Theoretical Principles of Deep Learning

Class I: Introduction and Core Concepts

Hédi Hadiji

CentraleSupelec

hedi.hadiji@l2s.centralesupelec.fr

Dec 4, 2023

Short Introduction

Hédi Hadiji: *hedi.hadiji@l2s.centralesupelec.fr*

- Ph.D. at Orsay in Bandit theory
- Postdoc in Amsterdam on Online Learning
- Assistant Professor at L2S of CS

Research area: Mathematician, work in bandits, online optimization and learning theory. At the intersection of Machine Learning, Statistics and Optimization.

Goal of the Class

Build a Theory of Deep Learning

- Describe a theoretical framework which relates to the practice of deep learning
- See how this can inform the intuition of practitioners (?)
- Understand the limitations of these approaches

Disclaimer: Deep learning theory is young, active, immature field (e.g. no established textbooks.) Practical successes are miles ahead of what theorists can study.
There is not even a consensus on what it means to “understand” deep learning.

Reading Material and References

Standard material: Diverse. Main content is what is discussed in class. Links on Edunao.

- *Deep Learning theory*, Lecture notes by Matus Telgarsky
- *Theory of Deep Learning*, Arora et al.
- *Patters, Predictions, Actions*, Hardt and Recht

Prerequisites:

- Mathematics: probability theory, real analysis, statistical learning basics, optimization basics
- Programming: `python`, `numpy`, `pytorch`

Assessment

Reading projects: by groups of 2. Each group will be assigned a paper to present. Last class will be a poster session.

- Poster session: print a few slides and prepare to explain the paper to me in 5 - 10 minutes
- Short report: write a 4-8 page discussion on the paper, including a toy experiment.

More details to come.

Timeline

Careful: the rooms change all the time.

- lundi 3 décembre
- lundi 11 décembre
- lundi 18 décembre
- lundi 8 janvier
- lundi 15 janvier
- lundi 22 janvier
- lundi 29 janvier
- jeudi 8 février $\rightarrow$ Poster session + report.

Each time slot is 1h30 lecture, followed by 1h30 exercise session or practical session. Please bring your computers.

Table of Contents

Today's Lecture

Setting up the stage: only shallow learning for now

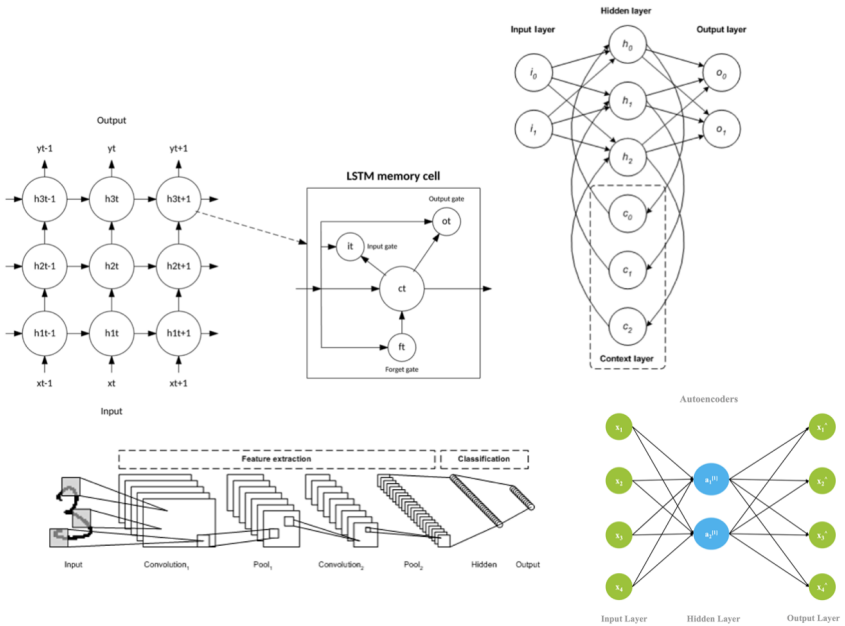
- Set up some basic mathematical definitions
- Recall the setting of supervised learning
- An example of theory working correctly for 1-layer nets

Reading material for this class:

- Patterns, Prediction and Actions, Chapter 3

What is Deep Learning?

