Mathematics of Deep Learning

Introduction & general overview

Lessons: Kevin Scaman

Practical details

Timeline

- **Dates:** 09/01/2024 - 12/03/2023 (8h30 - 11h45)
- **Format:** 8 classes (1h30 class + 1h30 TDs), 1 Exam (19/03, 8h30 - 10h30)

Validation

- One **homework** on 06/02. **Deadline:** 20/02.
- One **exam** on the 19/03.

Contact

- **Email:** `kevin.scaman@inria.fr`

Objectives for today and beyond

Overall objective

1. Explore the **mathematical aspects** of deep learning.
2. Understand **why** deep learning architectures work so well in practice.

Objectives for today and beyond

Overall objective

1. Explore the **mathematical aspects** of deep learning.
2. Understand **why** deep learning architectures work so well in practice.

Today's objective

1. Understand **what** is deep learning.
2. Set a **mathematical framework** for our analysis.
3. Learn about **simple neural networks**: Multi-layer perceptrons (MLP).

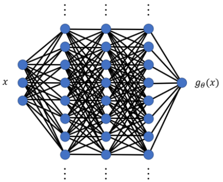
What is Deep Learning?

What is Deep Learning?

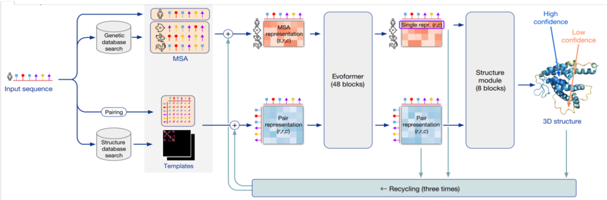
First, what are neural networks?

- The notion changed over the last 8 decades…!
- From early neural networks imitating real neurons…
- To highly complex architectures with multiple sub-modules.

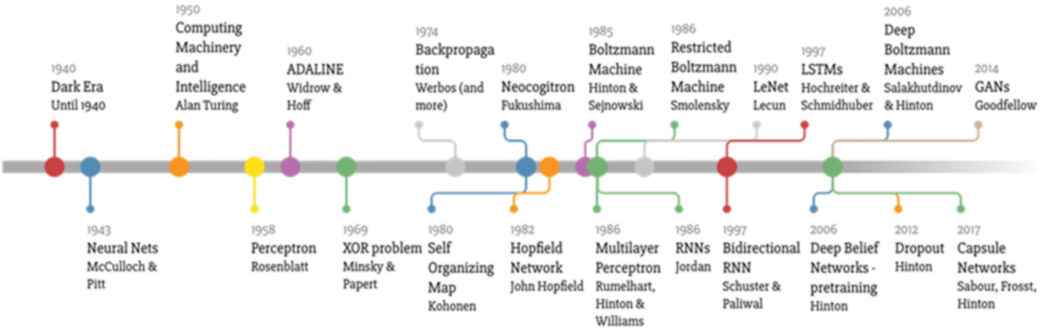
Multi-Layer Perceptron
(Rumelhart, Hinton, Williams, 75)



AlphaFold
(Jumper et.al., 2021)



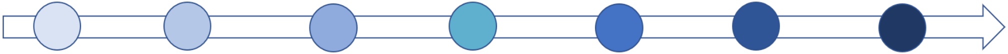
Timeline of Deep Learning



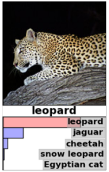
source: Mourtzis & Angelopoulos (2020)

Recent deep learning applications

Image classification	Image generation	Text generation	Board games (Go, chess)	Strategy games (Starcraft, Dota2)	Code generation	Protein folding



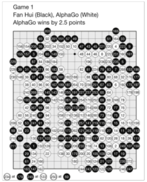
2012	...	...	...	...	...	2021



[Krizhevsky et.al., 2012]

[Goodfellow et.al., 2014]
[Karras et.al., 2018]
[news.artnet.com, 2018]

