

# Deep Learning & Applied AI

Linear regression, convexity, and gradients

Emanuele Rodolà  
[rodola@di.uniroma1.it](mailto:rodola@di.uniroma1.it)



# A glimpse into neural networks

In deep learning, we deal with highly parametrized models called deep neural networks:



$$f_{\Theta}$$

# A glimpse into neural networks

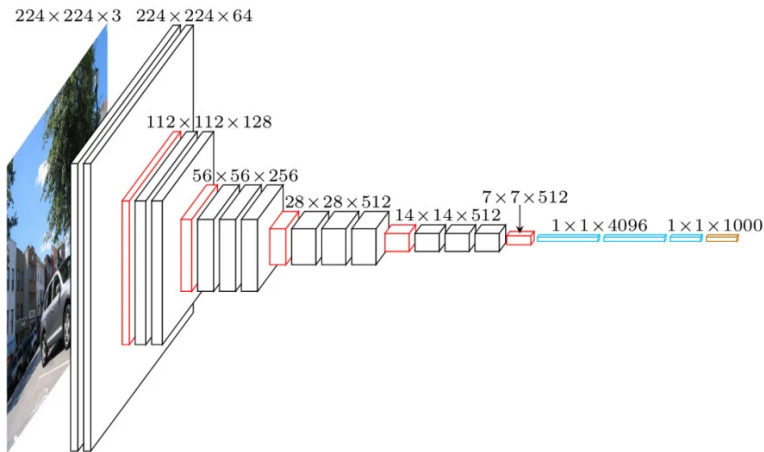
In deep learning, we deal with **highly parametrized models** called **deep neural networks**:



$$f_{\Theta}(\mathbf{x}) = \mathbf{y}$$

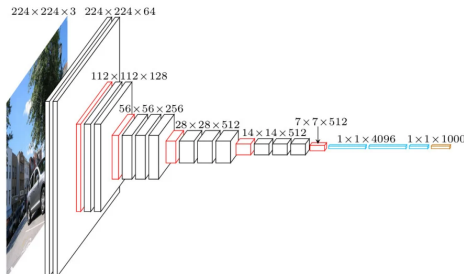
# A glimpse into neural networks

In deep learning, we deal with **highly parametrized models** called **deep neural networks**:



# A glimpse into neural networks

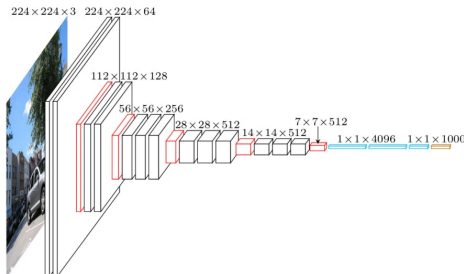
In deep learning, we deal with **highly parametrized models** called **deep neural networks**:



- Each block has a predefined structure (e.g., a **linear map**)

# A glimpse into neural networks

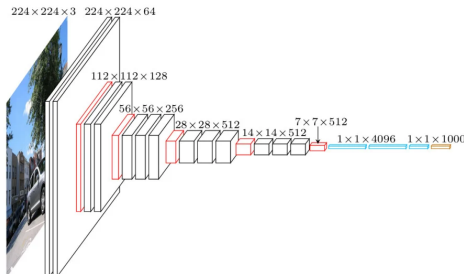
In deep learning, we deal with **highly parametrized models** called **deep neural networks**:



- Each block has a predefined structure (e.g., a **linear map**)
- Each block is defined in terms of **unknown parameters**  $\theta$

# A glimpse into neural networks

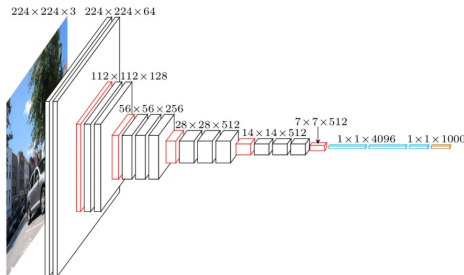
In deep learning, we deal with **highly parametrized models** called **deep neural networks**:



- Each block has a predefined structure (e.g., a **linear map**)
- Each block is defined in terms of **unknown parameters**  $\theta$
- Finding the parameter values is called **training**...

# A glimpse into neural networks

In deep learning, we deal with **highly parametrized models** called **deep neural networks**:

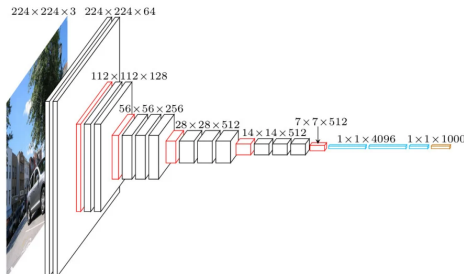


- Each block has a predefined structure (e.g., a **linear map**)
- Each block is defined in terms of **unknown parameters**  $\theta$
- Finding the parameter values is called **training**...
- ...which is done by minimizing a function called **loss**



# A glimpse into neural networks

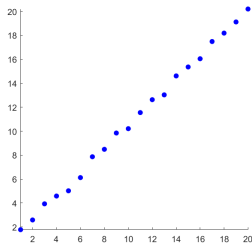
In deep learning, we deal with **highly parametrized models** called **deep neural networks**:



- Each block has a predefined structure (e.g., a **linear map**)
- Each block is defined in terms of **unknown parameters**  $\theta$
- Finding the parameter values is called **training**...
- ...which is done by minimizing a function called **loss**
- Minimization requires computing gradients, called **backpropagation**

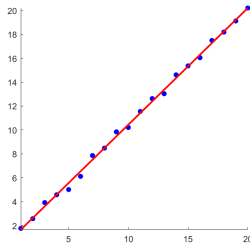
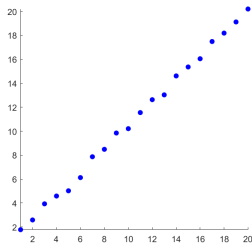
# Linear regression

We start from the simplest non-trivial case for a learning model:



# Linear regression

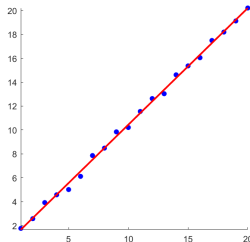
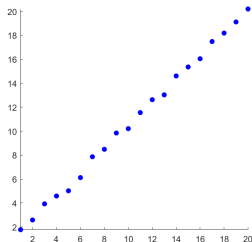
We start from the simplest non-trivial case for a learning model:



$$y_i = ax_i + b$$

# Linear regression

We start from the simplest non-trivial case for a learning model:



$$f_{\Theta}(x_i) = y_i$$

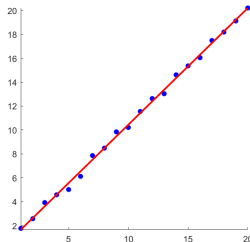
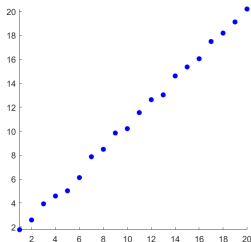
**Model:** linear + bias

**Parameters:**  $\Theta = \{a, b\}$

**Data:**  $n$  pairs  $(x_i, y_i)$ ; the  $x_i$  are called the **regressors**

# Linear regression

We start from the simplest non-trivial case for a learning model:



$$f_{\Theta}(x_i) = y_i$$

**Model:** linear + bias

**Parameters:**  $\Theta = \{a, b\}$

**Data:**  $n$  pairs  $(x_i, y_i)$ ; the  $x_i$  are called the **regressors**

Given  $a$  and  $b$ , we have a **mapping** that gives new output from new input.

# Linear regression

The equations:

$$f_{\Theta}(x_i) = y_i$$

must **approximately** hold for all  $i = 1, \dots, n$ .

# Linear regression

The equations:

$$f_{\Theta}(x_i) = y_i$$

must **approximately** hold for all  $i = 1, \dots, n$ .

**Problem:** Choose  $a$  and  $b$  that minimize the **mean squared error (MSE)** between input and predicted output:

$$\epsilon = \min_{a, b \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^n (y_i - f_{\Theta}(x_i))^2$$

where  $\Theta = \{a, b\}$ .

# Linear regression

The equations:

$$f_{\Theta}(x_i) = y_i$$

must **approximately** hold for all  $i = 1, \dots, n$ .

**Problem:** Choose  $a$  and  $b$  that minimize the **mean squared error (MSE)** between input and predicted output:

$$\epsilon = \min_{a, b \in \mathbb{R}} \sum_{i=1}^n (y_i - f_{\Theta}(x_i))^2$$

where  $\Theta = \{a, b\}$ .



# Linear regression

The equations:

$$f_{\Theta}(x_i) = y_i$$

must **approximately** hold for all  $i = 1, \dots, n$ .

**Problem:** Choose  $a$  and  $b$  that minimize the **mean squared error (MSE)** between input and predicted output:

$$\epsilon = \min_{a, b \in \mathbb{R}} \sum_{i=1}^n (y_i - f_{\Theta}(x_i))^2$$

where  $\Theta = \{a, b\}$ .

When  $f_{\Theta}$  is linear, this is called a **least-squares approximation** problem.

# Linear regression: Loss function

The equations:

$$f_{\Theta}(x_i) = y_i$$

must **approximately** hold for all  $i = 1, \dots, n$ .

**Problem:** Choose  $a$  and  $b$  that minimize the **mean squared error (MSE)** between input and predicted output:

$$\epsilon = \min_{\Theta} \ell_{\Theta}(\{x_i, y_i\})$$

The error criterion w.r.t. the parameters is also called a **loss** function, usually denoted by  $\ell$ :

$$\ell_{\Theta}(\{x_i, y_i\}) = \sum_{i=1}^n (y_i - f_{\Theta}(x_i))^2$$

# Linear regression: Loss function

The equations:

$$f_{\Theta}(x_i) = y_i$$

must **approximately** hold for all  $i = 1, \dots, n$ .

**Problem:** Choose  $a$  and  $b$  that minimize the **mean squared error (MSE)** between input and predicted output:

$$\epsilon = \min_{\Theta} \ell_{\Theta}(\{x_i, y_i\})$$

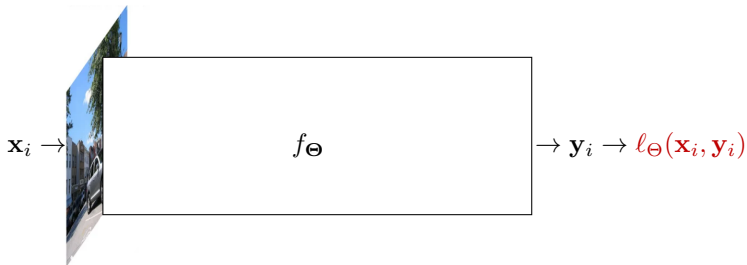
The error criterion w.r.t. the parameters is also called a **loss** function, usually denoted by  $\ell$ :

$$\ell_{\Theta}(\{x_i, y_i\}) = \sum_{i=1}^n (y_i - f_{\Theta}(x_i))^2$$

**Remark:** We minimize the loss **w.r.t. the parameters  $\Theta$** , and **not** w.r.t. the **data**  $(x_i, y_i)$ . Also, the loss is defined on the **entire dataset**, not on just one data point.

# Linear regression

We are considering the following case:



where  $f_{\Theta}$  is linear, and  $\ell_{\Theta}$  is quadratic.

# Optimization

We need to solve the general **minimization** problem:

$$\epsilon = \min_{\Theta} \ell(\Theta)$$

# Optimization

We need to solve the general **minimization** problem:

$$\epsilon = \min_{\Theta} \ell(\Theta)$$

In particular, we are interested in the **minimizer**  $\Theta$ .

# Optimization

We need to solve the general **minimization** problem:

$$\epsilon = \min_{\Theta} \ell(\Theta)$$

In particular, we are interested in the **minimizer**  $\Theta$ .

Finding minimizers for general  $\ell$  is an open problem. The research area is broadly called **optimization**.

In general, the optimization method depends on the properties of  $\ell$ .

# Optimization

We need to solve the general **minimization** problem:

$$\epsilon = \min_{\Theta} \ell(\Theta)$$

In particular, we are interested in the **minimizer**  $\Theta$ .

Finding minimizers for general  $\ell$  is an open problem. The research area is broadly called **optimization**.

In general, the optimization method depends on the properties of  $\ell$ .

We will mostly deal with **unconstrained** problems.



# Optimization

We need to solve the general **minimization** problem:

$$\epsilon = \min_{\Theta} \ell(\Theta)$$

In particular, we are interested in the **minimizer**  $\Theta$ .

