

# Introduction to Deep Learning

## Tutorial 1

Gabriel Deza

Department of Industrial Engineering  
Tel Aviv University

November 4th and 6th, 2025

## Goal of tutorials:

- **Review** lecture content from week prior
- **Familiarize** ourselves with coding (IDEs, Python, NumPy, PyTorch, etc.)

## Course structure:

- 60% final exam
- 25% project (Dror)
- 6% assignment ( $3 \times 2\%$  each, binary grade)

**Course contact:** Do it in English via email at [gabrieldeza@mail.tau.ac.il](mailto:gabrieldeza@mail.tau.ac.il) (\*)

(\*) Please check Moodle and other readily available resources (including Google/ChatGPT) before emailing. Questions that can be easily answered there may not receive a reply.

# Overview

Structure for today:

- Review of **Linear models**: estimate based on a linear function of input.
  - **Regression**: predict a scalar-value target (e.g. stock price)
  - **Binary classification**: predict a binary label (e.g spam vs. non-spam email)
  - **Multiway classification**: predict a discrete label (e.g. object category from a list)
- Gradient descent
- Training procedure
- Code

# Linear Regression (Review)

**Given:** dataset  $\{(\mathbf{x}^{(i)}, t^{(i)})\}_{i=1}^N$  of input vectors  $\mathbf{x}^{(i)} \in \mathbb{R}^D$  and target  $t^{(i)} \in \mathbb{R}$

**Goal:** Model relationships between inputs and targets to be able to predict target given an input.

**Model:** Affine + linear

$$y = w_1x_1 + \cdots + w_Dx_D + b = \underbrace{\mathbf{w}^T \mathbf{x}}_{\text{weight}} + \underbrace{b}_{\text{bias}}$$

**Loss function:** squared error

$$\mathcal{L}(y, t) = \frac{1}{2}(y - t)^2$$

**Cost function:** loss averaged over all training examples:

$$\mathcal{J}(\mathbf{w}, b) = \frac{1}{2N} \sum_{i=1}^N (y^{(i)} - t^{(i)})^2 = \frac{1}{2N} \sum_{i=1}^N (\mathbf{w}^\top \mathbf{x}^{(i)} + b - t^{(i)})^2$$

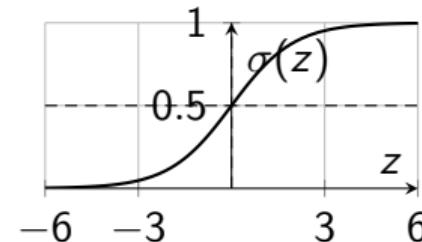
# Binary Classification (Review)

**Given:** dataset  $\{(\mathbf{x}^{(i)}, t^{(i)})\}_{i=1}^N$  of input vectors  $\mathbf{x}^{(i)} \in \mathbb{R}^n$  and binary targets  $t^{(i)} \in \{0, 1\}$

**Goal:** Learn a model that predicts the probability that  $t = 1$  given input  $\mathbf{x}$ .

**Model:** Affine + sigmoid

$$y = \sigma(z) = \frac{1}{1 + e^{-z}}, \quad z = \mathbf{w}^\top \mathbf{x} + b$$



**Loss function:** cross-entropy

$$\mathcal{L}(y, t) = -[t \log(y) + (1 - t) \log(1 - y)]$$

**Cost function:** loss averaged over all training examples:

$$\mathcal{J}(\mathbf{w}, b) = \frac{1}{N} \sum_{i=1}^N \left[ -t^{(i)} \log(y^{(i)}) - (1 - t^{(i)}) \log(1 - y^{(i)}) \right]$$

# Multi-class Classification (Review)

**Given:** dataset  $\{(\mathbf{x}^{(i)}, t^{(i)})\}_{i=1}^N$  of input vectors  $\mathbf{x}^{(i)} \in \mathbb{R}^n$  and targets  $t^{(i)} \in \{1, 2, \dots, K\}$

**Goal:** Learn a model that predicts the probability distribution over  $K$  classes.

**Model:** Affine + softmax

$$p(y = k \mid \mathbf{x}) = \frac{\exp(\mathbf{w}_k^\top \mathbf{x} + b_k)}{\sum_{j=1}^K \exp(\mathbf{w}_j^\top \mathbf{x} + b_j)}, \quad k = 1, \dots, K$$