Machine learning methods that involve *deep neural networks*, i.e., a function $\mathbb{R}^d \to \mathbb{R}^p$ that can be represented by a computational graph.



Examples of deep learning architectures, from the IBM blog

Practical success of DL

State of the art in:

- vision (classification, detection, segmentation, ..)
- natural language processing (machine translation, information extraction, ...)
- protein folding
- generative AI (image, text, music, ...)
- reinforcement learning (games, robotics(?))

Every aspect of engineering has been affected by deep learning.

Particularly relevant for our class: deep neural nets are the go-to tool for most **Supervised Learning** task.

Deep learning in practice

Important real life components of deep learning: large computational power (GPUs) + tricks + empirical knowledge + trial and error.

Regularisation tricks like

- learning rate schedules,
- data augmentation,
- weight decay,
- early stopping,
- ensembles (combine predictions of different models),
- stochastic regularization (e.g. dropout).

We ignore these aspects in this class: theory is about simplified models.

What is Deep Learning Theory?

A model h is a function from feature space $\mathcal{X}$ to the response space $\mathcal{Y}$, parameterized by some weights (or parameters).

For theory, what distinguishes deep nets is that they are:

- **highly non-linear**: composition of many non-linear functions
- **overparameterized**: models have many more degrees of freedom than the number of data points they are trained on
- **efficiently differentiable**: GPUs + backpropagation allows for fast computation of gradients.

Goal of the class

Why do methods that combine these three aspects have good practical success?

Already more concrete and modest than 'understand' deep learning... but still quite out of reach. Let us make the question more concrete first.

Table of Contents

Supervised Learning

Goal

Learning by examples: predict the response Y to some input X, based on examples.

Examples of Supervised Learning tasks

- X temperature + cloud positions today; Y temperature tomorrow
- X first words in an English sentence; Y next word
- X image; Y whether it contains a cat or not
- X sequence of amino acids; Y protein shape

Supervised Learning: Formal Framework

A learner has access to a sample S of n data points $(X_i, Y_i)_{1 \leqslant i \leqslant n}$.

- $X_i \in \mathcal{X}$ are called *features*
- $Y_i \in \mathcal{Y}$ are called *responses* (or labels when $\mathcal{Y}$ is finite)

The data are i.i.d. from an unknown distribution $(X_i, Y_i) \sim \mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$.

The goal of the learner is to output a prediction $\widehat{Y}$ of the response Y for new features X not in the sample.

The quality of prediction is measured by a *loss function* $\ell(\cdot, \cdot) : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$.

Objective of Supervised Learning

Given a sample S, find a hypothesis $h_S : \mathcal{X} \to \mathcal{Y}$ such that the risk

$$R(h_S) = \mathbb{E}_{(X,Y)\sim\mathcal{D}}\left[\ell(h_S(X), Y)\right]$$

is small with high probability.

(Note that the quantity above is a random variable because h_S may depend on the sample. This is deliberately left vague for the moment.)

Empirical Risk Minimization

Perhaps the most natural idea in supervised learning is to look for a hypothesis that minimizes the **empirical risk**

$$h \in \arg\min_{?} \frac{1}{n} \sum_{i=1}^{n} \ell(h(X_i), Y_i) \,.$$

This raises questions:

0. How should one choose the hypothesis space $\mathcal{H}$ over which to perform the minimization?
1. **Optimisation:** how would we compute this minimiser?
2. **Generalization:** why would this be good?

Terminology: we denote the empirical risk on sample S as

$$\widehat{R}_S(h) = \frac{1}{n} \sum_{i=1}^{n} \ell(h(X_i), Y_i)$$

Example: Linear Least-squares Regression

Setting: $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \mathbb{R}$

Linear Models

Take $\mathcal{H}$ to be the set of affine functions $\mathbb{R}^d \to \mathbb{R}$, i.e. functions of the form

$$h_{w,b} : x \mapsto \langle w, x \rangle + b \,,$$

for some $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$.

Square loss

$$\ell(\widehat{y}, y) = \frac{1}{2}(\widehat{y} - y)^2$$

Then ERM is the least-squares

$$(\widehat{w}, \widehat{b}) \in \arg\min_{w,b} \left\{ \frac{1}{2} \sum_{i=1}^{n} \left(\langle w, X_i \rangle + b_i - Y_i \right)^2 \right\}$$

can be computed explicitly.

Example: Neural networks

Setting: $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \mathbb{R}^p$

Feedforward neural networks

For dimensions p, q, r, a **layer** is a function $\mathbb{R}^p \to \mathbb{R}^r$

$$\Phi_{\sigma,A,b} : x \mapsto \sigma\big(Ax + b\big)$$

where $\sigma : \mathbb{R}^q \to \mathbb{R}^r$ is a simple non-linear function, A is an $q \times p$ matrix and $b \in \mathbb{R}^q$ is a vector.

A **neural network** is a function of the form

$$h : x \mapsto \Phi_{\sigma_L,A_L,b_L} \circ \cdots \circ \Phi_{\sigma_0,A_0,b_0}(x)\,.$$

Loss depends on applications.

Example: Neural networks II

A **neural network** is a function of the form

$$h : x \mapsto \Phi_{\sigma_L, A_L, b_L} \circ \cdots \circ \Phi_{\sigma_0, A_0, b_0}(x) .$$

The sequence of activation functions (σ_i) and dimensions determine the architecture, and the values of (A_i) and (b_i) the weights of a model.

In practice, training a model means finding weights with small empirical risk (aka train loss), with an optimiser based on gradient descent.

(We will expand on this example in future classes :))

What there is to understand in deep learning

Phenomena of interests to theorists in this class:

- **Approximation** Deep neural nets can approximate functions well
- **Optimization** Deep neural nets are efficiently trainable
- **Generalization** Small training loss converts to small test loss
- **Representation learning** Deep nets "learn useful features" (e.g., trained convolutional layers look like filters)

Successful theory should explain these phenomena.

(Other crucial aspects that are not discussed in this class: explainability, uncertainty quantification, robustness)

Table of Contents

The Single-Layer Perceptron

The Future of the Past of AI:

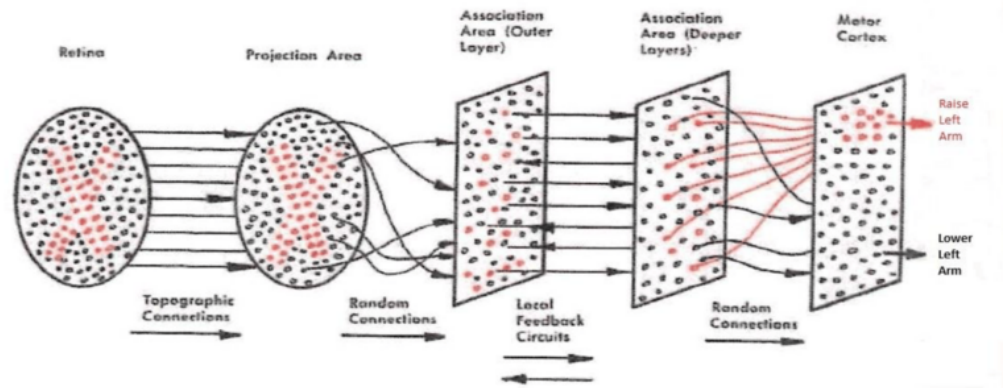


FIG. 1 — Organization of a biological brain. (Red areas indicate active cells, responding to the letter X.)

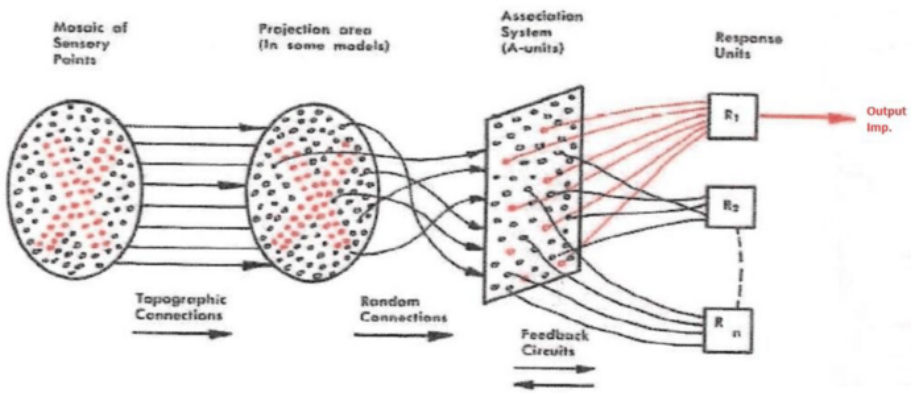


FIG. 2 — Organization of a perceptron.

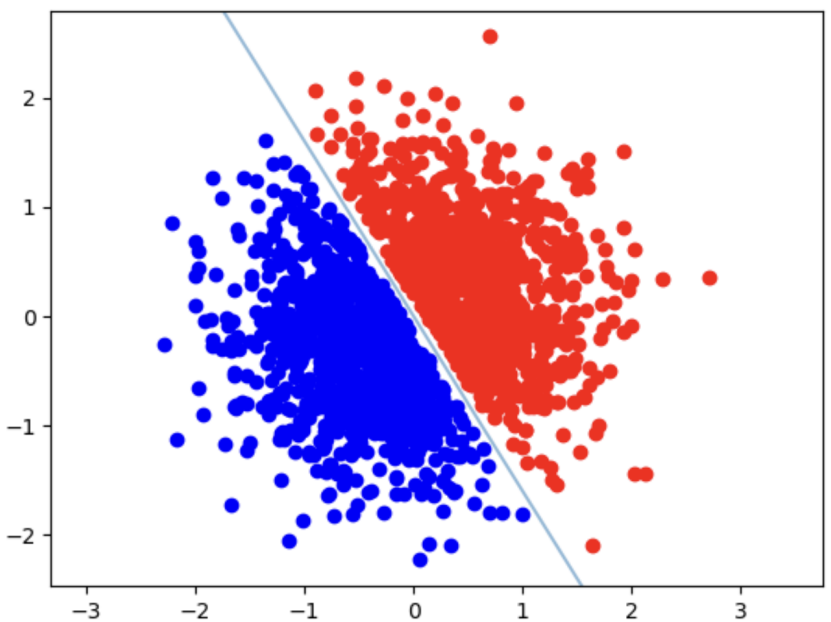
Invented by Rosenblatt, before general purpose computers.

Setting: Binary classification

$\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{-1, 1\}$. Goal: 0-1–loss, i.e. $\ell(\widehat{y}, y) = 2 \cdot \mathbb{1}\{\widehat{y} \neq y\} - 1$.

Assumption: Separable distribution

The data comes from a distribution $\mathcal{D}$ such that the samples are linearly separable with probability 1.



(Very strong) assumption

Binary classification with linear models

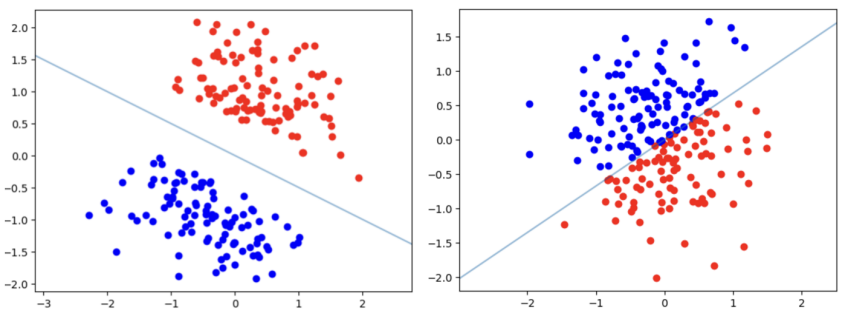
We parameterize hyperplanes by orthogonal vectors, i.e., for $w \in \mathbb{R}^d$, denote by

$$H_w = \{x \in \mathbb{R}^d | \langle x, w \rangle = 0\} \quad \text{and} \quad h_w(x) = \mathbb{1}\{\langle x, w \rangle > 0\}.$$

The Perceptron will pick a hypothesis among the ones of this form.

Definition (Margin of a separating hyperplane)

The *margin* $\gamma(S, w)$ of a hyperplane H_w that separates the data S is the distance between the hyperplane and the closest point instance.



If the margin is big, we can move the hyperplane and keep a good score.

The Perceptron: Algorithm

Objective of the Perceptron

Find a hyperplane H_w that separates the data with some margin.

Algorithm: Perceptron

Dataset S
Initialize: $w_0 = 0$
for $t = 0, 1, 2 \ldots,$ **do**
  - Choose an index $i \in \{1, \ldots, n\}$

$$w_{t+1} = \begin{cases} w_t + y_i x_i & \text{if } \langle w, x_i \rangle y_i < 1 \\ w_t & \text{otherwise.} \end{cases}$$

end

We say the algorithm makes a **margin mistake** if $\langle w, x_i \rangle y_i < 1$.

Perceptron finds a separating hyperplane

Define the diameter of the sample and the sample margin to be, respectively

$$D(S) = \max_{(x,y)\in S} \|x\| \quad \text{and} \quad \gamma(S) = \max_{\substack{w\in\mathbb{R}^d \\ \text{separating}}} \min_{(x,y)\in S} d(x, H_w)\,.$$

Theorem (Mistake Bound for the Perceptron, (Novikoff))

If a sample S is linearly separable, then the Perceptron algorithm makes at most

$$\frac{2 + D(S)^2}{\gamma(S)^2}$$

margin mistakes.

Proof: Board.

Generalization of the Perceptron

We know that the Perceptron finds a separating hyperplane, i.e., classifies the sample perfectly.

What about unseen data? We can bound the probability of error of the perceptron, if we make some assumptions on the distribution.

Assumption: Separable distribution

The data comes from a distribution $\mathcal{D}$ such that the samples are linearly separable with probability 1.

i.e., there exists $w \in \mathbb{R}^d$ such that $\mathbb{P}_{(X,Y)\sim\mathcal{D}}[Y = 1 \mid \langle X, w\rangle > 0] = 1$.
(ignoring the cases in which $\mathbb{P}[\langle X, w\rangle = 0] > 0$)

Generalization of the Perceptron

Theorem (Generalisation of the Perceptron)

For a linearly separable distribution $\mathcal{D}$, if $w(S_n)$ is the output of the Perceptron on sample S_n, then the probability of making a margin mistake on future data is bounded by

$$\mathbb{P}_{(X,Y)\sim\mathcal{D}}\left[Y\langle w(S_n), X\rangle < 1\right] \leqslant \frac{1}{n+1}\mathbb{E}_{S_{n+1}\sim\mathcal{D}^{\otimes(n+1)}}\left[\frac{2 + D(S_{n+1})^2}{\gamma(S_{n+1})^2}\right]$$

This is an example of a **stability argument**, a general and powerful principle to derive generalisation bounds.

Stability argument

Algorithms that do not depend too much on a single data point generalize well

Proof: Board.

Table of Contents

Lessons from the Single-Layer Perceptron

When binary classification data is linearly separable, the Perceptron:

- Finds a separating hyperplane **efficiently**
- This separating hyperplane has good **quantifiable performance** on unseen data

E.g., if we want to predict with 99% accuracy on data with features of norm $\leqslant D$ and margin $\geqslant \gamma$, it suffices to train a Perceptron on enough sample points to have

$$\frac{(2+D^2)/\gamma^2}{n_{1\%}+1} \leqslant 1\%, \quad \text{i.e.} \quad n_{1\%} \geqslant 100\,\frac{2+D^2}{\gamma^2} - 1\,.$$

... and the proof is simple!

Can we get a similar result for a Resnet on Imagenet data?
Probably not... but we can try

Challenges in understanding deep learning

All of this was nice, but:

- Most data is not separable, hence non-linear methods.
- The single-layer perceptron can be seen as some form of gradient descent on a convex surrogate loss function. For multi-layer nets, the loss landscape is very non-convex. Why does gradient descent still work there?
- The generalization proof we saw is very specific to the algorithm.

We will discuss these points in later lectures.

Next Time: Approximation in neural nets

What if data is not linearly separable? If we understand the geometry well, can use *kernels*.

What about cats vs. dogs in images? Can we guess the correct geometry to separate the data?

If data is not linearly separable. Use non-linear methods. We will prove that neural networks can approximate arbitrarily well any function (Barron's theorem).

Thanks!
Let’s have a break