Finding minimizers for general  $\ell$  is an open problem. The research area is broadly called **optimization**.

In general, the optimization method depends on the properties of  $\ell$ .

We will mostly deal with **unconstrained** problems.

Let's see what optimization problems we can solve **easily**!

# Convex functions

Jensen's inequality:

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y)$$

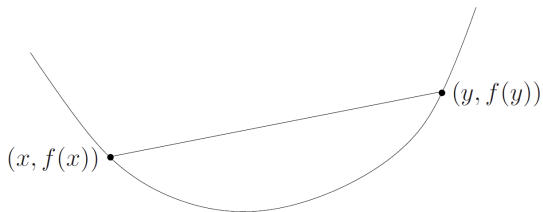
for all  $x, y$  and  $\alpha \in (0, 1)$

# Convex functions

Jensen's inequality:

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y)$$

for all  $x, y$  and  $\alpha \in (0, 1)$

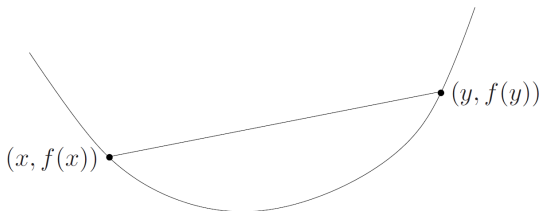


# Convex functions

Jensen's inequality:

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y)$$

for all  $x, y$  and  $\alpha \in (0, 1)$



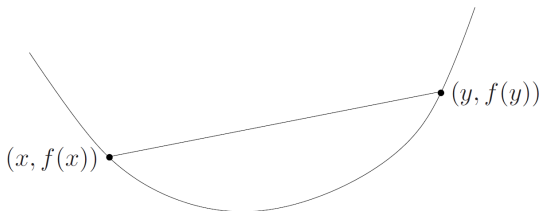
Let us further assume that  $f$  is a **differentiable** function, so that we can compute its **derivative**  $\frac{df}{dx}$  at all points  $x$ .

# Convex functions

Jensen's inequality:

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y)$$

for all  $x, y$  and  $\alpha \in (0, 1)$



Let us further assume that  $f$  is a **differentiable** function, so that we can compute its **derivative**  $\frac{df}{dx}$  at all points  $x$ .

**Theorem:** the **global** minimizer  $x$  is where  $\frac{df(x)}{dx} = 0$ .

## Convex functions on $\mathbb{R}^n$

In deep learning we deal with **loss functions** with  $n \gg 1$  parameters:

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

# Convex functions on $\mathbb{R}^n$

In deep learning we deal with **loss functions** with  $n \gg 1$  parameters:

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

The notion of derivative is replaced by the notion of **gradient**:

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

which is the vector of **partial derivatives** of  $f$ .

## Convex functions on $\mathbb{R}^n$

In deep learning we deal with **loss functions** with  $n \gg 1$  parameters:

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

The notion of derivative is replaced by the notion of **gradient**:

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

which is the vector of **partial derivatives** of  $f$ .

Convexity is defined as before:

$$f(\alpha \mathbf{x} + (1 - \alpha) \mathbf{y}) \leq \alpha f(\mathbf{x}) + (1 - \alpha) f(\mathbf{y})$$



# Convex functions on $\mathbb{R}^n$

In deep learning we deal with **loss functions** with  $n \gg 1$  parameters:

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

The notion of derivative is replaced by the notion of **gradient**:

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

which is the vector of **partial derivatives** of  $f$ .

Convexity is defined as before:

$$f(\alpha \mathbf{x} + (1 - \alpha) \mathbf{y}) \leq \alpha f(\mathbf{x}) + (1 - \alpha) f(\mathbf{y})$$

and we also have the **global optimality** condition:

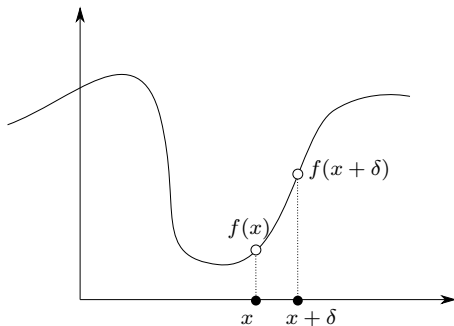
$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \mathbf{0} \implies f(\mathbf{x}) \leq f(\mathbf{y}) \text{ for all } \mathbf{y} \in \mathbb{R}^n$$

# The gradient

The gradient  $\nabla_{\mathbf{x}} f(\mathbf{x})$  encodes the **direction** of **steepest ascent** of  $f$  at point  $\mathbf{x}$ .

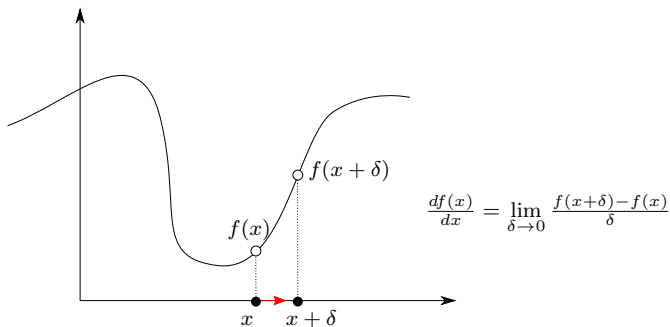
# The gradient

The gradient  $\nabla_{\mathbf{x}} f(\mathbf{x})$  encodes the **direction** of **steepest ascent** of  $f$  at point  $\mathbf{x}$ . In the simple 1D case:



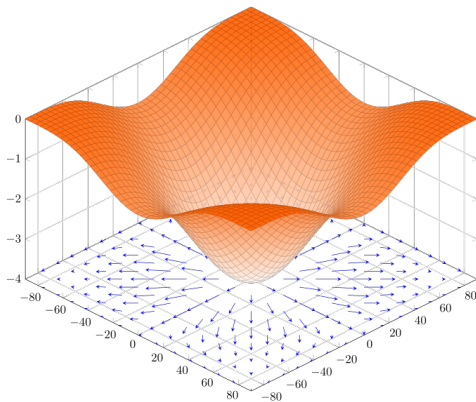
# The gradient

The gradient  $\nabla_{\mathbf{x}} f(\mathbf{x})$  encodes the **direction** of **steepest ascent** of  $f$  at point  $\mathbf{x}$ . In the simple 1D case:



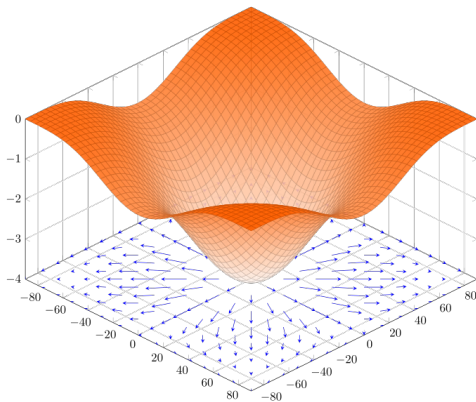
# The gradient

The gradient  $\nabla_{\mathbf{x}}f(\mathbf{x})$  encodes the **direction** of **steepest ascent** of  $f$  at point  $\mathbf{x}$ . In the more general case:



# The gradient

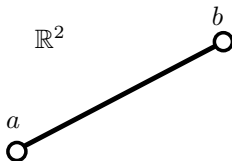
The gradient  $\nabla_{\mathbf{x}} f(\mathbf{x})$  encodes the **direction** of **steepest ascent** of  $f$  at point  $\mathbf{x}$ . In the more general case:



The **length** of the gradient vector encodes its steepness.

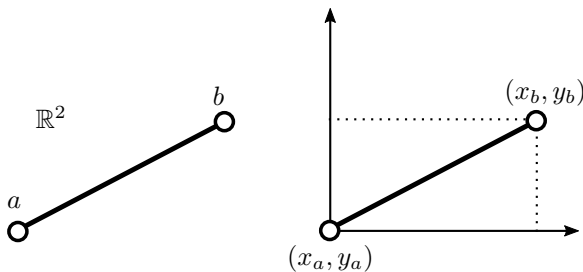
# Vector lengths

The **Euclidean distance** measures the length of a straight line connecting two points:



# Vector lengths

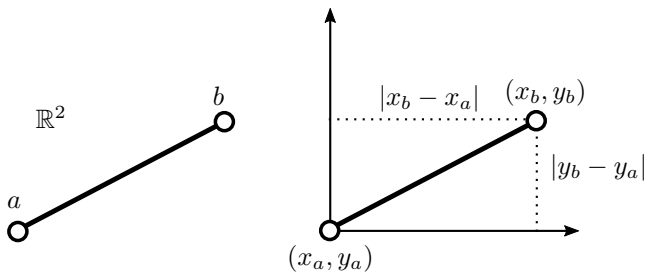
The **Euclidean distance** measures the length of a straight line connecting two points:





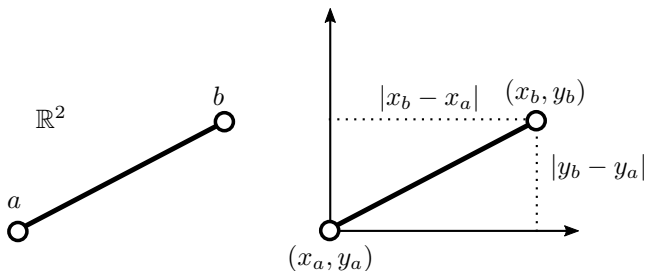
# Vector lengths

The **Euclidean distance** measures the length of a straight line connecting two points:



# Vector lengths

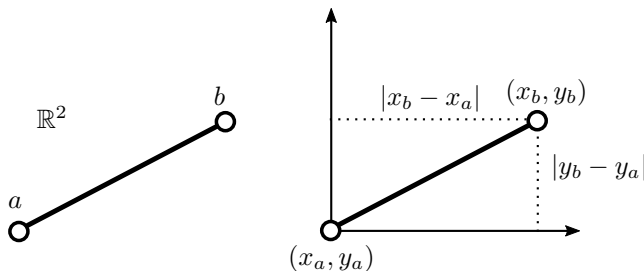
The **Euclidean distance** measures the length of a straight line connecting two points:



Apply Pythagoras' theorem:  $d(a, b) = (|x_b - x_a|^2 + |y_b - y_a|^2)^{\frac{1}{2}}$

# Vector lengths

The **Euclidean distance** measures the length of a straight line connecting two points:



Apply Pythagoras' theorem:  $d(a, b) = (|x_b - x_a|^2 + |y_b - y_a|^2)^{\frac{1}{2}}$

In matrix notation:

$$d(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\|_2$$

where  $\mathbf{a} = \begin{pmatrix} x_a \\ y_a \end{pmatrix}$  and  $\mathbf{b} = \begin{pmatrix} x_b \\ y_b \end{pmatrix}$

## $L_p$ distance in $\mathbb{R}^k$

One can generalize to different power coefficients  $p \geq 1$ :

$$\begin{aligned}\|\mathbf{x} - \mathbf{y}\|_2 &= (|x_1 - y_1|^2 + |x_2 - y_2|^2)^{\frac{1}{2}} \\ &\Downarrow \\ \|\mathbf{x} - \mathbf{y}\|_{\textcolor{red}{p}} &= (|x_1 - y_1|^{\textcolor{red}{p}} + |x_2 - y_2|^{\textcolor{red}{p}})^{\frac{1}{\textcolor{red}{p}}}\end{aligned}$$

## $L_p$ distance in $\mathbb{R}^k$

One can generalize to different power coefficients  $p \geq 1$ :

$$\begin{aligned}\|\mathbf{x} - \mathbf{y}\|_2 &= (|x_1 - y_1|^2 + |x_2 - y_2|^2)^{\frac{1}{2}} \\ &\quad \Downarrow \\ \|\mathbf{x} - \mathbf{y}\|_{\textcolor{red}{p}} &= (|x_1 - y_1|^{\textcolor{red}{p}} + |x_2 - y_2|^{\textcolor{red}{p}})^{\frac{1}{\textcolor{red}{p}}}\end{aligned}$$

As well as generalize from  $\mathbb{R}^2$  to  $\mathbb{R}^k$ :

$$\|\mathbf{x} - \mathbf{y}\|_p = \left( \sum_{i=1}^k |x_i - y_i|^p \right)^{\frac{1}{p}}$$

## $L_p$ distance in $\mathbb{R}^k$

One can generalize to different power coefficients  $p \geq 1$ :

$$\begin{aligned}\|\mathbf{x} - \mathbf{y}\|_2 &= (|x_1 - y_1|^2 + |x_2 - y_2|^2)^{\frac{1}{2}} \\ &\quad \Downarrow \\ \|\mathbf{x} - \mathbf{y}\|_{\textcolor{red}{p}} &= (|x_1 - y_1|^{\textcolor{red}{p}} + |x_2 - y_2|^{\textcolor{red}{p}})^{\frac{1}{\textcolor{red}{p}}}\end{aligned}$$

As well as generalize from  $\mathbb{R}^2$  to  $\mathbb{R}^k$ :

$$\|\mathbf{x} - \mathbf{y}\|_p = \left( \sum_{i=1}^k |x_i - y_i|^p \right)^{\frac{1}{p}}$$

This definition gives us the  $L_p$  distance between vectors in  $\mathbb{R}^k$ .