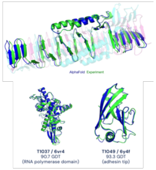
[Vaswani et.al., 2017]
[Hochreiter et.al., 2018]
[Devlin et.al., 2019]



[Silver et.al., 2016]
[Silver et.al., 2017]

[openai.com/five, 2018]
[Oriol et.al., 2019]

[Brown et.al., 2020]
[Chen et.al., 2021]



[Jumper et.al., 2021]

Since 2021

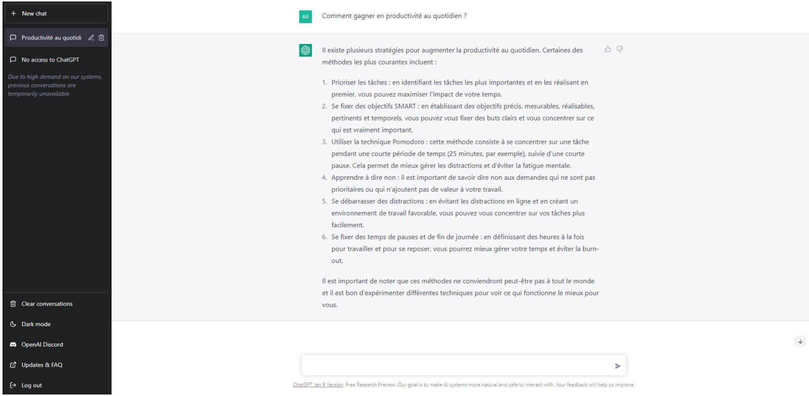
Thousands of applications

- **Voice/audio/music generation:** MusicGen, MusicLM, MusicLDM, Jukebox, HeyGen
- **Voice to text:** Whisper
- **Image generation/deep-fakes:** Dalle-3, MidJourney, Stable Diffusion XL
- **Text generation/chatbots:** ChatGPT, GPT4, LLama, Claude, Mistral
- **Video generation:** Make-a-video, HeyGen
- **Code generation/automatic app creation:** Codex, Code LLama, phi-1.5, AutoGPT
- **Strategic games (Go, chess, Starcraft, diplomacy):** AlphaZero, LeelaChess, Cicero
- **Autonomous driving**
- ...

Most recent breakthroughs: image generation (Dalle3, SD, MJ, ...)

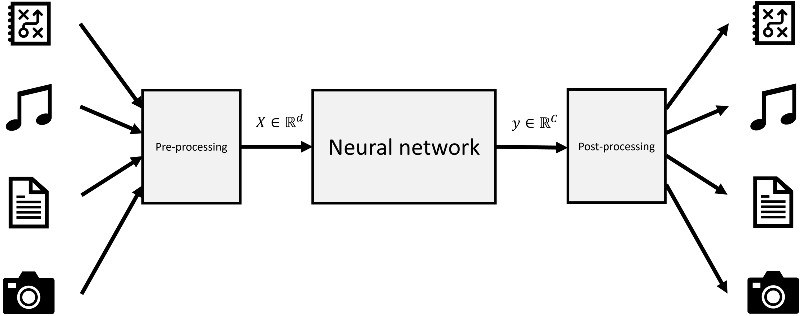
Images generated from prompts using MidJourney (https://www.midjourney.com/)

Most recent breakthroughs: text generation (GPT4, LLama, Claude, ...)



source: OpenAI's ChatGPT (`https://chat.openai.com/`)

What is Deep Learning? (usual setup)



What is Deep Learning? (required skills)

What do you need to create a DL architecture?

1. Know how to **encode/decode data**
    - Data loader, data augmentation, data handling during training, mini-batch, ...
    - Encoding layers, one-hot, tokenization, embeddings, ...

What is Deep Learning? (required skills)

What do you need to create a DL architecture?

