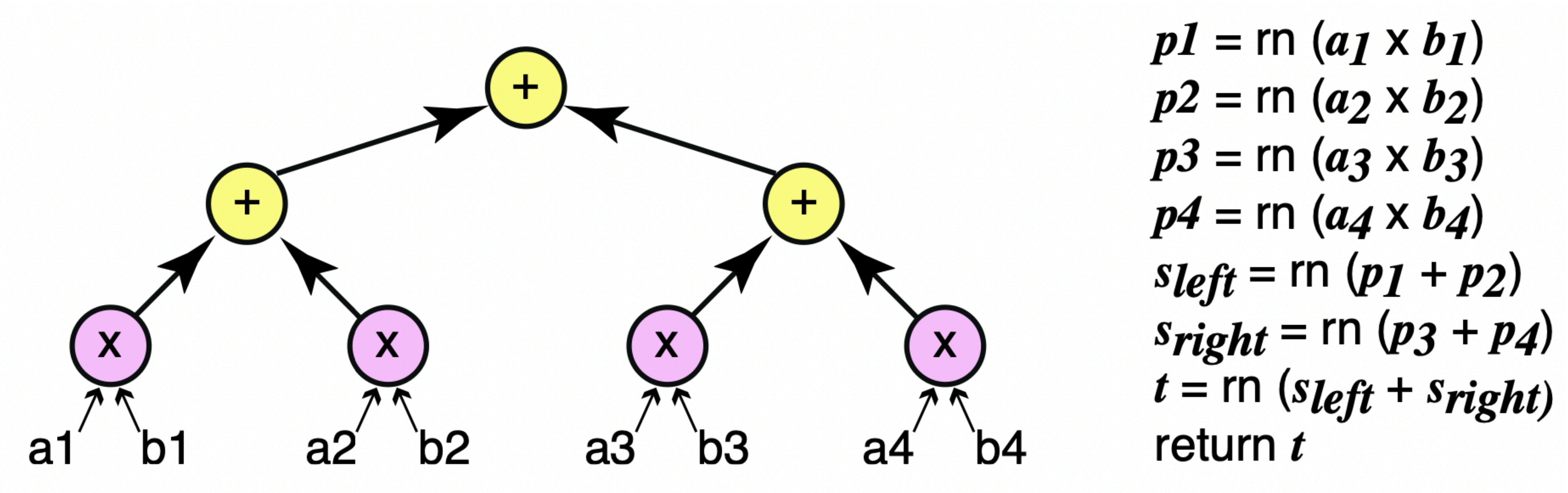


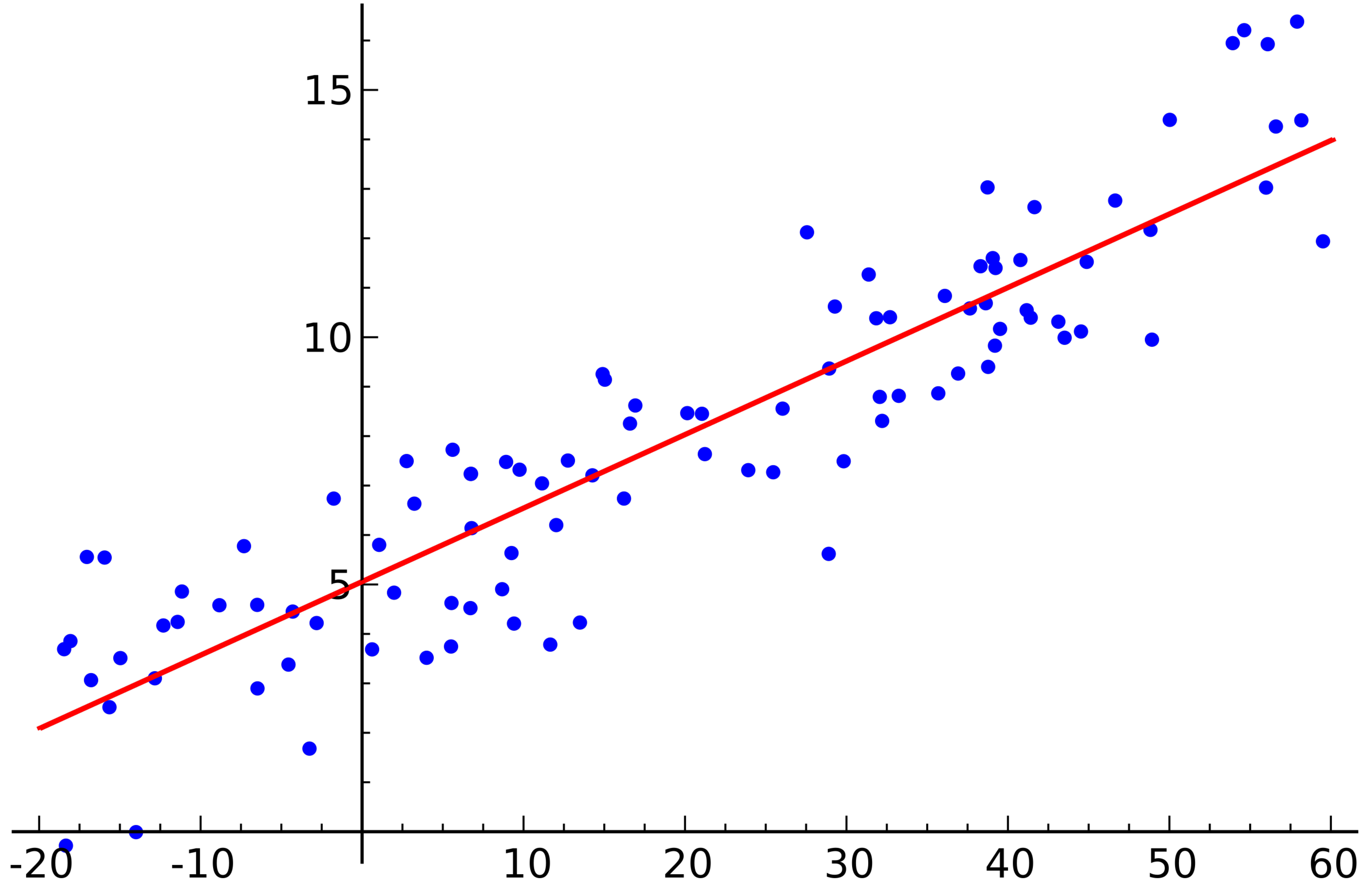
Fig 3. Parallel Multiply-Add

method	result	float value
exact	.0559587528435...	0x3D65350158...
serial	.0559588074	0x3D653510
FMA	.0559587515	0x3D653501
parallel	.0559587478	0x3D653500

Accuracy-Speed Tradeoff!
(potentially)

NVIDIA White Paper “Precision and Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs,” fetched Aug. 2022