## $L_p$ distance in $\mathbb{R}^k$

One can generalize to different power coefficients  $p \geq 1$ :

$$\begin{aligned}\|\mathbf{x} - \mathbf{y}\|_2 &= (|x_1 - y_1|^2 + |x_2 - y_2|^2)^{\frac{1}{2}} \\ &\Downarrow \\ \|\mathbf{x} - \mathbf{y}\|_{\textcolor{red}{p}} &= (|x_1 - y_1|^{\textcolor{red}{p}} + |x_2 - y_2|^{\textcolor{red}{p}})^{\frac{1}{\textcolor{red}{p}}}\end{aligned}$$

As well as generalize from  $\mathbb{R}^2$  to  $\mathbb{R}^k$ :

$$\|\mathbf{x} - \mathbf{y}\|_p = (\sum_{i=1}^k |x_i - y_i|^p)^{\frac{1}{p}}$$

This definition gives us the  $L_p$  distance between vectors in  $\mathbb{R}^k$ .

The  $L_p$  distance (or norm) of a vector is simply its distance from the origin:

$$\|\mathbf{x} - \mathbf{0}\|_2 = \|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^k |x_i|^2}$$

## $L_p$ distance in $\mathbb{R}^k$

One can generalize to different power coefficients  $p \geq 1$ :

$$\begin{aligned}\|\mathbf{x} - \mathbf{y}\|_2 &= (|x_1 - y_1|^2 + |x_2 - y_2|^2)^{\frac{1}{2}} \\ &\Downarrow \\ \|\mathbf{x} - \mathbf{y}\|_{\textcolor{red}{p}} &= (|x_1 - y_1|^{\textcolor{red}{p}} + |x_2 - y_2|^{\textcolor{red}{p}})^{\frac{1}{\textcolor{red}{p}}}\end{aligned}$$

As well as generalize from  $\mathbb{R}^2$  to  $\mathbb{R}^k$ :

$$\|\mathbf{x} - \mathbf{y}\|_p = (\sum_{i=1}^k |x_i - y_i|^p)^{\frac{1}{p}}$$

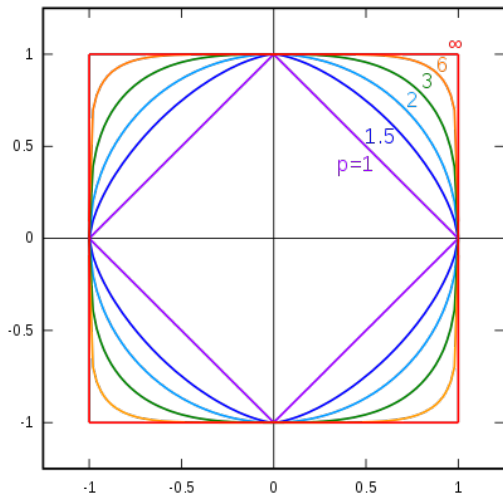
This definition gives us the  $L_p$  distance between vectors in  $\mathbb{R}^k$ .

The  $L_2$  (or norm) of a vector is simply its distance from the origin:

$$\|\mathbf{x} - \mathbf{0}\|_2 = \|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^k |x_i|^2} = \sqrt{\mathbf{x}^\top \mathbf{x}}$$



$L_p$  unit balls in  $\mathbb{R}^2$



## Linear regression: Finding a solution

$$\min_{a,b \in \mathbb{R}} \sum_{i=1}^n (y_i - ax_i - b)^2$$

# Linear regression: Finding a solution

$$\Theta^* = \arg \min_{\Theta \in \mathbb{R}^2} \ell(\Theta)$$

where  $\ell : \mathbb{R}^2 \rightarrow \mathbb{R}$  is defined as:

$$\ell(a, b) = \sum_{i=1}^n (y_i - ax_i - b)^2$$

# Linear regression: Finding a solution

$$\Theta^* = \arg \min_{\Theta \in \mathbb{R}^2} \ell(\Theta)$$

where  $\ell : \mathbb{R}^2 \rightarrow \mathbb{R}$  is defined as:

$$\ell(a, b) = \sum_{i=1}^n (y_i - ax_i - b)^2$$

A solution is found by setting  $\nabla_{\Theta} \ell(\Theta) = \mathbf{0}$ :

$$\nabla_{\Theta} \sum_{i=1}^n (y_i - ax_i - b)^2 = \sum_{i=1}^n \nabla_{\Theta} (y_i - ax_i - b)^2$$

# Linear regression: Finding a solution

$$\Theta^* = \arg \min_{\Theta \in \mathbb{R}^2} \ell(\Theta)$$

where  $\ell : \mathbb{R}^2 \rightarrow \mathbb{R}$  is defined as:

$$\ell(a, b) = \sum_{i=1}^n (y_i - ax_i - b)^2$$

A solution is found by setting  $\nabla_{\Theta} \ell(\Theta) = \mathbf{0}$ :

$$\begin{aligned} \nabla_{\Theta} \sum_{i=1}^n (y_i - ax_i - b)^2 &= \sum_{i=1}^n \nabla_{\Theta} (y_i - ax_i - b)^2 \\ &= \sum_{i=1}^n \nabla_{\Theta} (y_i^2 + a^2 x_i^2 + b^2 - 2ax_i y_i - 2by_i + 2abx_i) \end{aligned}$$