**Loss function:** categorical cross-entropy

$$\mathcal{L}(\mathbf{p}, t) = - \sum_{k=1}^K \mathbf{1}\{t = k\} \log p_k$$

**Cost function:** loss averaged over all training examples:

$$\mathcal{J}(\mathbf{W}, b) = \frac{1}{N} \sum_{i=1}^N \left[ - \sum_{k=1}^K \mathbf{1}\{t^{(i)} = k\} \log p_k^{(i)} \right]$$

## Where does cross-entropy come from? Maximum Likelihood

**Maximum Likelihood:** What  $\mathbf{W}$  maximizes the probability of being correct:

$$\max_{\mathbf{W}} \prod_{i=1}^N p(t^{(i)} | \mathbf{x}^{(i)}, \mathbf{W})$$

$$p(\mathbf{t} | \mathbf{X}, \mathbf{W}) = \prod_{i=1}^N p(t^{(i)} | \mathbf{x}^{(i)}, \mathbf{W}) = \prod_{i=1}^N \prod_{k=1}^K p(k | \mathbf{x}^{(i)}, \mathbf{W}) \mathbf{1}\{t^{(i)}=k\}$$

Given large enough dataset, product of many terms less than 1 leads to numerical underflow  $\Rightarrow$  maximize a monotonically increasing function of  $p(\mathbf{t} | \mathbf{X}, \mathbf{W})$

$$\log \left( \prod_{i=1}^N \prod_{k=1}^K p(k | \mathbf{x}^{(i)}, \mathbf{W}) \mathbf{1}\{t^{(i)}=k\} \right) = \sum_{i=1}^N \sum_{k=1}^K \mathbf{1}\{t^{(i)} = k\} \log(p(k | \mathbf{x}^{(i)}, \mathbf{W}))$$

To convert to a loss, we negate the term and end up getting the most common loss function in ML.

# How do you solve for $w$ and $b$ ?

$$\min_{w,b} \mathcal{J}(w, b)?$$

Two strategies for optimization:

- **Direct optimization**: Solve system of partial derivatives set to 0. Works for a few very specific problems (e.g., OLS).
- **Iterative methods** (e.g., GD): repeatedly update a solution in a direction that improves the current solution.

**Gradient Descent**: Update each weight in the opposite direction of the gradient, i.e. the direction of **steepest decrease of the cost function**.

$$w_i \leftarrow w_i - \alpha \underbrace{\frac{\partial \mathcal{J}}{\partial w_i}}_{\text{derivative}}$$

Matrix form  $\mathbf{w} \leftarrow \mathbf{w} - \alpha \underbrace{\nabla \mathcal{J}(\mathbf{w})}_{\text{gradient}}$

# Training a model using gradient descent

---

**Algorithm 1** Gradient Descent Training Loop

---

- 1: **Input:** training data  $\{(\mathbf{x}^{(i)}, t^{(i)})\}_{i=1}^N$ , initial parameters  $\mathbf{w}, b$ , learning rate  $\alpha$ , number of epochs  $N$
  - 2: **for**  $i = 1$  to  $N$  **do**
  - 3:   Compute predictions:  $\hat{y} \leftarrow f(\mathbf{x}; \mathbf{w}, b)$
  - 4:   Compute cost:  $\mathcal{J} \leftarrow \text{cost}(\hat{y}, t)$
  - 5:   Compute gradients  $\nabla_{\mathbf{w}}\mathcal{J}, \nabla_b\mathcal{J}$  (via **backpropagation**)
  - 6:   Update parameters:  $\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}}\mathcal{J}, \quad b \leftarrow b - \alpha \nabla_b\mathcal{J}$
  - 7: **end for**
  - 8: **Output:** trained parameters  $\mathbf{w}, b$
-

## Code time

Let's now go to Jupyter notebook to learn about PyTorch.

<https://colab.research.google.com/drive/1coRkBCDCZ3LtYa7hoB8T30TVU7ow74Uj?usp=sharing>