Example 1: Linear Regression



https://en.wikipedia.org/wiki/Linear_regression

Q. Cost of <u>inference</u>?

$$f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$$

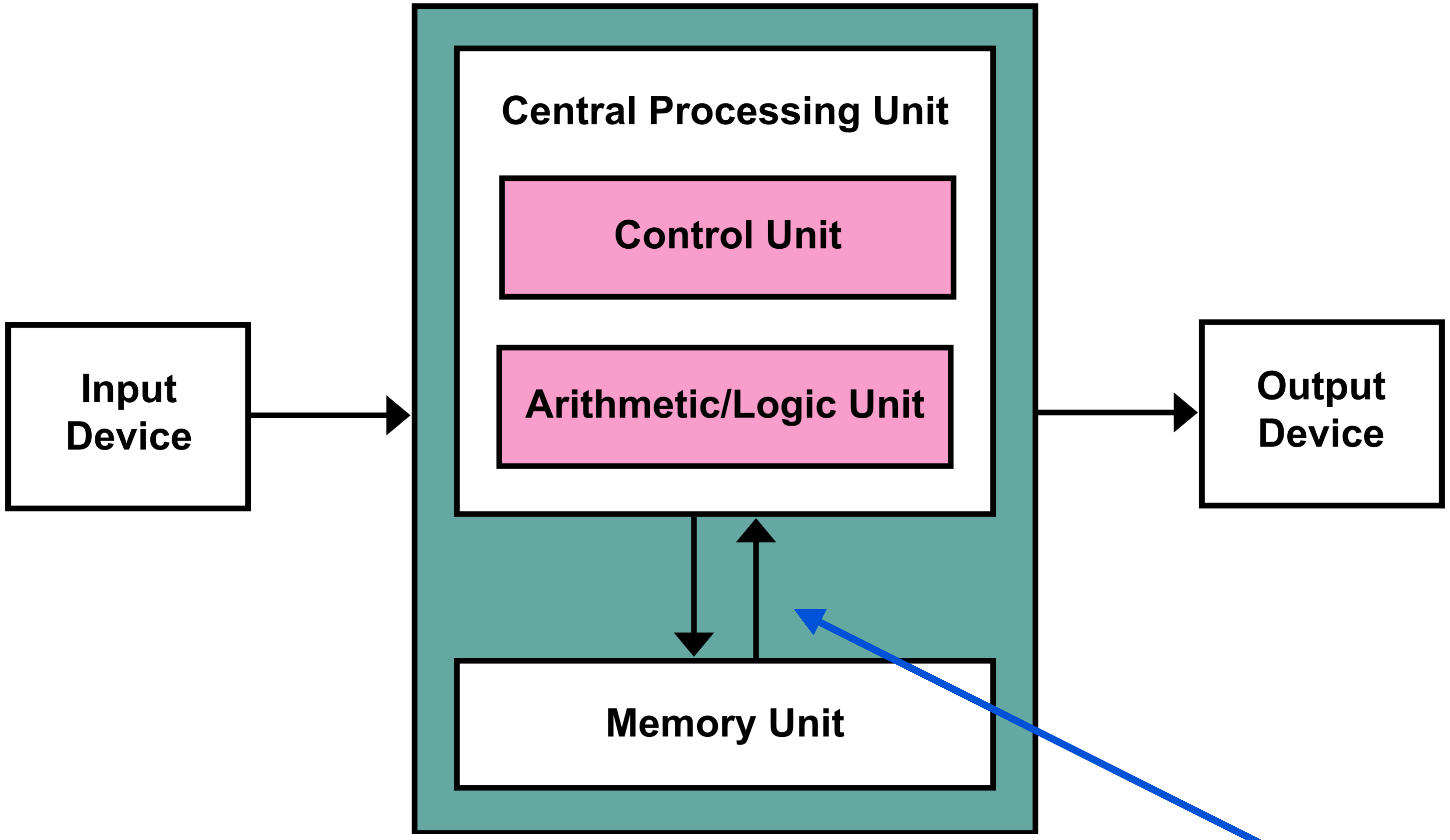
Compute? Memory?

Memory.

- Need to load parameters $\mathbf{w} \in \mathbb{R}^d$
- Need to load data $\mathbf{x} \in \mathbb{R}^d$
- Need to allocate sum $s \in \mathbb{R}$

$$s \leftarrow 0$$
$$s \leftarrow s + x_1 \times w_1$$
$$s \leftarrow s + x_2 \times w_2$$
$$\cdots$$
$$s \leftarrow s + w_d \times x_d$$

Note. In cases where your cache is not big enough, expect a *von Neumann bottleneck*.

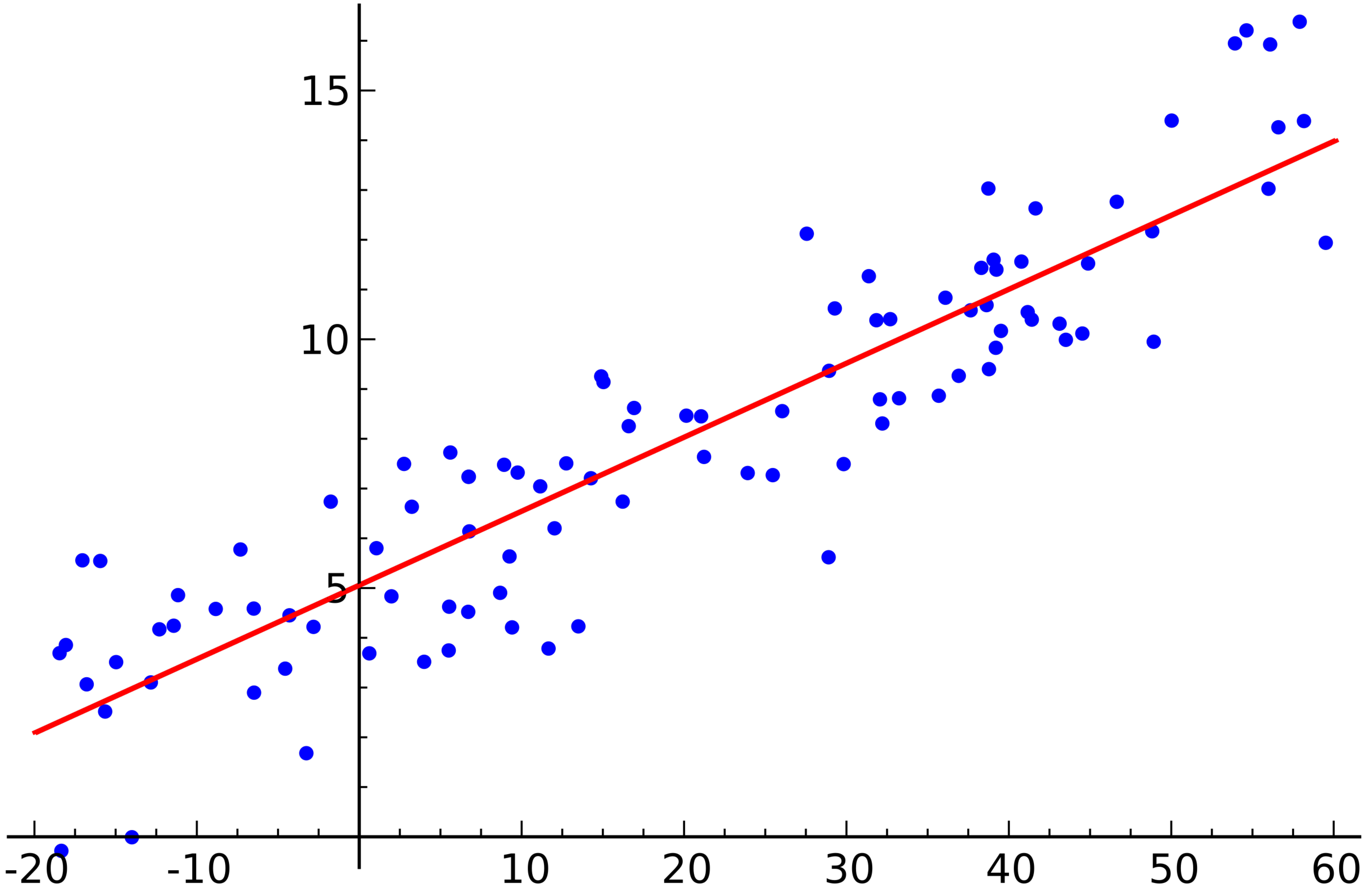


Note. This and cache should affect whether the algorithm is compute- or memory-bound. (will come back to this later)