# Linear regression: Finding a solution

$$\Theta^* = \arg \min_{\Theta \in \mathbb{R}^2} \ell(\Theta)$$

where  $\ell : \mathbb{R}^2 \rightarrow \mathbb{R}$  is defined as:

$$\ell(a, b) = \sum_{i=1}^n (y_i - ax_i - b)^2$$

A solution is found by setting  $\nabla_{\Theta} \ell(\Theta) = \mathbf{0}$ :

$$\begin{aligned} \nabla_{\Theta} \sum_{i=1}^n (y_i - ax_i - b)^2 &= \sum_{i=1}^n \nabla_{\Theta} (y_i - ax_i - b)^2 \\ &= \sum_{i=1}^n \nabla_{\Theta} (y_i^2 + a^2 x_i^2 + b^2 - 2ax_i y_i - 2by_i + 2abx_i) \\ &= \sum_{i=1}^n \begin{pmatrix} 2ax_i^2 - 2x_i y_i + 2bx_i \\ 2b - 2y_i + 2ax_i \end{pmatrix} \end{aligned}$$

# Linear regression: Finding a solution

$$\Theta^* = \arg \min_{\Theta \in \mathbb{R}^2} \ell(\Theta)$$

where  $\ell : \mathbb{R}^2 \rightarrow \mathbb{R}$  is defined as:

$$\ell(a, b) = \sum_{i=1}^n (y_i - ax_i - b)^2$$

A solution is found by setting  $\nabla_{\Theta} \ell(\Theta) = \mathbf{0}$ :

$$\begin{aligned} \nabla_{\Theta} \sum_{i=1}^n (y_i - ax_i - b)^2 &= \sum_{i=1}^n \nabla_{\Theta} (y_i - ax_i - b)^2 \\ &= \sum_{i=1}^n \nabla_{\Theta} (y_i^2 + a^2 x_i^2 + b^2 - 2ax_i y_i - 2by_i + 2abx_i) \\ &= \begin{pmatrix} \sum_{i=1}^n 2ax_i^2 - 2x_i y_i + 2bx_i \\ \sum_{i=1}^n 2b - 2y_i + 2ax_i \end{pmatrix} \end{aligned}$$

# Linear regression: Finding a solution

$$\Theta^* = \arg \min_{\Theta \in \mathbb{R}^2} \ell(\Theta)$$

where  $\ell : \mathbb{R}^2 \rightarrow \mathbb{R}$  is defined as:

$$\ell(a, b) = \sum_{i=1}^n (y_i - ax_i - b)^2$$

A solution is found by setting  $\nabla_{\Theta} \ell(\Theta) = \mathbf{0}$ :

$$\nabla_{\Theta} \sum_{i=1}^n (y_i - ax_i - b)^2 = \begin{pmatrix} \sum_{i=1}^n 2ax_i^2 - 2x_iy_i + 2bx_i \\ \sum_{i=1}^n 2b - 2y_i + 2ax_i \end{pmatrix}$$

We get 2 linear equations in the 2 unknowns  $a, b$ :

$$\begin{pmatrix} \sum_{i=1}^n ax_i^2 + bx_i - x_iy_i \\ \sum_{i=1}^n ax_i + b - y_i \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$



# Linear regression: Matrix notation

The learning model of linear regression is **linear in the parameters** (while it is **not** linear in  $x$ , due to the bias).

Therefore, in matrix notation the equations  $y_i = ax_i + b$  read:

$$\underbrace{\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}}_{\mathbf{y}} = \underbrace{\begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix}}_{\mathbf{X}} \underbrace{\begin{pmatrix} a \\ b \end{pmatrix}}_{\boldsymbol{\theta}}$$

# Linear regression: Matrix notation

The learning model of linear regression is **linear in the parameters** (while it is **not** linear in  $x$ , due to the bias).

Therefore, in matrix notation the equations  $y_i = ax_i + b$  read:

$$\underbrace{\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}}_{\mathbf{y}} = \underbrace{\begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix}}_{\mathbf{X}} \underbrace{\begin{pmatrix} a \\ b \end{pmatrix}}_{\boldsymbol{\theta}}$$

**Remark:** Deep learning frameworks frequently use the alternative expression with the bias encoded separately:

$$\underbrace{\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}}_{\mathbf{y}} = a \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}}_{\mathbf{X}} + b$$

## Linear regression: Matrix notation

Familiarize with matrix calculus.

When implementing deep nets, we manipulate matrices, vectors, and tensors.

## Linear regression: Matrix notation

$$\underbrace{\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}}_{\mathbf{y}} = \underbrace{\begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix}}_{\mathbf{X}} \underbrace{\begin{pmatrix} a \\ b \end{pmatrix}}_{\boldsymbol{\theta}}$$

This expresses all the equations  $y_i = ax_i + b$  at once and makes the linearity w.r.t.  $a, b$  evident.

The MSE is simply:

$$\ell(\boldsymbol{\theta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|_2^2$$

## Linear regression: Matrix notation

$$\underbrace{\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}}_{\mathbf{y}} = \underbrace{\begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix}}_{\mathbf{X}} \underbrace{\begin{pmatrix} a \\ b \end{pmatrix}}_{\boldsymbol{\theta}}$$

This expresses all the equations  $y_i = ax_i + b$  at once and makes the linearity w.r.t.  $a, b$  evident.

The MSE is simply:

$$\ell(\boldsymbol{\theta}) = (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})$$

## Linear regression: Matrix notation

$$\underbrace{\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}}_{\mathbf{y}} = \underbrace{\begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix}}_{\mathbf{X}} \underbrace{\begin{pmatrix} a \\ b \end{pmatrix}}_{\boldsymbol{\theta}}$$

This expresses all the equations  $y_i = ax_i + b$  at once and makes the linearity w.r.t.  $a, b$  evident.

The MSE is simply:

$$\ell(\boldsymbol{\theta}) = \mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^\top \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\theta}$$

## Linear regression: Matrix notation

$$\underbrace{\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}}_{\mathbf{y}} = \underbrace{\begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix}}_{\mathbf{X}} \underbrace{\begin{pmatrix} a \\ b \end{pmatrix}}_{\boldsymbol{\theta}}$$

This expresses all the equations  $y_i = ax_i + b$  at once and makes the linearity w.r.t.  $a, b$  evident.

The MSE is simply:

$$\ell(\boldsymbol{\theta}) = \mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^\top \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\theta}$$

Setting  $\nabla_{\boldsymbol{\theta}} \ell = \mathbf{0}$  we get:

$$-2\mathbf{X}^\top \mathbf{y} + 2\mathbf{X}^\top \mathbf{X}\boldsymbol{\theta} = \mathbf{0}$$

## Linear regression: Matrix notation

$$\underbrace{\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}}_{\mathbf{y}} = \underbrace{\begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix}}_{\mathbf{X}} \underbrace{\begin{pmatrix} a \\ b \end{pmatrix}}_{\boldsymbol{\theta}}$$

This expresses all the equations  $y_i = ax_i + b$  at once and makes the linearity w.r.t.  $a, b$  evident.