1. Know how to **encode/decode data**
   - Data loader, data augmentation, data handling during training, mini-batch, ...
   - Encoding layers, one-hot, tokenization, embeddings, ...
2. Know how to **create a neural network**
   - Different types of layers, attention mechanism, batch normalization, ...
   - Multiple architectures: MLPs, RNNs, CNNs, GNNs, Transformers, ...

What is Deep Learning? (required skills)

What do you need to create a DL architecture?

1. Know how to **encode/decode data**
   - Data loader, data augmentation, data handling during training, mini-batch, ...
   - Encoding layers, one-hot, tokenization, embeddings, ...
2. Know how to **create a neural network**
   - Different types of layers, attention mechanism, batch normalization, ...
   - Multiple architectures: MLPs, RNNs, CNNs, GNNs, Transformers, ...
3. Know how to **train the neural network**
   - Optimization perspective, auto-diff, SGD, Adam, momentum, ...
   - Weight initialization, loss functions, scheduling, hyper-parameter optimization...

What is Deep Learning? (twitter wisdom)

Yann LeCun
@ylecun

Some folks still seem confused about what deep learning is. Here is a definition:

DL is constructing networks of parameterized functional modules & training them from examples using gradient-based optimization.... facebook.com/722677142/post...

Traduire le Tweet

4:32 PM · 24 déc. 2019 · Facebook

What is Deep Learning? (twitter wisdom)

Yann LeCun
@ylecun

Some folks still seem confused about what deep learning is. Here is a definition:

DL is constructing networks of parameterized functional modules & training them from examples using gradient-based optimization....
facebook.com/722677142/post...

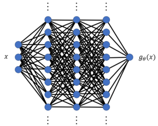
Traduire le Tweet

4:32 PM · 24 déc. 2019 · Facebook

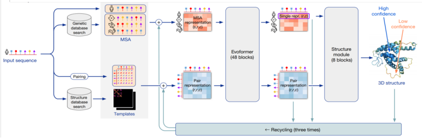
Why Deep Learning Now?

- Five decades of research in machine learning

Multi-Layer Perceptron
(Rumelhart, Hinton, Williams, 75)

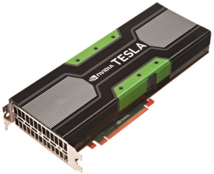


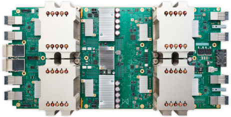
AlphaFold
(Jumper et.al., 2021)



Why Deep Learning Now?

- Five decades of research in machine learning
- CPUs/GPUs/storage developed for other purposes





Why Deep Learning Now?

- Five decades of research in machine learning
- CPUs/GPUs/storage developed for other purposes
- lots of data from “the internet”

Why Deep Learning Now?

- Five decades of research in machine learning
- CPUs/GPUs/storage developed for other purposes
- lots of data from "the internet"
- tools and culture of collaborative and reproducible science

Why Deep Learning Now?

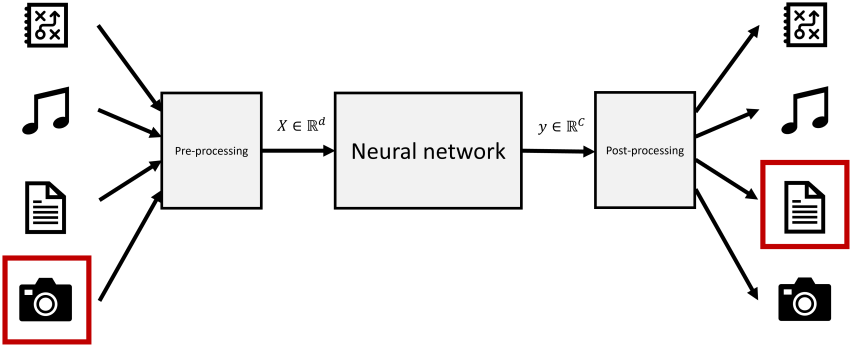
- Five decades of research in machine learning
- CPUs/GPUs/storage developed for other purposes
- lots of data from “the internet”
- tools and culture of collaborative and reproducible science
- resources and efforts from large corporations