Note. Better not leave idle—major source of power consumption

https://en.wikipedia.org/wiki/Von_Neumann_architecture

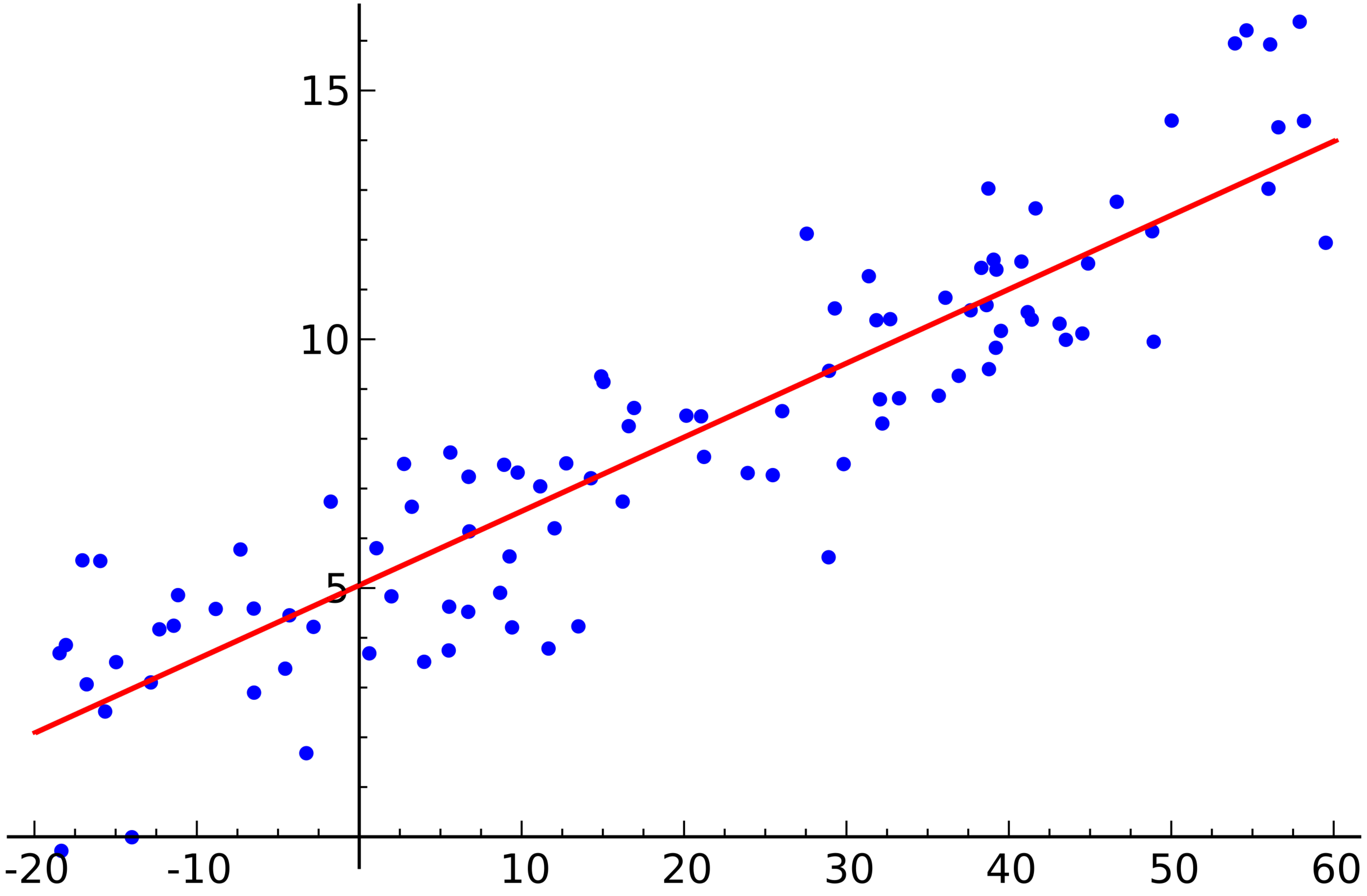
Example 1: Linear Regression



Q. Cost of training?

https://en.wikipedia.org/wiki/Linear_regression

Example 1: Linear Regression



Q. Cost of <u>training</u>?

This really depends on "how" one optimizes the loss, i.e., how one solves the quadratic program

$$\min_{\mathbf{w}\in\mathbb{R}^d} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$$

(1) Solve it analytically—we know linear algebra!

(2) Use indirect method—iterative optimization (e.g., gradient descent)

https://en.wikipedia.org/wiki/Linear_regression

Analytic method

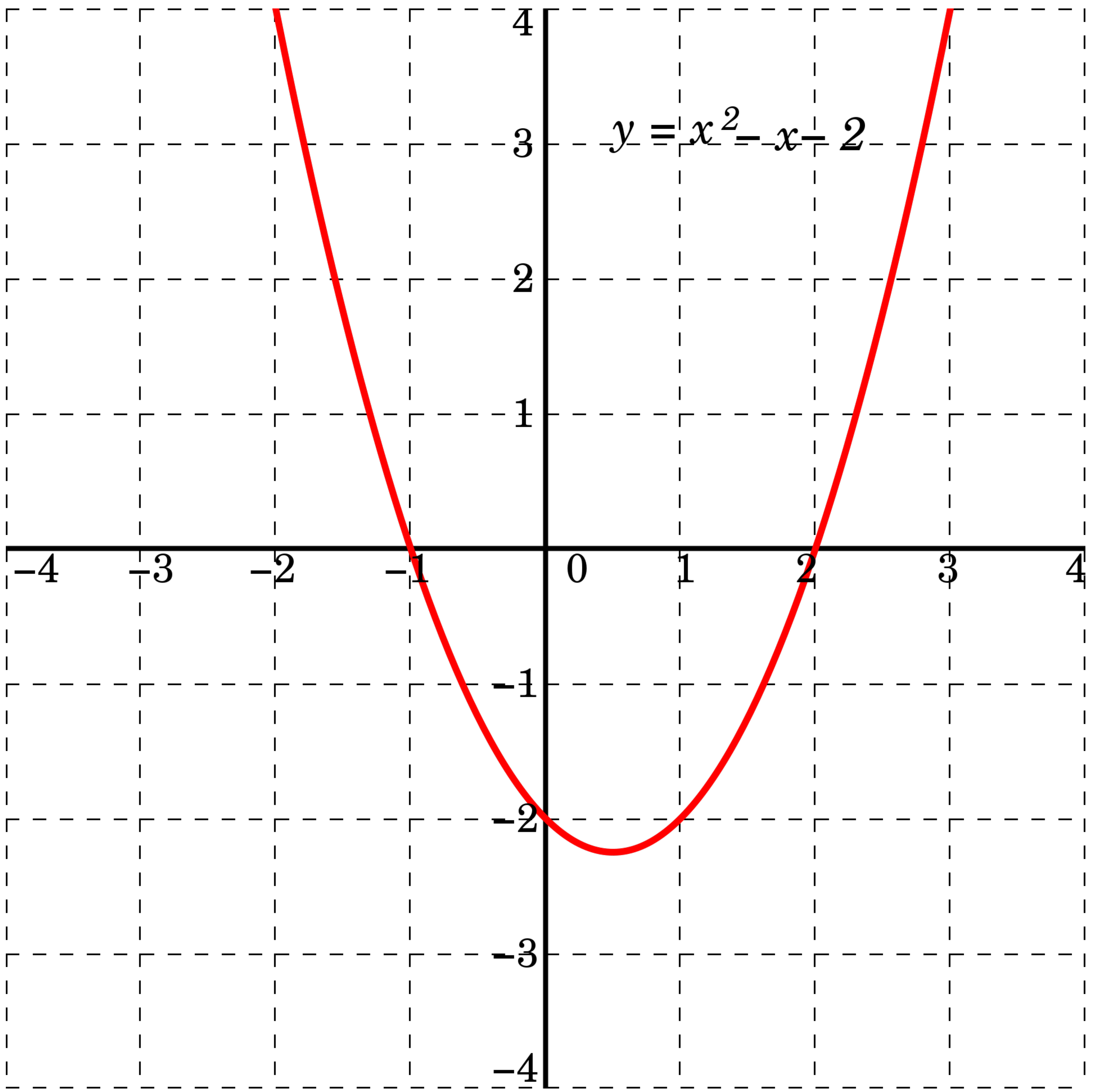
$$\boxed{\min_{\mathbf{w}\in\mathbb{R}^d} \| \mathbf{y} - \mathbf{X}\mathbf{w} \|^2 = \min_{\mathbf{w}\in\mathbb{R}^d} \left(\mathbf{y}^\top\mathbf{y} + \mathbf{w}^\top\mathbf{X}^\top\mathbf{X}\mathbf{w} - 2\mathbf{w}^\top\mathbf{X}^\top\mathbf{y}\right)}$$

Case. $d = 1$

$$\min_{w\in\mathbb{R}} \sum_{i=1}^{N} (y_i - wx_i)^2 = \min_{w\in\mathbb{R}} \left(Aw^2 + Bw + C\right), \quad A > 0$$

- Unconstrained optimization ⇒ Check only the critical point!
- The optimal point is

$$w^* = \frac{-B}{2A} = \frac{\sum_{i=1}^{N} x_i y_i}{\sum_{i=1}^{N} x_i^2}$$



Analytic method

$$\min_{\mathbf{w}\in\mathbb{R}^d} \left\| \mathbf{y} - \mathbf{X}\mathbf{w} \right\|^2 = \min_{\mathbf{w}\in\mathbb{R}^d} \left(\mathbf{y}^\top\mathbf{y} + \mathbf{w}^\top\mathbf{X}^\top\mathbf{X}\mathbf{w} - 2\mathbf{w}^\top\mathbf{X}^\top\mathbf{y}\right)$$

Case. $d \geq 2$

- The optimum achieved at the point where

$$\nabla_{\mathbf{w}}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 = \mathbf{0}$$

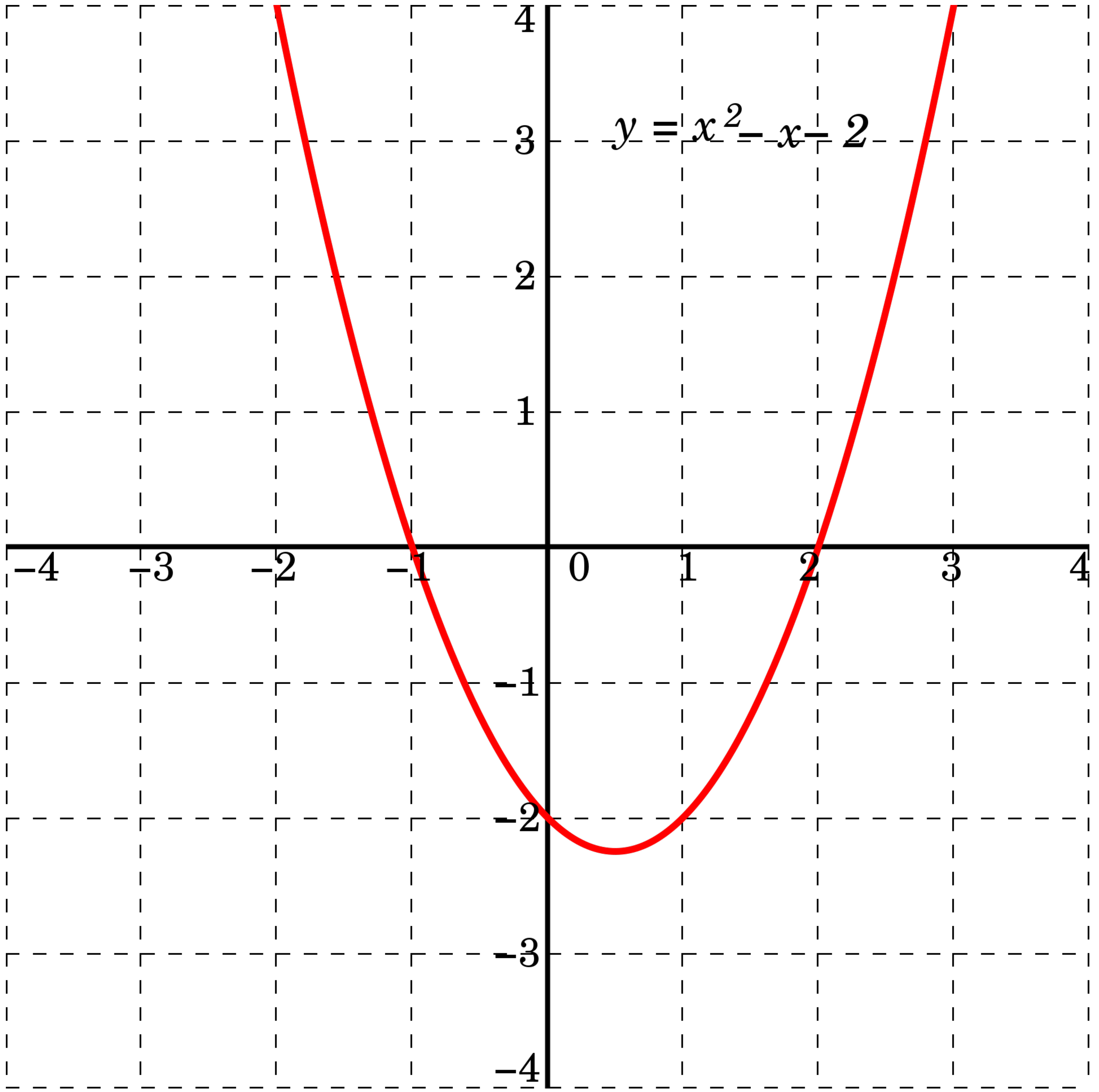
- Equivalent to:

$$\mathbf{X}^\top\mathbf{X}\mathbf{w} = \mathbf{X}^\top\mathbf{y}$$

- Whenever $\mathbf{X}^\top\mathbf{X}$ is invertible, we have the unique optimizer

$$\mathbf{w}^* = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$$

Q. Compute / Memory?



Analytic method

$$\boxed{\mathbf{w}^* = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}}$$

We do this in four steps:

- Matrix-matrix multiplication $\mathbf{A} \leftarrow \mathbf{X}^\top\mathbf{X}$
- Matrix-vector multiplication $\mathbf{b} \leftarrow \mathbf{X}^\top\mathbf{y}$
- Matrix inversion $\mathbf{C} \leftarrow \mathbf{A}^{-1}$
- Matrix-vector multiplication $\mathbf{w}^* = \mathbf{C}\mathbf{b}$

Note 1. Never confuse the order!
Otherwise you need to do
2 matrix-matrix multiplication and
1 matrix-vector multiplication

Note 2. People avoid doing "matrix inverse"
—considered numerically unstable—
and solve $\mathbf{A}\mathbf{w} = \mathbf{b}$ directly.

Analytic method

$$\boxed{\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}}$$

We do this in four steps:

- Matrix-matrix multiplication $\mathbf{A} \leftarrow \mathbf{X}^\top \mathbf{X}$
- Matrix-vector multiplication $\mathbf{b} \leftarrow \mathbf{X}^\top \mathbf{y}$
- Matrix inversion $\mathbf{C} \leftarrow \mathbf{A}^{-1}$
- Matrix-vector multiplication $\mathbf{w}^* = \mathbf{C}\mathbf{b}$

Q. How many FLOPs?

$\mathbf{X} \mapsto \mathbf{X}^\top \mathbf{X}$ where $\mathbf{X} \in \mathbb{R}^{N \times d}$

Analytic method

$$\boxed{\mathbf{w}^* = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}}$$

We do this in four steps:

- Matrix-matrix multiplication $\mathbf{A} \leftarrow \mathbf{X}^\top\mathbf{X}$
- Matrix-vector multiplication $\mathbf{b} \leftarrow \mathbf{X}^\top\mathbf{y}$
- Matrix inversion $\mathbf{C} \leftarrow \mathbf{A}^{-1}$
- Matrix-vector multiplication $\mathbf{w}^* = \mathbf{C}\mathbf{b}$

Q. How many FLOPs?

$\mathbf{X} \mapsto \mathbf{X}^\top\mathbf{X}$ where $\mathbf{X} \in \mathbb{R}^{N\times d}$

Naïvely, we need $2d^2N$ FLOPs.

$$\begin{bmatrix} \boxed{\sum_{i=1}^N x_{i1}x_{1i}} & \sum_{i=1}^N x_{i1}x_{2i} & \cdots & \sum_{i=1}^N x_{i1}x_{di} \\ \sum_{i=1}^N x_{i2}x_{1i} & \sum_{i=1}^N x_{i2}x_{2i} & \cdots & \sum_{i=1}^N x_{i2}x_{di} \\ \cdots & \cdots & \cdots & \cdots \\ \boxed{\sum_{i=1}^N x_{id}x_{1i}} & \sum_{i=1}^N x_{id}x_{2i} & \cdots & \sum_{i=1}^N x_{id}x_{di} \end{bmatrix}$$

Each entry needs $2N$ FLOPs

Need to compute d^2 entries

Why "naïvely"?

Multiplying two $d \times d$ matrices requires less compute than $\mathcal{O}(d^3)$.
Ancient result (Strassen). $\sim \mathcal{O}(d^{2.807})$
Recent result (SODA 2021). $\sim \mathcal{O}(d^{2.37286})$
Idea: Divide matrices into blocks, and multiply submatrices

Also, we are multiplying $\mathbf{X}^\top$ and $\mathbf{X}$, not just any matrix...
Can we do any better? :)

Does it give a "speedup" with CPU / GPU?

Asymptotic order... really careful automation with compilers

Analytic method

$$\boxed{\mathbf{w}^* = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}}$$

We do this in four steps:

- Matrix-matrix multiplication $\mathbf{A} \leftarrow \mathbf{X}^\top\mathbf{X}$
- Matrix-vector multiplication $\mathbf{b} \leftarrow \mathbf{X}^\top\mathbf{y}$
- Matrix inversion $\mathbf{C} \leftarrow \mathbf{A}^{-1}$
- Matrix-vector multiplication $\mathbf{w}^* = \mathbf{C}\mathbf{b}$

Q. How much memory?

Load. dN Floating points.

New. d^2 Floating points.

Analytic method

$$\boxed{\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}}$$

We do this in four steps:

- Matrix-matrix multiplication $\quad \mathbf{A} \leftarrow \mathbf{X}^\top \mathbf{X}$
- Matrix-vector multiplication $\quad \mathbf{b} \leftarrow \mathbf{X}^\top \mathbf{y}$
- Matrix inversion $\quad \mathbf{C} \leftarrow \mathbf{A}^{-1}$
- Matrix-vector multiplication $\quad \mathbf{w}^* = \mathbf{C}\mathbf{b}$

Q. How many FLOPs?

$\mathbf{A} \mapsto \mathbf{A}^{-1} \qquad$ where $\mathbf{A} \in \mathbb{R}^{d \times d}$

Analytic method

$$\boxed{\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}}$$

We do this in four steps:

- Matrix-matrix multiplication $\mathbf{A} \leftarrow \mathbf{X}^\top \mathbf{X}$
- Matrix-vector multiplication $\mathbf{b} \leftarrow \mathbf{X}^\top \mathbf{y}$
- Matrix inversion $\mathbf{C} \leftarrow \mathbf{A}^{-1}$
- Matrix-vector multiplication $\mathbf{w}^* = \mathbf{C}\mathbf{b}$

Q. How many FLOPs?

$\mathbf{A} \mapsto \mathbf{A}^{-1}$ where $\mathbf{A} \in \mathbb{R}^{d \times d}$

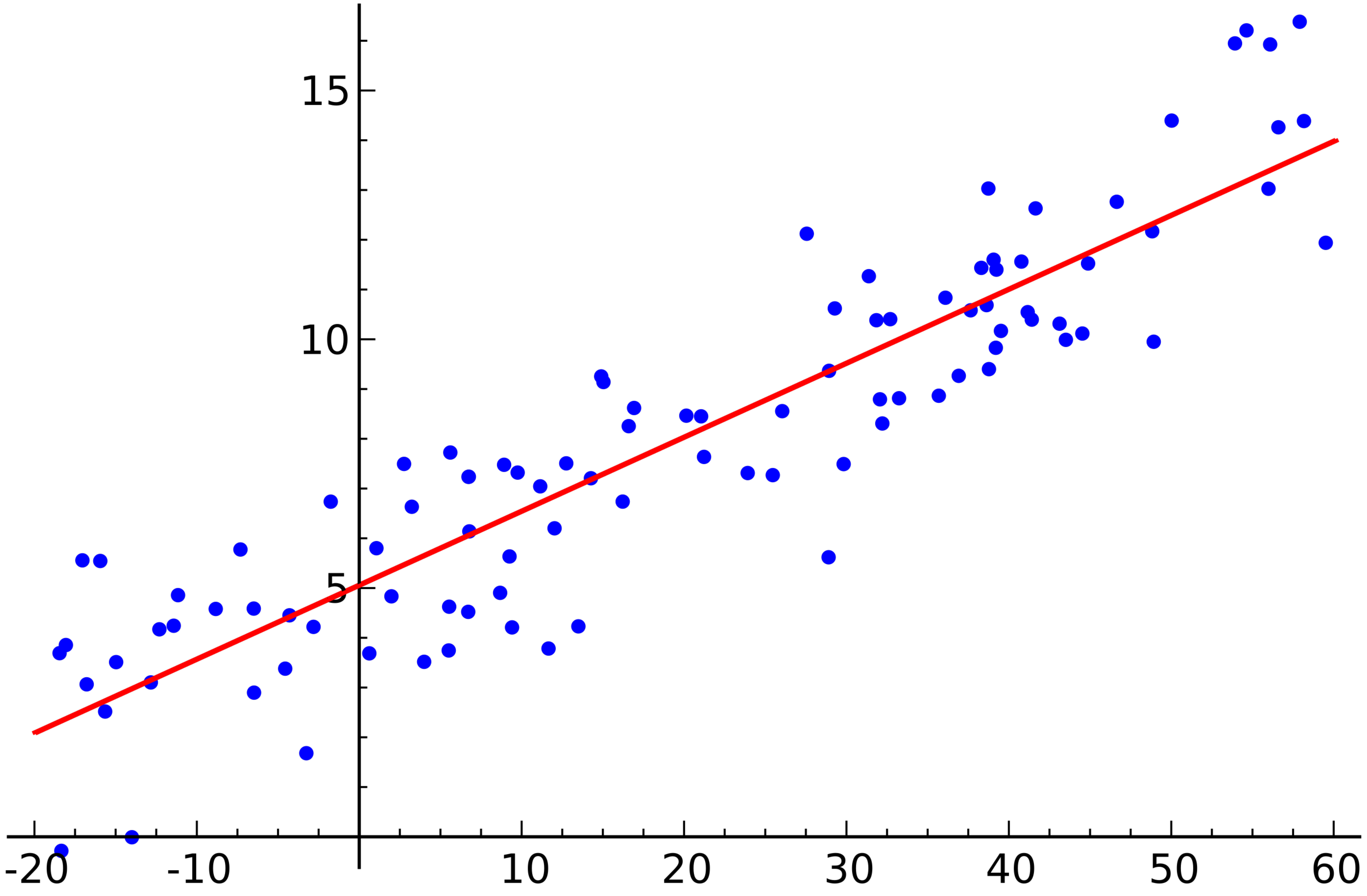
If we use LU decomposition, we need $\sim \frac{2}{3} d^3$ FLOPs.

Note. One can also use Strassen's method for smaller exponents.

Note. LU decomposition is GPU-accelerable, but
(1) difficult to parallelize, and
(2) requires many memory access.

No inverse for DL anyways!

Example 1: Linear Regression



Q. Cost of training?

This really depends on “how” one optimizes the loss, i.e., how one solves the quadratic program

$$\min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$$

(1) Solve it analytically—we know linear algebra!

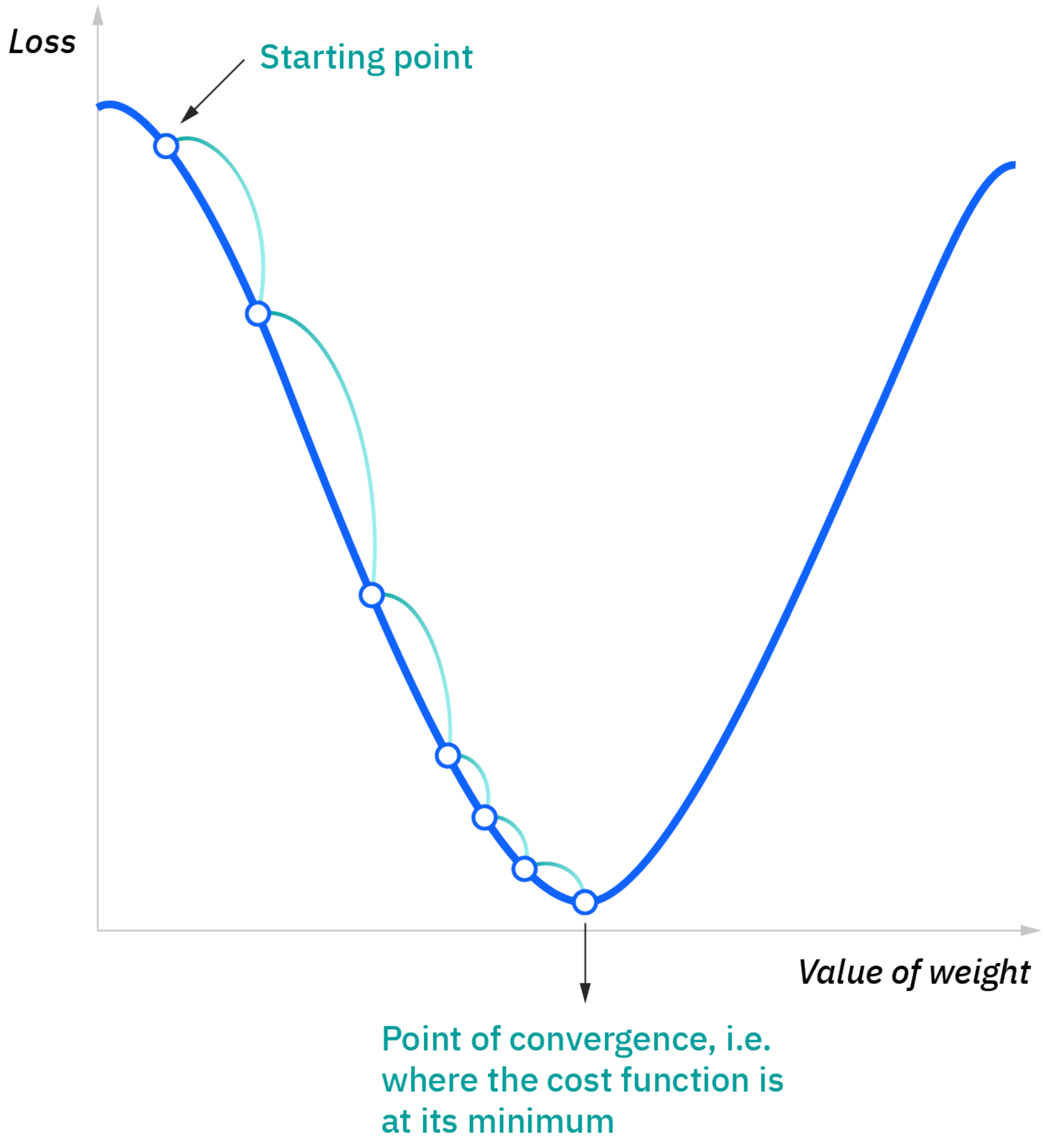
(2) Use indirect method—iterative optimization (e.g., gradient descent)

https://en.wikipedia.org/wiki/Linear_regression

Indirect method

Method. There are many indirect methods:
We consider gradient descent (for later uses).

Note. For this linear regression, better use
Jacobi's method / Gauss-Seidl
(more like coordinate descent)



https://www.ibm.com/cloud/learn/gradient-descent

Indirect method

Method. There are many indirect methods:
We consider gradient descent (for later uses).

GD. An iterative optimization method using

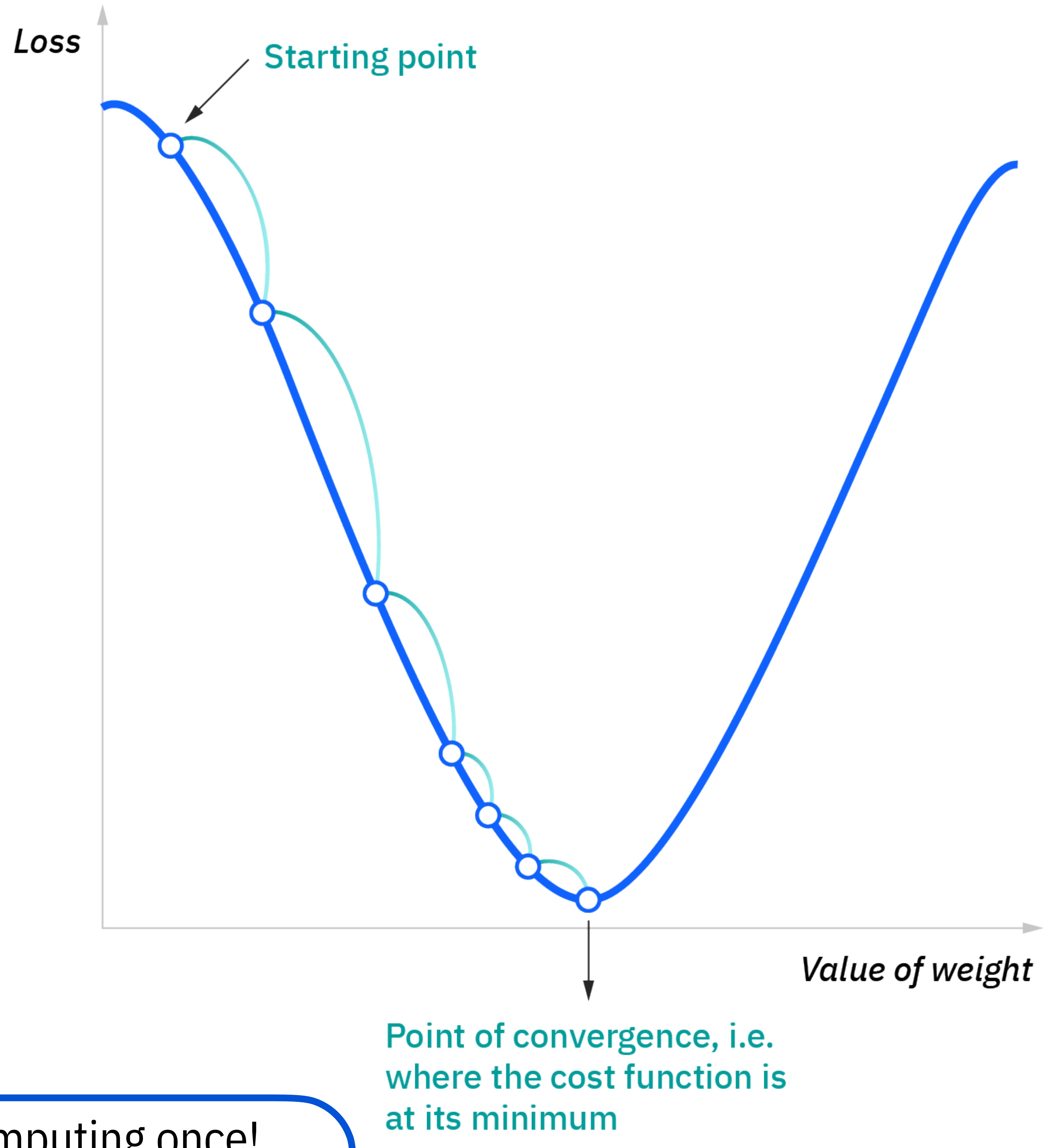
$$\boxed{\mathbf{w}^{\text{new}} = \mathbf{w} - \epsilon \cdot \nabla_{\mathbf{w}} R(\mathbf{w}; D)}$$

- De facto standard optimization method for deep learning.
- Not always convergent to optimum (but in this case, yes)
- Requires many steps, usually.

$$\mathbf{w}^{\text{new}} = \mathbf{w} - 2\epsilon \cdot (\mathbf{X}^\top \mathbf{X}\mathbf{w} - \mathbf{X}^\top \mathbf{y})$$

Option 1. Precompute $\mathbf{X}^\top \mathbf{X}$ and reuse.
Option 2. Compute $\mathbf{X}\mathbf{w}$ first
(better with less #steps).

Note. Only requires computing once!
Can be reused for every iteration.



https://www.ibm.com/cloud/learn/gradient-descent

$$\mathbf{w}^{\text{new}} = \mathbf{w} - 2\epsilon \cdot (\mathbf{X}^\top\mathbf{X}\mathbf{w} - \mathbf{X}^\top\mathbf{y})$$

Option 1. Precompute $\mathbf{X}^\top\mathbf{X}$ and reuse.
Option 2. Compute $\mathbf{X}\mathbf{w}$ first
(better with less #steps).

Option 1

Compute (Initial). $2d^2N + 2dN$

Compute (Repeat). $2d^2 + d$

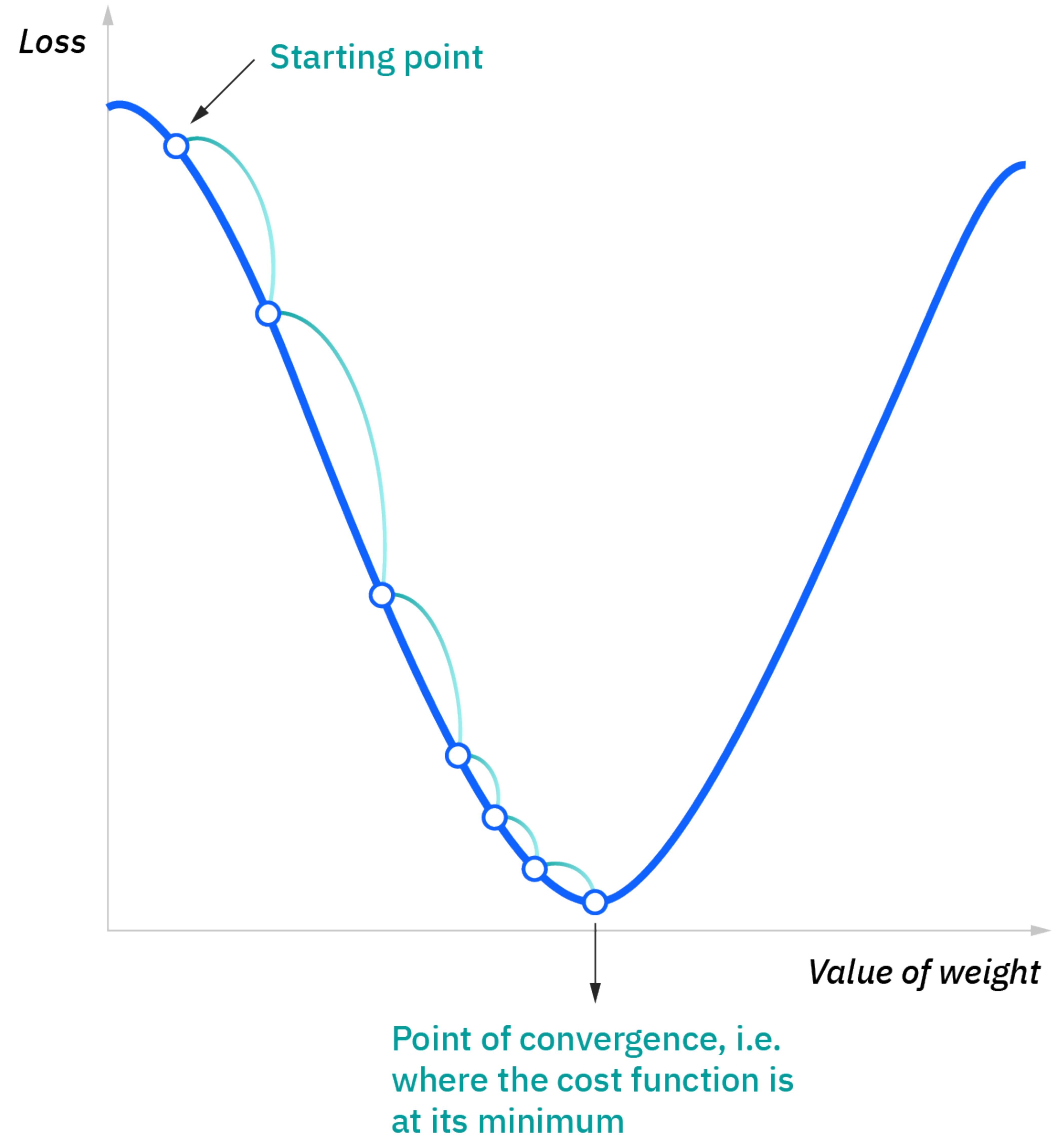
Memory. additional $\mathbb{R}^{d\times d} + \mathbb{R}^d$ throughout GD.
(but no more $\mathbf{X}$, saving $\mathbb{R}^{d\times N}$)

Option 2

Compute (Initial). $2dN$

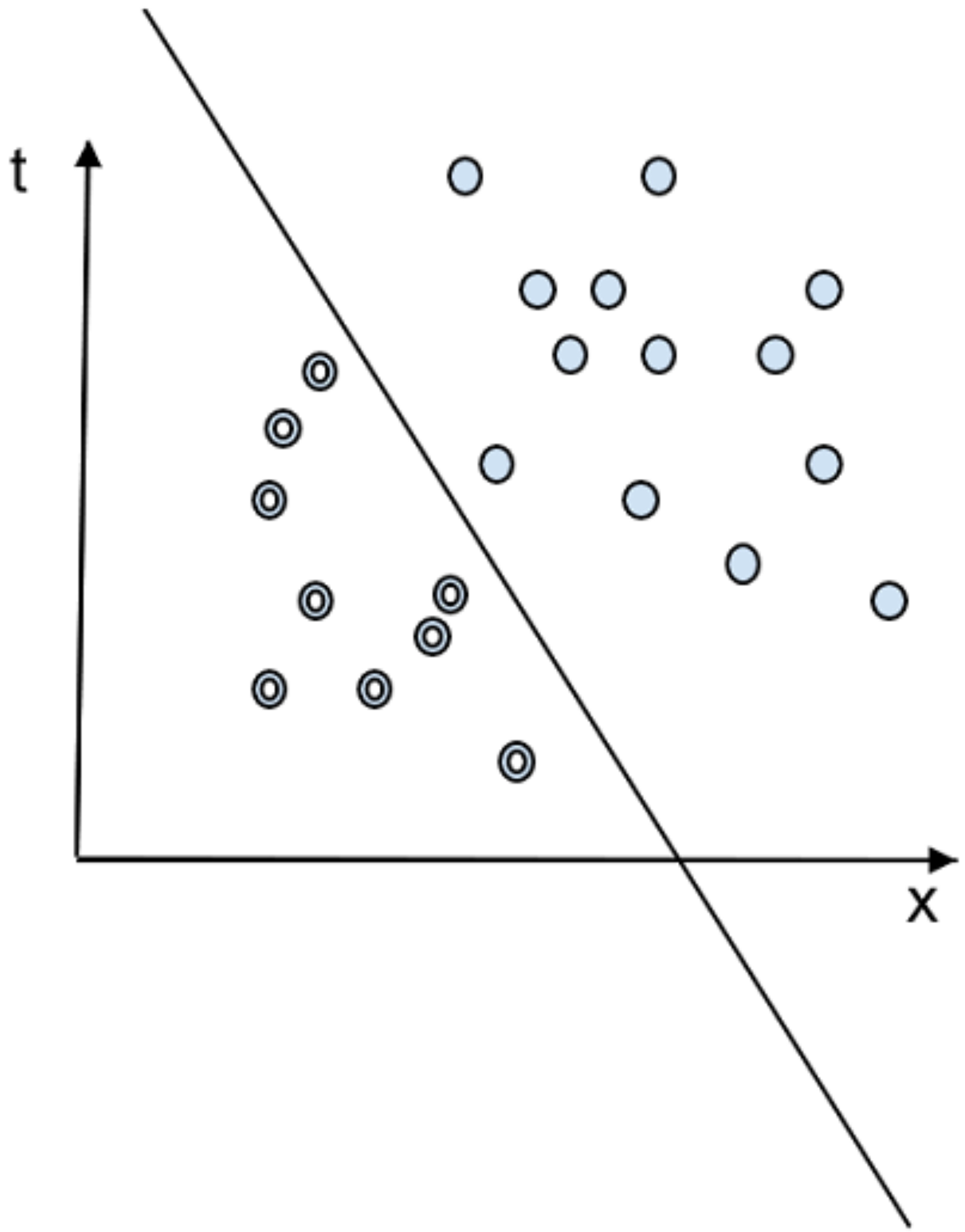
Compute (Repeat). $4dN + d$

Memory. additional $\mathbb{R}^d$ throughout GD



https://www.ibm.com/cloud/learn/gradient-descent

Example 2: Perceptron (online)



Dataset. $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$,
given the features $\mathbf{x}_i \in \mathbb{R}^d$
and the labels $y_i \in \{+1, -1\}$.

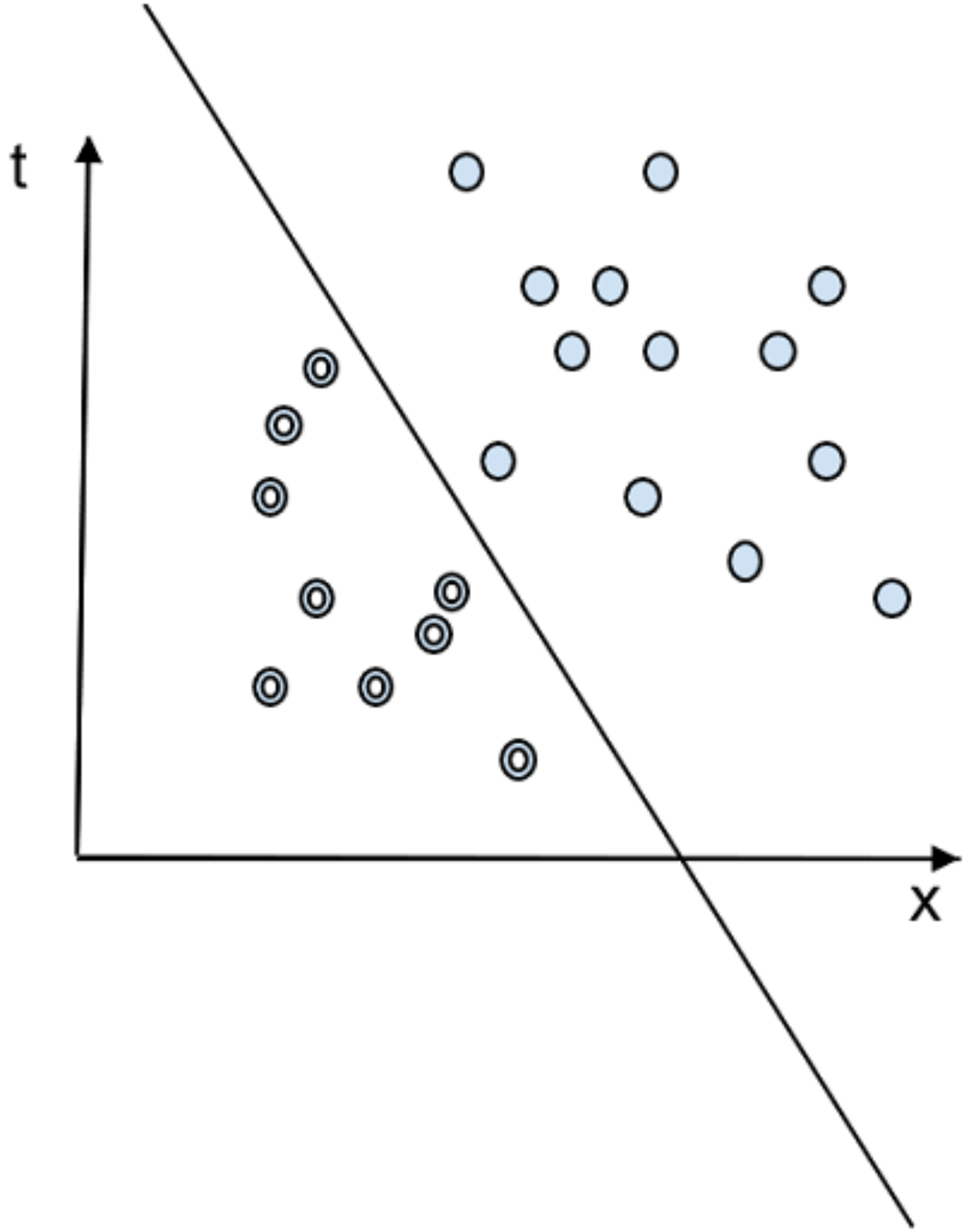
Note. Arrives one-by-one; no batch learning!

Model. $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$,
given the model params $\mathbf{w} \in \mathbb{R}^d$

Note. Actual classification is done with $\operatorname{sign}(\mathbf{w}^\top \mathbf{x})$

Note. Inference cost does not change—
Focus on training!

Example 2: Perceptron (online)



Loss. ReLU loss

$$\ell(\mathbf{w}, (\mathbf{x}, y)) = [-y \cdot \mathbf{w}^\top \mathbf{x}]_+$$

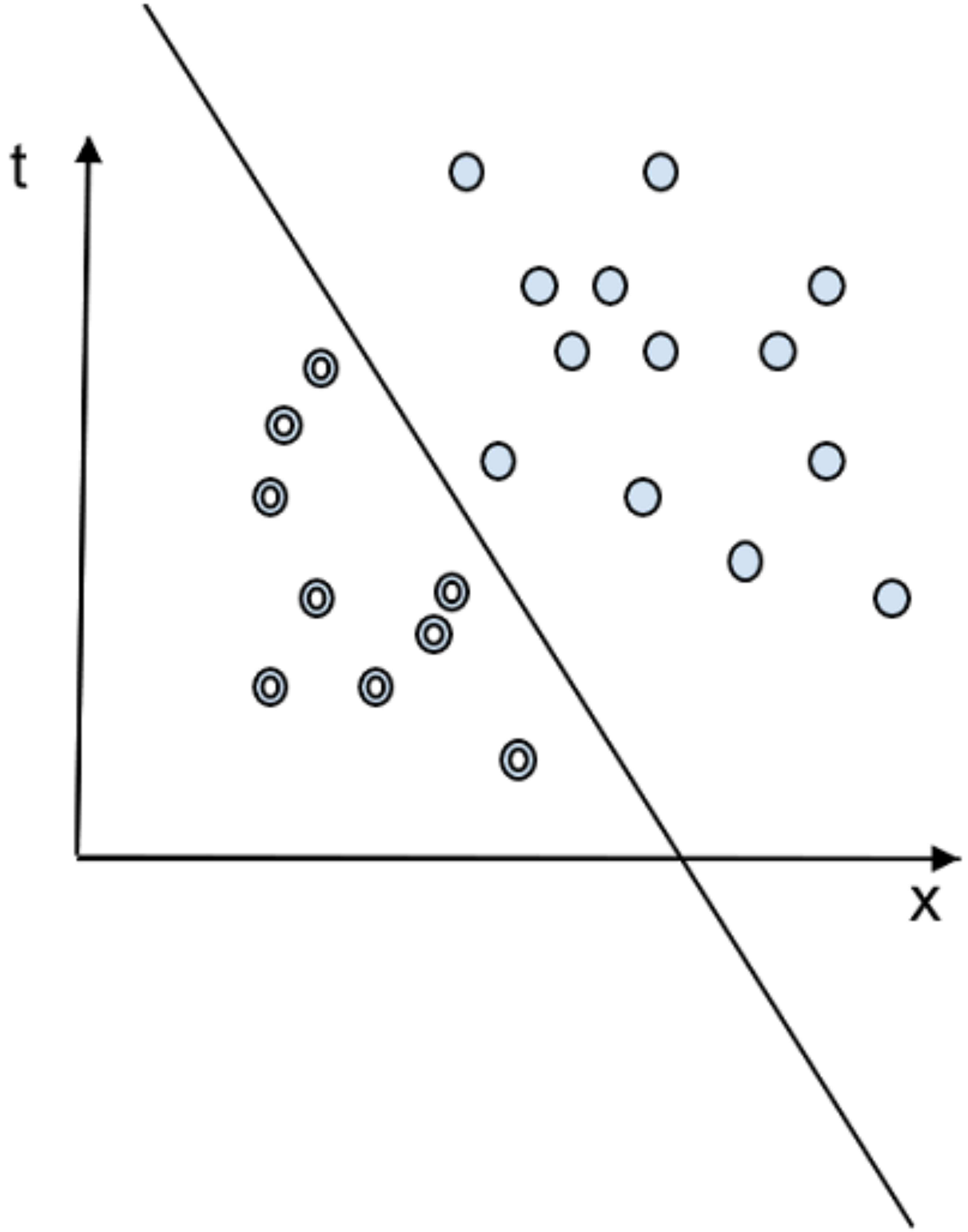
where $[\,\cdot\,]_+ = \begin{cases} 0 & \cdots \quad x < 0 \\ x & \cdots \quad x \geq 0 \end{cases}$

Note. If correct, no loss.
If wrong, penalize the "confidence."

Note. If $\mathbf{w} = \mathbf{0}$, no loss, and no training!
(how do we fix this?)

https://www.includehelp.com/python/perceptron-algorithm-and-its-implementation.aspx

Example 2: Perceptron (online)



Method. Mini-batch GD
with batch size 1, learning rate 1

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \nabla_{\mathbf{w}} \left[-y_i \cdot \mathbf{w}^{(i)\top}\mathbf{x}_i\right]_+$$
$$= \mathbf{w}^{(i)} + y_i \mathbf{x}_i \cdot \mathbf{1}\left[-y_i \mathbf{w}^{(i)\top}\mathbf{x}_i \geq 0\right]$$

Note. Having $\geq$ and not $>$ is critical!
(handles $\mathbf{w} = \mathbf{0}$)

Compute. $2d$ each step, total N steps.

Note. forced #iterations = #data in this case

https://www.includehelp.com/python/perceptron-algorithm-and-its-implementation.aspx

Wrapping up the examples

- Inference cost is deeply related to the model itself, but not about the optimization procedure
  e.g., same for all linear model

- Training cost depends heavily on "how to optimize"
  e.g., analytic vs indirect

- Training cost depends on small details…
  e.g., order of computation

- … and the optimal choice of such details depend on the hyperparameters we use
  e.g., the number of GD steps

- Elaborate methods can work faster, but may be difficult to parallelize / use with GPU.
  e.g., Strassen method for matrix multiplication

Hinted but not yet covered.

- "Memory vs. Compute" Tradeoff
  Scheduling & buffering

Coming next.

- Start talking about deep neural networks!