The MSE is simply:

$$\ell(\boldsymbol{\theta}) = \mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^\top \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\theta}$$

Setting  $\nabla_{\boldsymbol{\theta}} \ell = \mathbf{0}$  we get:

$$\mathbf{X}^\top \mathbf{X}\boldsymbol{\theta} = \mathbf{X}^\top \mathbf{y}$$



## Linear regression: Matrix notation

$$\underbrace{\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}}_{\mathbf{y}} = \underbrace{\begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix}}_{\mathbf{X}} \underbrace{\begin{pmatrix} a \\ b \end{pmatrix}}_{\boldsymbol{\theta}}$$

This expresses all the equations  $y_i = ax_i + b$  at once and makes the linearity w.r.t.  $a, b$  evident.

The MSE is simply:

$$\ell(\boldsymbol{\theta}) = \mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^\top \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\theta}$$

Setting  $\nabla_{\boldsymbol{\theta}} \ell = \mathbf{0}$  we get:

$$\boldsymbol{\theta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

We get a **closed form solution** to our problem.

# A note on the gradient in matrix form

In the previous slide, for the differentiation step:

$$\mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^\top \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\theta} \xrightarrow{\nabla_{\boldsymbol{\theta}}} -2\mathbf{X}^\top \mathbf{y} + 2\mathbf{X}^\top \mathbf{X}\boldsymbol{\theta}$$

what we did is **exactly equivalent** to the element-by-element computation of slide #13/20, but we did it directly in matrix form.

# A note on the gradient in matrix form

In the previous slide, for the differentiation step:

$$\mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^\top \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\theta} \xrightarrow{\nabla_{\boldsymbol{\theta}}} -2\mathbf{X}^\top \mathbf{y} + 2\mathbf{X}^\top \mathbf{X}\boldsymbol{\theta}$$

what we did is **exactly equivalent** to the element-by-element computation of slide #13/20, but we did it directly in matrix form.

Example:  $f(\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{A}\boldsymbol{\theta}$

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} (\theta_1 \quad \cdots \quad \theta_n) \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} \theta_1 \\ \cdots \\ \theta_n \end{pmatrix}$$

# A note on the gradient in matrix form

In the previous slide, for the differentiation step:

$$\mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^\top \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\theta} \xrightarrow{\nabla_{\boldsymbol{\theta}}} -2\mathbf{X}^\top \mathbf{y} + 2\mathbf{X}^\top \mathbf{X}\boldsymbol{\theta}$$

what we did is **exactly equivalent** to the element-by-element computation of slide #13/20, but we did it directly in matrix form.

Example:  $f(\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{A}\boldsymbol{\theta}$

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \sum_{i=1}^n \sum_{j=1}^n a_{ij} \theta_i \theta_j$$

# A note on the gradient in matrix form

In the previous slide, for the differentiation step:

$$\mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^\top \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\theta} \xrightarrow{\nabla_{\boldsymbol{\theta}}} -2\mathbf{X}^\top \mathbf{y} + 2\mathbf{X}^\top \mathbf{X}\boldsymbol{\theta}$$

what we did is **exactly equivalent** to the element-by-element computation of slide #13/20, but we did it directly in matrix form.

Example:  $f(\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{A}\boldsymbol{\theta}$

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_1} \sum_{i=1}^n \sum_{j=1}^n a_{ij} \theta_i \theta_j \\ \vdots \\ \frac{\partial}{\partial \theta_n} \sum_{i=1}^n \sum_{j=1}^n a_{ij} \theta_i \theta_j \end{pmatrix}$$

# A note on the gradient in matrix form

In the previous slide, for the differentiation step:

$$\mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^\top \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\theta} \xrightarrow{\nabla_{\boldsymbol{\theta}}} -2\mathbf{X}^\top \mathbf{y} + 2\mathbf{X}^\top \mathbf{X}\boldsymbol{\theta}$$

what we did is **exactly equivalent** to the element-by-element computation of slide #13/20, but we did it directly in matrix form.

Example:  $f(\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{A}\boldsymbol{\theta}$

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \begin{pmatrix} \sum_j a_{1j}\theta_j + \sum_i a_{i1}\theta_i \\ \vdots \\ \sum_j a_{nj}\theta_j + \sum_i a_{in}\theta_i \end{pmatrix}$$

# A note on the gradient in matrix form

In the previous slide, for the differentiation step:

$$\mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^\top \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\theta} \xrightarrow{\nabla_{\boldsymbol{\theta}}} -2\mathbf{X}^\top \mathbf{y} + 2\mathbf{X}^\top \mathbf{X}\boldsymbol{\theta}$$

what we did is **exactly equivalent** to the element-by-element computation of slide #13/20, but we did it directly in matrix form.

Example:  $f(\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{A}\boldsymbol{\theta}$

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \begin{pmatrix} \sum_i (a_{1i} + a_{i1})\theta_i \\ \vdots \\ \sum_i (a_{ni} + a_{in})\theta_i \end{pmatrix}$$

# A note on the gradient in matrix form

In the previous slide, for the differentiation step:

$$\mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^\top \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\theta} \xrightarrow{\nabla_{\boldsymbol{\theta}}} -2\mathbf{X}^\top \mathbf{y} + 2\mathbf{X}^\top \mathbf{X}\boldsymbol{\theta}$$

what we did is **exactly equivalent** to the element-by-element computation of slide #13/20, but we did it directly in matrix form.

Example:  $f(\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{A}\boldsymbol{\theta}$

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = (\mathbf{A} + \mathbf{A}^\top)\boldsymbol{\theta}$$



# A note on the gradient in matrix form

In the previous slide, for the differentiation step:

$$\mathbf{y}^\top \mathbf{y} - 2\mathbf{y}^\top \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\theta} \xrightarrow{\nabla_{\boldsymbol{\theta}}} -2\mathbf{X}^\top \mathbf{y} + 2\mathbf{X}^\top \mathbf{X}\boldsymbol{\theta}$$

what we did is **exactly equivalent** to the element-by-element computation of slide #13/20, but we did it directly in matrix form.