Machine Learning pipeline

A short recap

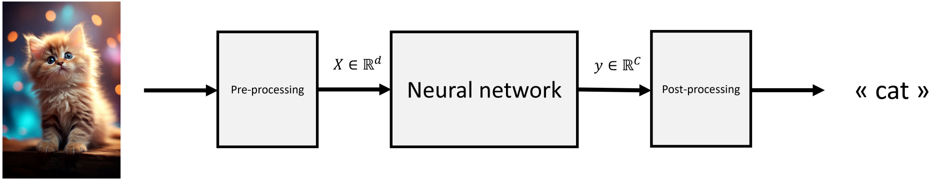
Simple example: cats vs. dogs

Typical binary classification task. Objective is to distinguish **cat images from dog images**.



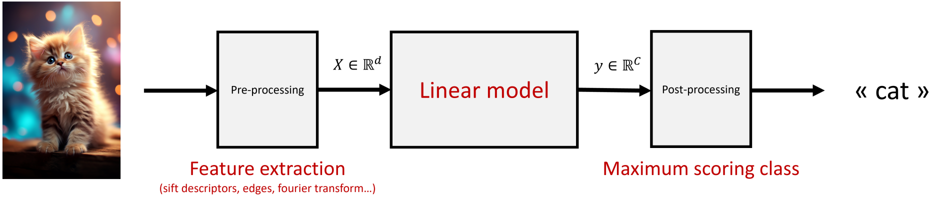
Simple example: cats vs. dogs

Output class is represented as a **2d vector** ($(0,1)$ for "cat" and $(1,0)$ for "dog").



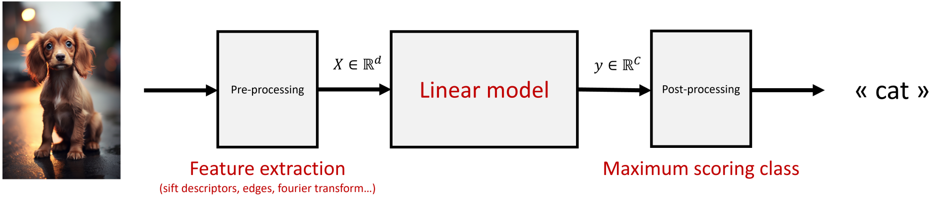
Simple example: cats vs. dogs (linear model)

Image features (sift, wavelets,...) are extracted and given as input to the model.



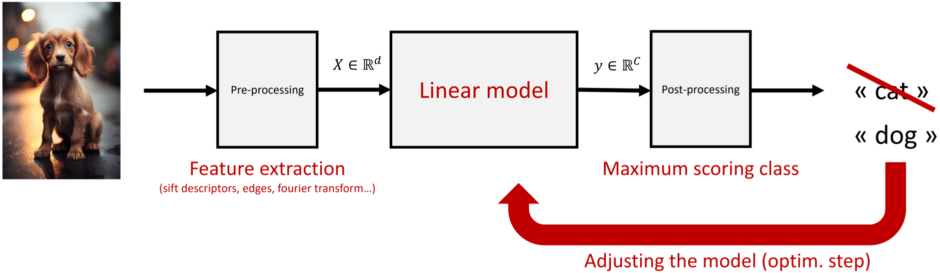
Simple example: cats vs. dogs (inference)

The model makes a **prediction** ("cat" or "dog") for a given image.



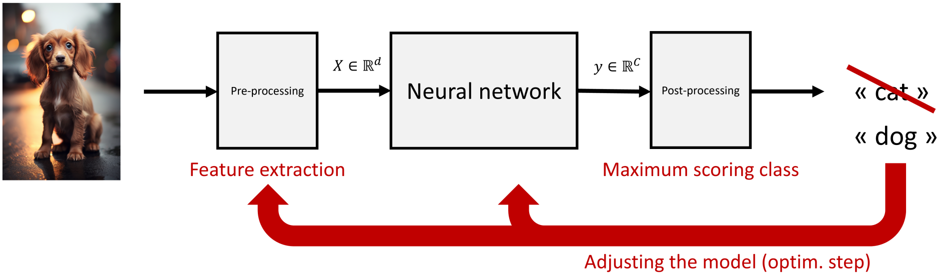
Simple example: cats vs. dogs (training loop)

If the prediction is false, the **model updates its parameters** to improve its prediction.



Simple example: cats vs. dogs (deep learning version)

In deep learning, we can train the **whole pipeline** using **automatic differentiation**.



Typical Machine Learning setup

Data distribution

Let $\mathcal{X}, \mathcal{Y}$ be an input and output space and $\mathcal{D}$ a distribution over $(\mathcal{X}, \mathcal{Y})$. Then, we denote our (test) input/output pair as

$$(X, Y) \sim \mathcal{D}$$

Typical Machine Learning setup

Data distribution

Let $\mathcal{X}, \mathcal{Y}$ be an input and output space and $\mathcal{D}$ a distribution over $(\mathcal{X}, \mathcal{Y})$. Then, we denote our (test) input/output pair as

$$(X, Y) \sim \mathcal{D}$$

Risk minimization (a.k.a. supervized ML)

The objective of *risk minimization* is to find a minimizer $\theta^\star \in \mathbb{R}^p$ of the optimization problem

$$\min_{\theta \in \mathbb{R}^p} \mathbb{E}\big(\ell(g_\theta(X), Y)\big)$$

where $\ell : \mathcal{Y}^2 \to \mathbb{R}_+$ is a loss function and $g_\theta : \mathcal{X} \to \mathcal{Y}$ a model parameterized by $\theta \in \mathbb{R}^p$.

Typical Machine Learning setup

Data distribution

Let $\mathcal{X}, \mathcal{Y}$ be an input and output space and $\mathcal{D}$ a distribution over $(\mathcal{X}, \mathcal{Y})$. Then, we denote our (test) input/output pair as

$$(X, Y) \sim \mathcal{D}$$

Risk minimization (a.k.a. supervized ML)

The objective of *risk minimization* is to find a minimizer $\theta^\star \in \mathbb{R}^p$ of the optimization problem

$$\min_{\theta \in \mathbb{R}^p} \mathbb{E}\big(\ell(g_\theta(X), Y)\big)$$

where $\ell : \mathcal{Y}^2 \to \mathbb{R}_+$ is a loss function and $g_\theta : \mathcal{X} \to \mathcal{Y}$ a model parameterized by $\theta \in \mathbb{R}^p$.

⚠ The target loss (e.g. accuracy) may be hard to train, and can thus be different from the one used as objective during training!

Typical Machine Learning setup (back to cats and dogs)

- **Input data:** $X \in [0, 255]^{w \times h \times 3}$ are images encoded as **tensors** (i.e. high-dim. matrices)

Typical Machine Learning setup (back to cats and dogs)

- **Input data:** $X \in [0, 255]^{w \times h \times 3}$ are images encoded as **tensors** (i.e. high-dim. matrices)
- **Output data:** $Y \in \mathbb{R}^2$ are classes, **one-hot** encoded (i.e. $Y_i = 1$ iff i is the true class).

Typical Machine Learning setup (back to cats and dogs)

- **Input data:** $X \in [0,255]^{w \times h \times 3}$ are images encoded as **tensors** (i.e. high-dim. matrices)
- **Output data:** $Y \in \mathbb{R}^2$ are classes, **one-hot** encoded (i.e. $Y_i = 1$ iff i is the true class).
- **Training data:** Image-label pairs $(x_1, y_1)_{i \in [\![1,n]\!]}$ (n number of data points).

Typical Machine Learning setup (back to cats and dogs)

- **Input data:** $X \in [0, 255]^{w \times h \times 3}$ are images encoded as **tensors** (i.e. high-dim. matrices)
- **Output data:** $Y \in \mathbb{R}^2$ are classes, **one-hot** encoded (i.e. $Y_i = 1$ iff i is the true class).
- **Training data:** Image-label pairs $(x_1, y_1)_{i \in [\![1,n]\!]}$ (n number of data points).
- **Model:** $g_\theta : X \mapsto \langle \theta, f(X) \rangle$, where $f(X) \in \mathbb{R}^F$ are pre-computed features and $\theta \in \mathbb{R}^F$.

Typical Machine Learning setup (back to cats and dogs)

- **Input data:** $X \in [0, 255]^{w \times h \times 3}$ are images encoded as **tensors** (i.e. high-dim. matrices)
- **Output data:** $Y \in \mathbb{R}^2$ are classes, **one-hot** encoded (i.e. $Y_i = 1$ iff i is the true class).
- **Training data:** Image-label pairs $(x_1, y_1)_{i \in [\![1,n]\!]}$ (n number of data points).
- **Model:** $g_\theta : X \mapsto \langle \theta, f(X) \rangle$, where $f(X) \in \mathbb{R}^F$ are pre-computed features and $\theta \in \mathbb{R}^F$.
- **Loss function (test):** $\ell(y, y') = \mathbb{1}\{\mathrm{argmax}_i\, y'_i = \mathrm{argmax}_i\, y_i\}$ (accuracy)

Typical Machine Learning setup (back to cats and dogs)

- **Input data:** $X \in [0, 255]^{w \times h \times 3}$ are images encoded as **tensors** (i.e. high-dim. matrices)
- **Output data:** $Y \in \mathbb{R}^2$ are classes, **one-hot** encoded (i.e. $Y_i = 1$ iff i is the true class).
- **Training data:** Image-label pairs $(x_1, y_1)_{i \in [\![1,n]\!]}$ (n number of data points).
- **Model:** $g_\theta : X \mapsto \langle \theta, f(X) \rangle$, where $f(X) \in \mathbb{R}^F$ are pre-computed features and $\theta \in \mathbb{R}^F$.
- **Loss function (test):** $\ell(y, y') = \mathbb{1}\{\text{argmax}_i\, y'_i = \text{argmax}_i\, y_i\}$ (accuracy)
- **Loss function (train):** $\ell(y, y') = -\sum_i y'_i \ln \left(\exp(y_i) / \sum_j \exp(y_j) \right)$ (cross entropy)

Training objective

Empirical risk minimization

Let $(x_i, y_i)_{i \in [\![1,n]\!]}$ be a collection of n observations drawn independently according to $\mathcal{D}$. Then, the objective of *empirical risk minimization* (ERM) is to find a minimizer $\hat{\theta}_n \in \mathbb{R}^p$ of

$$\min_{\theta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} \ell(g_\theta(x_i), y_i)$$

Training objective

Empirical risk minimization

Let $(x_i, y_i)_{i \in [\![1,n]\!]}$ be a collection of n observations drawn independently according to $\mathcal{D}$.
Then, the objective of *empirical risk minimization* (ERM) is to find a minimizer $\hat{\theta}_n \in \mathbb{R}^p$ of

$$\min_{\theta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} \ell(g_\theta(x_i), y_i)$$

Optimization by gradient descent

We can minimize this loss by iterating

$$\theta_{t+1} = \theta_t - \eta \nabla \hat{\mathcal{L}}_n(\theta_t)$$

where $\eta > 0$ is a fixed step-size and $\hat{\mathcal{L}}_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(g_\theta(x_i), y_i)$ is our objective.

Typical loss functions

- In its simplest form, the **accuracy** is $\ell(y, y') = \mathbb{1}\{y \neq y'\}$.

Typical loss functions

- In its simplest form, the **accuracy** is $\ell(y, y') = \mathbb{1}\{y \neq y'\}$.
- For **classification** tasks, we usually use $\mathcal{Y} = \mathbb{R}^C$ where C is the number of classes, and
  - $\ell(y, y') = \mathbb{1}\{\operatorname{argmax}_i y'_i = \operatorname{argmax}_i y_i\}$ (top-1 accuracy) or,
  - $\ell(y, y') = -\sum_i y'_i \ln\left(\exp(y_i)/\sum_j \exp(y_j)\right)$ (cross entropy).

Typical loss functions

- In its simplest form, the **accuracy** is $\ell(y, y') = \mathbb{1}\{y \neq y'\}$.
- For **classification** tasks, we usually use $\mathcal{Y} = \mathbb{R}^C$ where C is the number of classes, and
  - $\ell(y, y') = \mathbb{1}\{\operatorname{argmax}_i y'_i = \operatorname{argmax}_i y_i\}$ (top-1 accuracy) or,
  - $\ell(y, y') = -\sum_i y'_i \ln\left(\exp(y_i)/\sum_j \exp(y_j)\right)$ (cross entropy).
- For **regression** tasks, we usually use $\mathcal{Y} = \mathbb{R}^d$ and
  - $\ell(y, y') = \|y - y'\|_2^2 = \sum_i (y_i - y'_i)^2$ (mean square error) or,
  - $\ell(y, y') = \|y - y'\|_1 = \sum_i |y_i - y'_i|$ (mean absolute error).

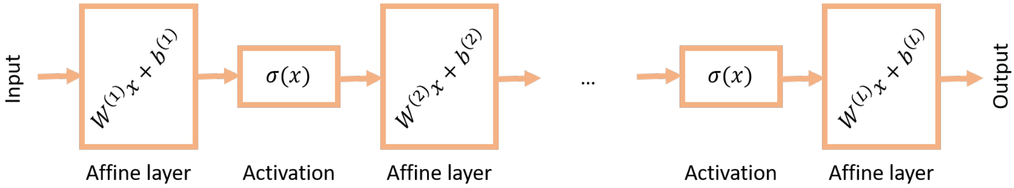
Recap

- Learning is rephrased as minimizing a **loss function** over the **training dataset**.
- Loss is typically **cross entropy** for classification and **MSE** for regression.
- Training achieved by (stochastic) **gradient descent** (or its variants).
- The whole pipeline is trained (i.e. its parameters are optimized) using **autodiff**.

Multi-Layer Perceptron

Definition and first properties

Multi-Layer Perceptron (Rumelhart, Hinton, Williams, 75)



Details

- **Idea:** Composition of **affine** (also called linear) and **activation** (simple non-linear coordinate-wise) functions. Simple extension of linear models.
- **Activations:** Coordinate-wise functions. (usually ReLU i.e. $\sigma(x)_i = \max\{0, x_i\}$).
- **Update rule:** $x^{(l+1)} = \sigma(W^{(l)}x^{(l)} + b^{(l)})$ (except for the last layer!).
- **Brain analogy:** A *"neuron"* is a coordinate of an activation layer.

Multi-Layer Perceptron: formal definition

Definition (MLP)

A *Multi-Layer Perceptron* (MLP) of depth $L \geqslant 1$, widths $(d^{(l)})_{l \in [\![0,L]\!]} \in {\mathbb{N}^*}^{L+1}$ and non-linear activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is a function $g_\theta : \mathbb{R}^{d^{(0)}} \to \mathbb{R}^{d^{(L)}}$ of the form:

$$g_\theta(x) = f^{(2L-1)} \circ f^{(2L-2)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(x)$$

where $\forall l \in [\![1, L]\!]$:

- Odd layers are **affine maps** $f^{(2l-1)}(x) = W^{(l)}x + b^{(l)}$ and $W^{(l)} \in \mathbb{R}^{d^{(l)} \times d^{(l-1)}}$, $b^{(l)} \in \mathbb{R}^{d^{(l)}}$.
- Even layers are **activation functions** $f^{(2l)}(x)_i = \sigma(x_i)$.
- Its **parameter** is $\theta = \left(W^{(l)}, b^{(l)}\right)_{l \in [\![1,L]\!]}$, and we denote as $g_\theta^{(l)}(x) = f^{(l)} \circ \cdots \circ f^{(1)}(x)$ the **intermediate output** after layer $l \in [\![0, 2L-1]\!]$.

Simple properties of ReLU networks

Definition (ReLU networks)

Let $\mathrm{ReLU}_{d,d'}$ be the space of all MLPs with ReLU activations s.t. $d^{(0)} = d$ and $d^{(L)} = d'$.

Simple properties of ReLU networks

Definition (ReLU networks)

Let $\text{ReLU}_{d,d'}$ be the space of all MLPs with ReLU activations s.t. $d^{(0)} = d$ and $d^{(L)} = d'$.

Lemma (stability)

$\text{ReLU}_{d,d}$ is **stable** by **addition** and **composition**. That is, $\forall g, g' \in \text{ReLU}_{d,d}$,

$$g + g' \in \text{ReLU}_{d,d} \quad \text{and} \quad g \circ g' \in \text{ReLU}_{d,d}$$

Simple properties of ReLU networks

Definition (ReLU networks)

Let $\text{ReLU}_{d,d'}$ be the space of all MLPs with ReLU activations s.t. $d^{(0)} = d$ and $d^{(L)} = d'$.

Lemma (stability)

$\text{ReLU}_{d,d}$ is **stable** by **addition** and **composition**. That is, $\forall g, g' \in \text{ReLU}_{d,d}$,

$$g + g' \in \text{ReLU}_{d,d} \quad \text{and} \quad g \circ g' \in \text{ReLU}_{d,d}$$

Lemma (continuity and piecewise linearity)

A ReLU network is **continuous** and **piecewise linear**.

Gradient computation

Definition (Jacobian matrix)

Let $f : \mathbb{R}^n \to \mathbb{R}^m$ a differentiable function. Its Jacobian $J_f(x) \in \mathbb{R}^{m \times n}$ is the matrix whose coordinates are the partial derivatives:

$$J_f(x) = \begin{bmatrix} \nabla f_1(x)^\top \\ \cdots \\ \nabla f_m(x)^\top \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \cdots & \cdots & \cdots \\ \frac{\partial f_m(x)}{\partial x_1} & \cdots & \frac{\partial f_m(x)}{\partial x_n} \end{bmatrix}$$

Gradient computation

Definition (Jacobian matrix)

Let $f : \mathbb{R}^n \to \mathbb{R}^m$ a differentiable function. Its Jacobian $J_f(x) \in \mathbb{R}^{m\times n}$ is the matrix whose coordinates are the partial derivatives:

$$J_f(x) = \begin{bmatrix} \nabla f_1(x)^\top \\ \cdots \\ \nabla f_m(x)^\top \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \cdots & \cdots & \cdots \\ \frac{\partial f_m(x)}{\partial x_1} & \cdots & \frac{\partial f_m(x)}{\partial x_n} \end{bmatrix}$$

Lemma (Jacobian of MLPs)

The Jacobian of an MLP g_θ is

$$J_{g_\theta}(x) = W^{(L)} D^{(L-1)} W^{(L-1)} D^{(L-2)} \dots W^{(2)} D^{(1)} W^{(1)}$$

where $D^{(l)} = \mathrm{diag}(\sigma'(g_\theta^{(2l-1)}(x)))$ and $g_\theta^{(2l-1)}(x)$ is the input of the l^{th} activation.

Class overview