Example:  $f(\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{A}\boldsymbol{\theta}$

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = (\mathbf{A} + \mathbf{A}^\top)\boldsymbol{\theta}$$

If  $\mathbf{A}$  is symmetric (e.g.,  $\mathbf{A} = \mathbf{X}^\top \mathbf{X}$ ), then:

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = 2\mathbf{A}\boldsymbol{\theta}$$

# Linear regression: Higher dimensions

Until now we have seen the case where:

$$y_i = ax_i + b \quad \text{for } i = 1, \dots, n$$

that is, each data point is one-dimensional (just one number).

# Linear regression: Higher dimensions

Until now we have seen the case where:

$$y_i = ax_i + b \quad \text{for } i = 1, \dots, n$$

that is, each data point is one-dimensional (just one number).

In the more general case, the data points  $(\mathbf{x}_i, \mathbf{y}_i)$  are vectors in  $\mathbb{R}^d$ :

$$\mathbf{y}_i = \mathbf{A}\mathbf{x}_i + \mathbf{b} \quad \text{for } i = 1, \dots, n$$

# Linear regression: Higher dimensions

Until now we have seen the case where:

$$y_i = ax_i + b \quad \text{for } i = 1, \dots, n$$

that is, each data point is one-dimensional (just one number).

In the more general case, the data points  $(\mathbf{x}_i, \mathbf{y}_i)$  are vectors in  $\mathbb{R}^d$ :

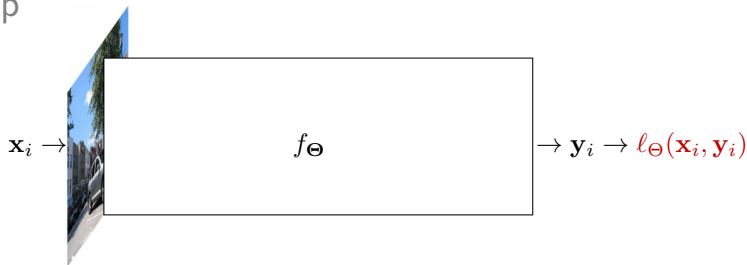
$$\mathbf{y}_i = \mathbf{A}\mathbf{x}_i + \mathbf{b} \quad \text{for } i = 1, \dots, n$$

Defining the matrices  $\mathbf{X} = \begin{pmatrix} | & | & \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots \\ | & | & \\ 1 & 1 & \end{pmatrix}$ ,  $\mathbf{Y} = \begin{pmatrix} | & | & \\ \mathbf{y}_1 & \mathbf{y}_2 & \cdots \\ | & | & \end{pmatrix}$ ,  $\mathbf{\Theta} = \begin{pmatrix} \mathbf{A} \\ \mathbf{b}^\top \end{pmatrix}$ ,

we get a closed-form solution to  $\nabla_{\mathbf{\Theta}} \ell(\mathbf{\Theta}) = \mathbf{0}$ :

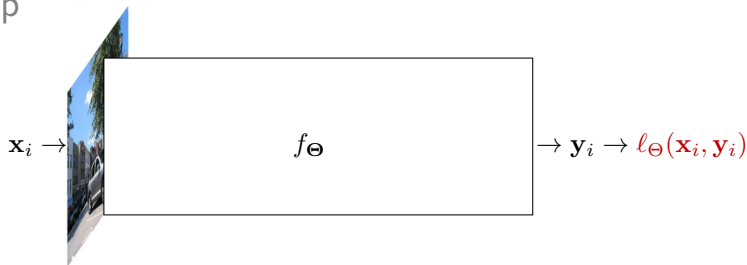
$$\mathbf{\Theta} = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{Y}^\top$$

## Wrap-up



Sometimes, the learning model is **linear** and the loss is **quadratic**.  
This case can be solved in closed form.

## Wrap-up

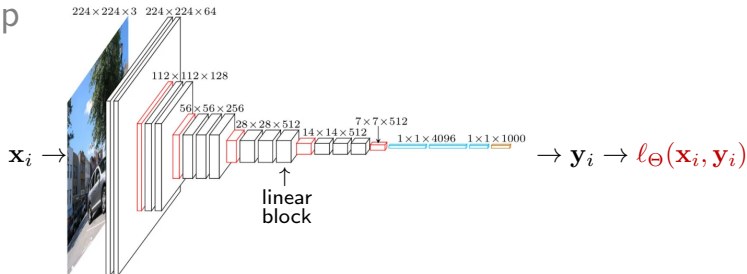


Sometimes, the learning model is **linear** and the loss is **quadratic**.

This case can be solved in closed form.

The more data points  $(\mathbf{x}_i, \mathbf{y}_i)$  we have, the better.

# Wrap-up

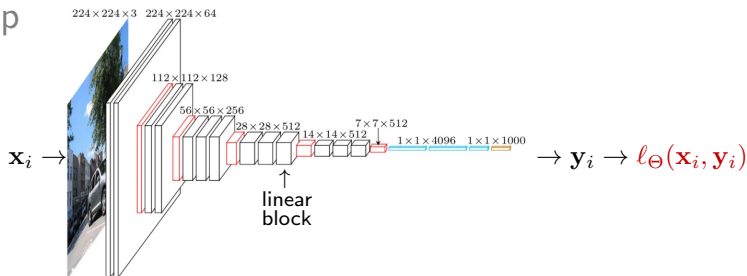


Sometimes, the learning model is **linear** and the loss is **quadratic**.  
This case can be solved in closed form.

The more data points  $(\mathbf{x}_i, \mathbf{y}_i)$  we have, the better.

- MLP: Linear blocks alternated with **nonlinear** functions

# Wrap-up



Sometimes, the learning model is **linear** and the loss is **quadratic**.  
This case can be solved in closed form.

The more data points  $(\mathbf{x}_i, \mathbf{y}_i)$  we have, the better.

- MLP: Linear blocks alternated with **nonlinear** functions
- **Deep linear networks**: Simple sequence of linear blocks

Saxe et al, Exact solutions to the nonlinear dynamics of learning in deep linear neural networks, 2013



## Suggested reading

For convexity and optimality, read Sections 3.1.1 and 3.1.3 of the book:

S. Boyd & L. Vandenberghe, “Convex optimization”. Cambridge University Press, 2009

Public download link: [https://web.stanford.edu/~boyd/cvxbook/bv\\_cvxbook.pdf](https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf)