

# IFT6135 – Apprentissage de Représentations (Deep Learning)

F. Wilhelmy  
fwilhelmy@hotmail.com

Automne 2024

IFT6135-H25  
Notes du cours – Hivers 2022

Montreal, QC

# Introduction

These are my personal notes from the course IFT6135 — Representation Learning — taught by Aaron Courville at MILA in Montreal, Quebec, during the Winter 2025 semester. The course ran from January to April, and it was a deep (pun intended) dive into the theory and practice of learning useful representations from data.

That said — fair warning! These notes are incomplete, sometimes messy (I often switch between French and English), and occasionally wrong. They reflect my own understanding of the material, and I've tried my best to make sense of complex ideas. Some parts were generated or cleaned up with the help of ChatGPT, others go slightly beyond what was taught in class. There are definitely missing citations, and I still plan to add more figures and references.

I'm sharing these in case they help others going through this intense (but rewarding!) course. If you spot errors, have suggestions, or want to expand a section, feel free to fork the repo and open a pull request. I'll do my best to review changes promptly. If you do contribute, please don't forget to add your name to the author list!

A sincere thank you to Professor Aaron Courville for his inspiring lectures, to the TAs for crafting such challenging and thought-provoking assignments, and to all the students — especially Thomas, Maël and Olivier — whose discussions, questions, and collaboration made the learning process far more engaging. It was a real privilege to be part of such a passionate and driven group.

And for those reading this outside the context of MILA — welcome! You'll probably get the most out of these notes if you already have some background in math, computer science, and deep learning. A solid starting point is the Deep Learning book: <https://www.deeplearningbook.org/>.

Enjoy the ride — and good luck with your learning journey!

# Contents

<b>1</b>	<b>History</b>	<b>8</b>
<b>2</b>	<b>Fundamental Concepts</b>	<b>10</b>
2.1	Définition de l'apprentissage machine . . . . .	10
2.2	Le Perceptron . . . . .	10
2.3	Conception d'un Modèle . . . . .	11
2.3.1	Le Modèle . . . . .	11
2.3.2	L'Optimisation . . . . .	12
2.4	Types de Problèmes . . . . .	12
2.5	Types de Modèles . . . . .	12
2.6	Inductive Bias in Machine Learning . . . . .	13
2.7	Bias-Variance Tradeoff . . . . .	14
<b>3</b>	<b>Optimization Methods</b>	<b>15</b>
3.1	Optimizers . . . . .	15
3.1.1	Stochastic Gradient Descent (SGD) . . . . .	15
3.1.2	Momentum . . . . .	15
3.1.3	Nesterov Accelerated Gradient (NAG) . . . . .	16
3.1.4	ADAGRAD (Adaptive Gradient) . . . . .	16
3.1.5	RMSPROP . . . . .	16
3.1.6	ADAM (Adaptive Moment Estimation) . . . . .	17
3.1.7	ADAMW . . . . .	17
3.1.8	AMSGRAD . . . . .	17
3.1.9	ADABOUND . . . . .	17
3.2	Normalization . . . . .	17
3.2.1	Batch Normalization . . . . .	18

3.2.2	Layer Normalization . . . . .	18
3.2.3	Instance Normalization . . . . .	18
3.2.4	RMS Normalization . . . . .	19
3.2.5	Group Normalization . . . . .	19
3.2.6	Weight Normalization . . . . .	19
<b>4</b>	<b>Regularization</b>	<b>20</b>
4.1	L2 Regularization . . . . .	20
4.2	L1 Regularization . . . . .	20
4.3	Early Stopping . . . . .	20
4.3.1	Variants . . . . .	20
4.4	Dropout Training . . . . .	20
4.5	Stochastic Depth . . . . .	20
4.6	Transfer Learning . . . . .	20
4.7	Multi-Task Learning . . . . .	21
4.8	Label Smoothing . . . . .	21
4.9	Data Augmentation . . . . .	21
4.9.1	Data Augmentation for Images . . . . .	21
<b>5</b>	<b>Convolutional Neural Networks (CNNs)</b>	<b>23</b>
5.1	Key Features . . . . .	23
5.2	Convolution Layer . . . . .	23
5.3	Pooling Layer . . . . .	23
5.4	Stride and Padding . . . . .	24
5.5	Receptive Field . . . . .	24
5.6	Backpropagation . . . . .	24
5.7	CNN Variants . . . . .	25

5.7.1	ResNet (Residual Networks) . . . . .	25
5.7.2	SENet (Squeeze-and-Excitation Networks) . . . . .	25
5.7.3	Dilated Convolutions . . . . .	25
5.7.4	EfficientNet . . . . .	25
5.7.5	ConvNeXt . . . . .	25
<b>6</b>	<b>Recurrent Neural Networks (RNNs)</b>	<b>26</b>
<b>7</b>	<b>Transformers</b>	<b>27</b>
7.1	Attention Mechanism . . . . .	27
7.2	Temperature in Transformers . . . . .	27
7.3	Transformer Variants . . . . .	28
7.3.1	GPT (Generative Pre-trained Transformer) . . . . .	28
7.3.2	BERT (Bidirectional Encoder Representations from Transformers) . . . . .	28
7.3.3	ELMo (Embeddings from Language Models) . . . . .	29
7.3.4	Transformer-XL . . . . .	29
7.3.5	T5 (Text-to-Text Transfer Transformer) . . . . .	30
<b>8</b>	<b>Variational Auto-Encoders (VAEs)</b>	<b>31</b>
8.1	Manifold Hypothesis . . . . .	31
8.2	Latent Variable Representation . . . . .	31
8.2.1	Maximum Likelihood . . . . .	31
8.3	The Variational Objective . . . . .	31
8.3.1	ECLL (Expected Complete-Data Log-Likelihood) . . . . .	31
8.3.2	KL Divergence . . . . .	32
8.3.3	ELBO . . . . .	32
8.4	Backpropagation . . . . .	32
8.4.1	The Reparameterization Trick . . . . .	32

8.5	VAE Variants . . . . .	32
8.5.1	VAE-IAF . . . . .	32
8.5.2	IWAE (Weighted Importances) . . . . .	32
8.5.3	SVG / VRNN . . . . .	33
<b>9</b>	<b>Diffusion models</b>	<b>34</b>
9.1	Controlling Diffusion models (Part 2) . . . . .	34
<b>10</b>	<b>Reinforcement Learning</b>	<b>35</b>
10.1	Reward Hacking . . . . .	35
10.2	Markov Decision Process . . . . .	35
10.3	Policies . . . . .	35
10.4	Bellman Equations . . . . .	35
10.5	Value-Based RL . . . . .	35
10.5.1	Goal . . . . .	35
10.5.2	Policy . . . . .	35
10.5.3	Q-Learning . . . . .	35
10.5.4	Deep Q-Learning . . . . .	35
10.6	Policy-Based RL . . . . .	35
10.6.1	Goal . . . . .	35
10.6.2	Policy . . . . .	35
10.7	Actor-Critic . . . . .	36
10.7.1	Policy . . . . .	36
<b>11</b>	<b>Reinforcement Learning from Human Feedback (RLHF)</b>	<b>37</b>
11.1	Flan-T5 . . . . .	37
11.2	Limitations of Finetuning . . . . .	37
11.3	Human-in-the-loop . . . . .	37

11.4 Reward . . . . .	37
11.5 InstructGPT . . . . .	38
11.6 LLM-as-a-judge . . . . .	38
11.7 Benchmarks and comparisons . . . . .	38
11.7.1 MT Bench . . . . .	38
<b>12 Self-Supervised Learning</b>	<b>39</b>
12.1 Pretext Task Paradigm . . . . .	39
12.2 Spatial Context and Structural Tasks . . . . .	39
12.2.1 Context Prediction . . . . .	39
12.2.2 Jigsaw Puzzles . . . . .	39
12.2.3 Rotation Prediction . . . . .	39
12.3 Contrastive Learning and Beyond . . . . .	40
12.3.1 MoCo (Momentum Contrast) . . . . .	40
12.3.2 SimCLR (Simple Framework for Contrastive Learning) . . . . .	40
12.3.3 CLIP (Contrastive Language–Image Pre-training) . . . . .	40
12.3.4 BYOL (Bootstrap Your Own Latent) . . . . .	40
12.3.5 Barlow Twins . . . . .	41
12.3.6 DINO (Self-Distillation with No Labels) . . . . .	41
12.3.7 JEPA (Joint Embedding Predictive Architecture) and I-JEPA (Improved JEPA) . . . . .	41
12.4 Knowledge Distillation . . . . .	41

# 1 History

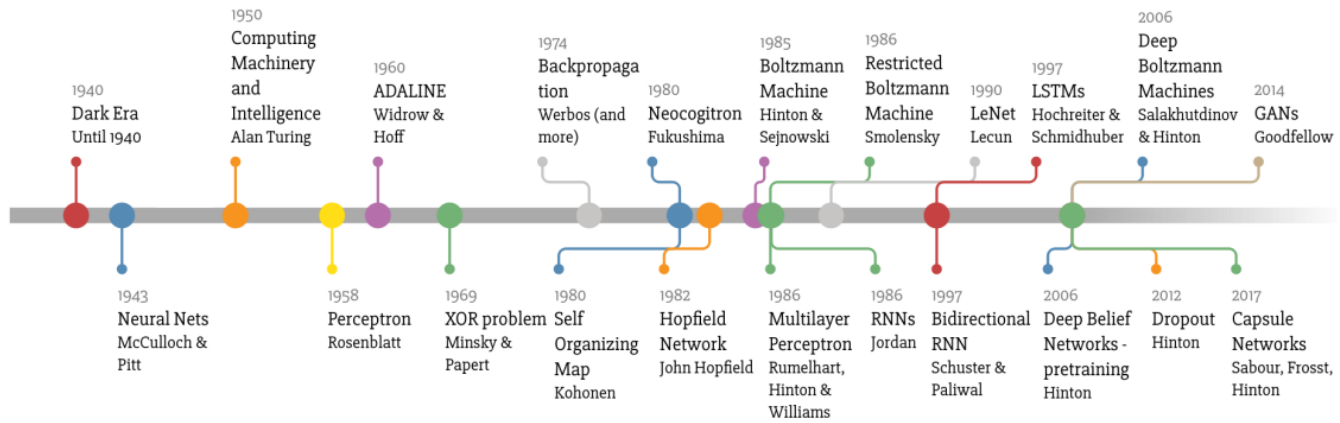


Figure 1: Contexte historique

**Naissance de l'IA (années 1950)** Le début du domaine de l'IA (intelligence artificielle) a eu lieu vers 1950.

- Perceptron de McCulloch et Pitts (modèle de neurone artificiel).
- Apprentissage de Rosenblatt (introduction du calcul de l'erreur).
- Théorie du calcul de Turing.
- Théorie de l'information de Shannon.

Le neurone artificiel est une inspiration de son homologue biologique. Il est composé d'entrées, de poids et d'une fonction d'activation. Initialement, il n'existait pas de mécanisme d'apprentissage. Rosenblatt a introduit un apprentissage basé sur le calcul de l'erreur, mais sans descente de gradient, limitant les réseaux à une seule couche.

**Premier hiver de l'IA (années 1960-1970)** Le problème du XOR soulevé par Minsky et Papert a mis en évidence une limitation majeure du perceptron, menant à un désintérêt général pour l'IA et à une réduction des investissements dans le domaine. C'est à cette époque que les Multi-Layer Perceptrons (MLP) ont été introduits, permettant de dépasser la contrainte de séparabilité linéaire des données. L'introduction de la **rétropropagation** a marqué une avancée majeure :

- **Efficace** : Complexité de  $O(N)$ , où  $N$  est le nombre d'exemples.
- **Générique** : Basé sur la descente de gradient, encore utilisé aujourd'hui.
- **Local** : Aucune garantie de trouver une solution optimale globale.

**Retour de l'IA (années 1990)** Dans les années 1990, l'intelligence artificielle a connu un renouveau avec l'apparition de nouveaux modèles :

- **LeNet** de Yann LeCun, utilisé pour la reconnaissance automatique de caractères.
- **Réseaux récurrents** développés par Hochreiter et Schmidhuber, permettant la représentation de séquences.



**Deuxième hiver de l'IA (années 2000)** Un nouvel hiver de l'IA est survenu, causé principalement par :

- Le problème du *vanishing gradient*, empêchant l'entraînement des réseaux de plus de 3-4 couches.
- Le manque de justification théorique des performances des réseaux de neurones.
- L'apparition des SVMs (machines à vecteurs de support) qui offraient des performances comparables avec moins de complexité et d'hyper-paramètres.

**Début de l'apprentissage profond (2006)** À partir de 2006, des méthodes ont été développées pour entraîner efficacement des réseaux de neurones profonds :

- Deep Boltzmann Machines.
- Deep Belief Networks.
- Pré-entraînement non supervisé et apprentissage par couche.

L'entraînement des autoencodeurs par couche était alors fastidieux, nécessitant des sous-entraînements successifs.

**Explosion de l'apprentissage profond (2012-2013)** L'année 2012 marque un tournant majeur avec l'introduction d'AlexNet (SuperVision) lors de la compétition ImageNet. Plusieurs innovations ont contribué à ce succès :

- Fonction d'activation **ReLU**.
- **Augmentation des données** pour améliorer la robustesse des modèles.
- **Réseaux convolutifs** pour extraire des caractéristiques hiérarchiques des images.
- **Utilisation des GPUs**, rendant l'entraînement des modèles plus rapide et efficace.
- Accès à de grandes bases de données comme ImageNet.

## Aujourd'hui

L'apprentissage profond est omniprésent et trouve des applications variées dans de nombreux domaines.

## 2 Fundamental Concepts

### 2.1 Définition de l'apprentissage machine

La programmation 'manuelle' est un processus/algorithme qui a comme entrée des données et des instructions (règles) et qui produit une sortie.

À l'inverse, l'apprentissage machine est un système où l'on fournit en entrée des données et la réponse attendue pour une certaine donnée. La sortie de ce système est un ensemble de règles (modèle) qui cherche à généraliser une relation entre les données et leur sortie respective.

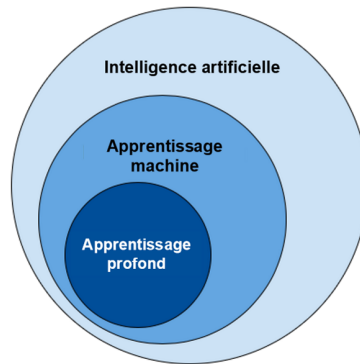


Figure 2: Terminologie de l'IA

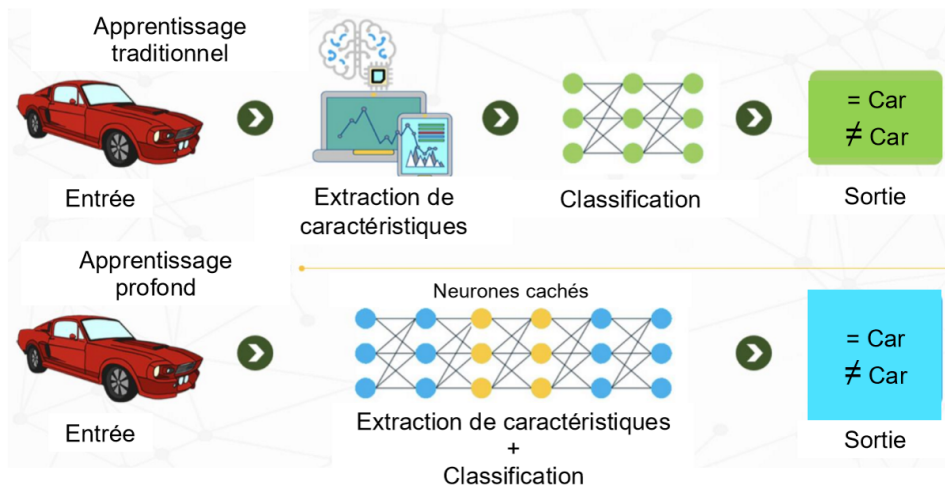


Figure 3: Différence entre l'apprentissage machine et l'apprentissage profond

### 2.2 Le Perceptron

Le perceptron est le bloc de construction de base dans les réseaux de neurones. Deux composants essentiels :

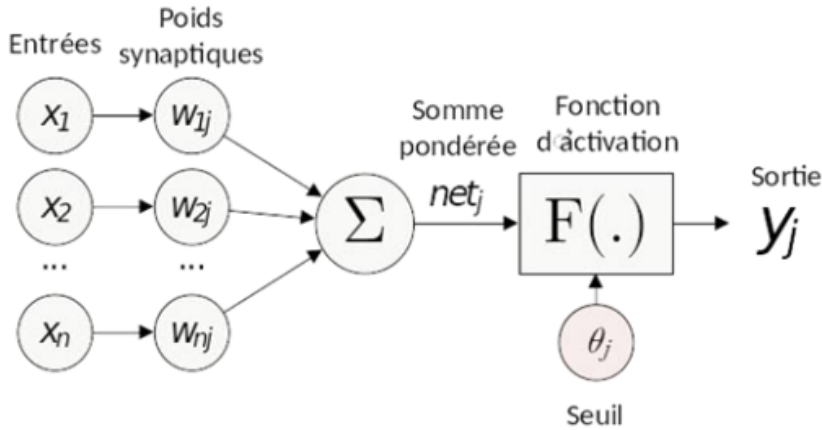
1. La somme pondérée ( $net_j$ ) :

$$net_j = \sum_{i=0}^n x_i w_{ij} = \vec{w}_j \cdot \vec{x} = \mathbf{w}_j \mathbf{x}$$

ou, en termes du produit scalaire et de l'angle  $\alpha$ ,

$$net_j = |\mathbf{w}_j| |\mathbf{x}| \cos(\alpha).$$

2. **La fonction d'activation ( $F$ )** : introduit la non-linéarité en activant ou désactivant le neurone selon l'agrégation pondérée des entrées.



$$net_j = \sum_{i=1}^n x_i w_{ij}$$

$$y_j = F(net_j, \theta_j)$$

Figure 4: Schéma d'un perceptron

Note importante : Une caractéristique importante du perceptron est qu'il est sensible à l'ordre des données d'entrée. Deux ensembles de données identiques mais présentés dans un ordre différent peuvent produire des ajustements de poids différents.

L'ajustement des poids par la correction de l'erreur a permis au perceptron d'apprendre, mais il n'était pas capable de résoudre certains problèmes comme le problème du XOR. Cette limitation a été surmontée avec l'introduction de méthodes d'optimisation plus avancées, telles que la descente de gradient. Cependant, le choix de la fonction d'activation dans ces méthodes est crucial pour assurer un apprentissage efficace.

## 2.3 Conception d'un Modèle

La conception d'un modèle d'apprentissage profond peut être décomposée en 4 étapes :

- Définition de la tâche et du dataset
- **Sélection du modèle**
- **Choix de la fonction de coût**
- Optimisation du modèle

### 2.3.1 Le Modèle

Un modèle efficace doit généraliser et représenter les données. Le choix dépend aussi des contraintes de déploiement (puissance de calcul, rapidité, etc.) et il peut être pertinent de se demander si l'utilisation de l'IA est nécessaire ou si un algorithme plus simple pourrait suffire.

### 2.3.2 L'Optimisation

Cette phase cherche la configuration optimale des paramètres dans l'espace de la fonction de coût. La validation permet d'éviter les problèmes de surapprentissage et de sous-apprentissage.

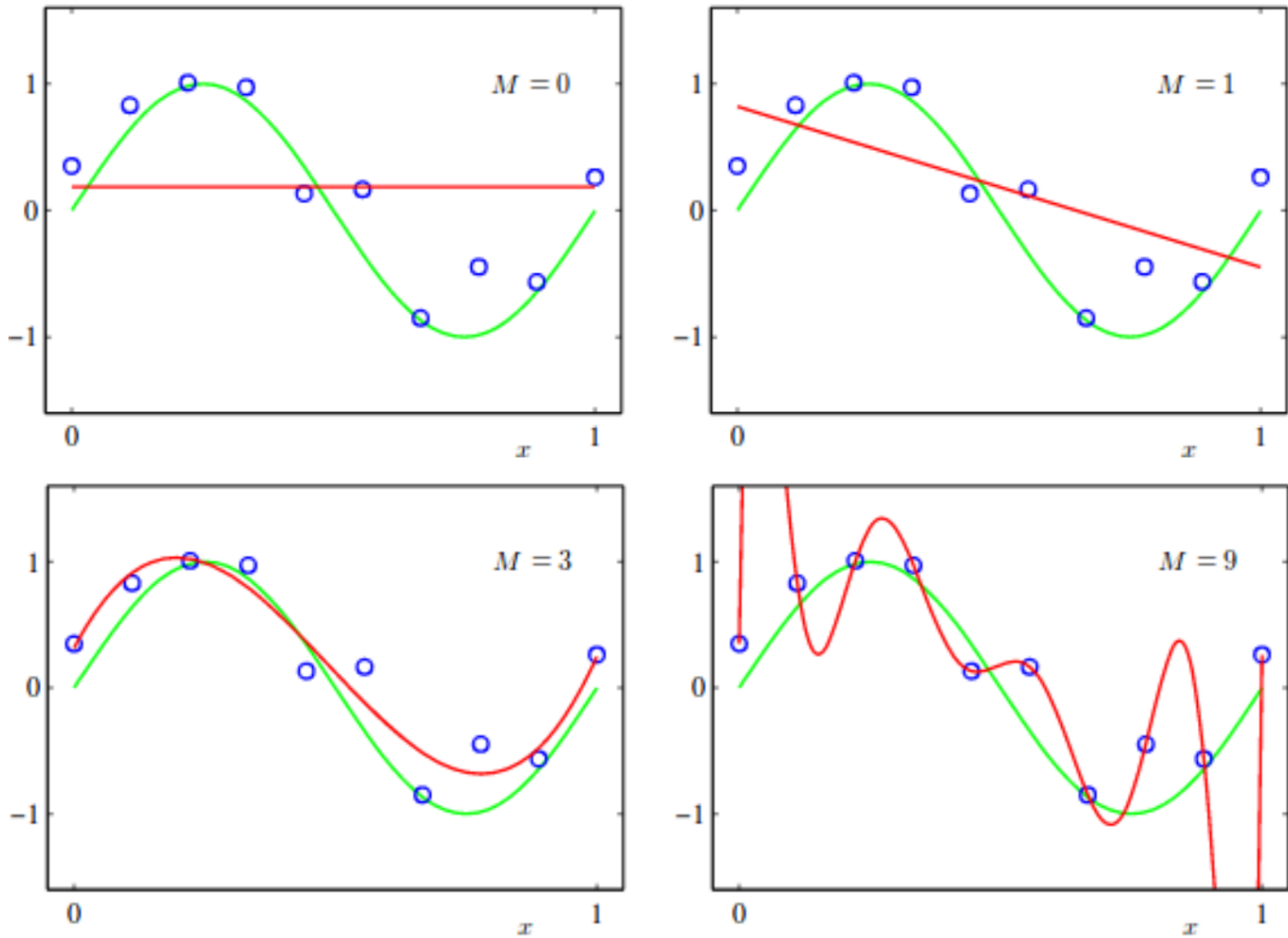


Figure 5: Exemple de surapprentissage (overfit) et de sous-apprentissage (underfit)

## 2.4 Types de Problèmes

- **Régression** :  $x \rightarrow \mathbb{R}$
- **Classification** :  $x \rightarrow \mathbb{N}$

## 2.5 Types de Modèles

Les approches de l'apprentissage machine se classent en différentes familles :

- **Supervisé** : Toutes les données sont étiquetées.
- **Non-supervisé** : Aucune étiquette n'est disponible.

- **Semi-supervisé** : Seule une partie des données est étiquetée.
- **Auto-supervisé** : Des étiquettes peu coûteuses sont générées pour des tâches auxiliaires.
- **Apprentissage par renforcement** :

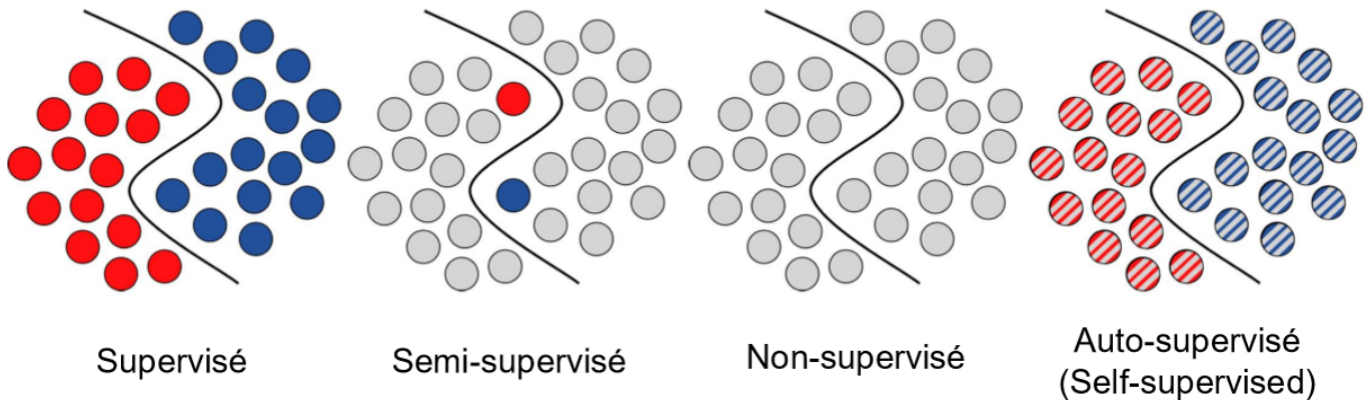


Figure 6: Différents types d'apprentissage

## 2.6 Inductive Bias in Machine Learning

Inductive bias refers to the assumptions a model makes to generalize from training data to unseen data. Different architectures (e.g., CNNs, RNNs, Transformers) incorporate unique biases that affect how they learn and represent information.

Given that real-world data is limited, an inductive bias enables a model to make reasonable predictions by imposing constraints on the possible functions it can learn.

Deep learning architectures have specific biases that give them particular strengths and limitations. For instance, CNNs exploit spatial locality, while Transformers capture long-range dependencies using attention mechanisms.

- **Smoothness Bias**: Similar inputs yield similar outputs.
- **Linear Separability**: Data can be separated by a linear boundary (used in logistic regression and SVMs).
- **Locality Bias**: Local structures are emphasized (as in CNNs).
- **Sequential Dependence**: Past information influences future predictions (as in RNNs).
- **Attention-Based Bias**: Certain input features are weighted more heavily (as in Transformers).
- **Manifold Hypothesis**:

Depending on the level of bias imposed on the model, we get different results.

- **Strong Inductive Bias**: Imposes strong assumptions for faster learning but reduced flexibility (e.g., linear regression).
- **Weak Inductive Bias**: Fewer assumptions allow greater flexibility but require more data (e.g., deep neural networks).

## 2.7 Bias-Variance Tradeoff

The overall error can be decomposed as:

$$\text{Total Error} = \text{Bias}^2 + \text{Variance} + \sigma^2,$$

with

$$\text{Bias}[\tilde{f}] = E[\tilde{f}] - f \quad \text{and} \quad \text{Variance}[\tilde{f}] = E[(\tilde{f} - E[\tilde{f}])^2].$$

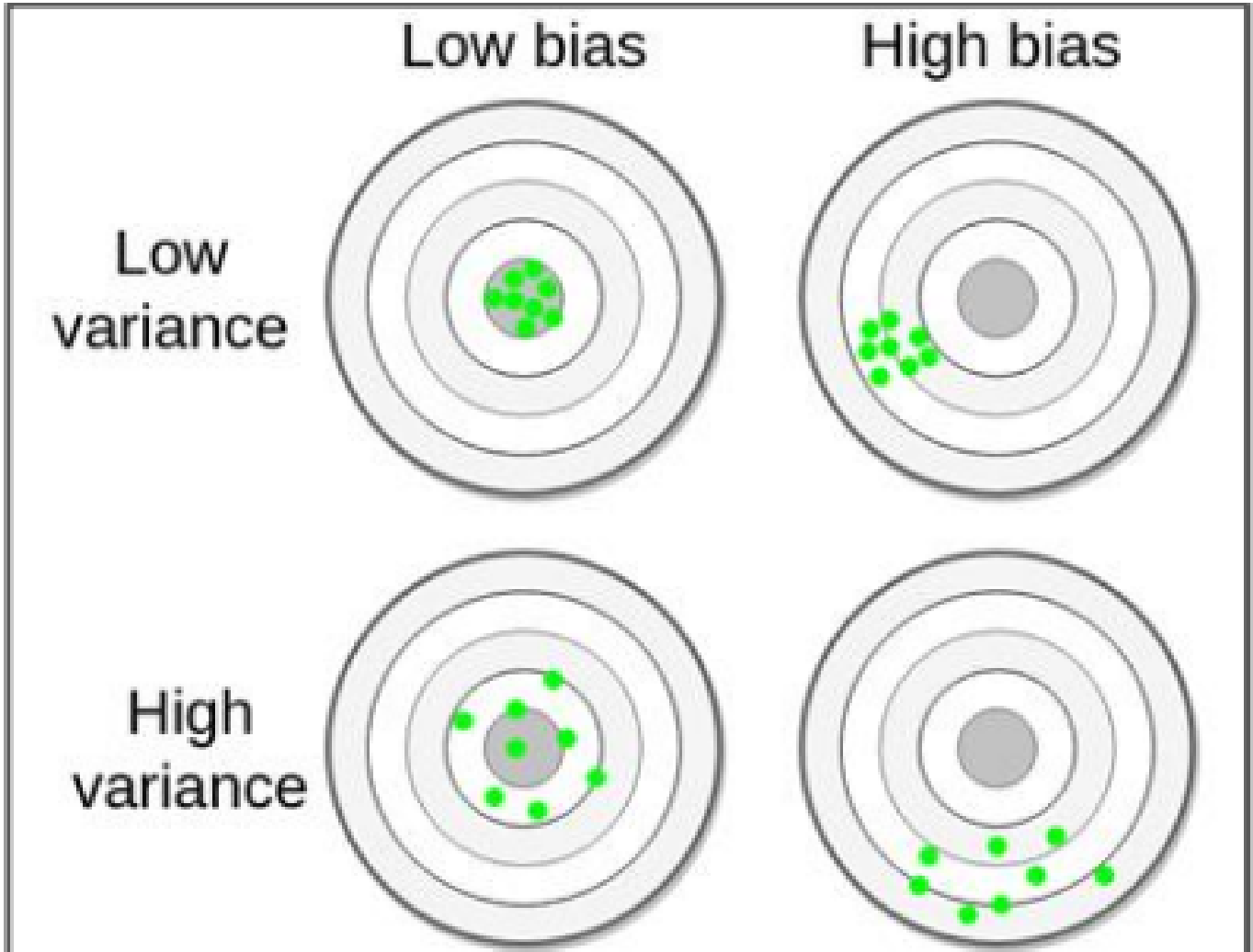


Figure 7: Bias versus Variance

## 3 Optimization Methods

Optimization methods search for the best parameter configuration by minimizing the loss function. Generally, optimization techniques will focus on improving the training accuracy of the model and its convergence speed. As opposed to regularisation techniques (that we will cover later that are more focused on the ability of the model to generalise). Therefore sometimes slowing the training. In other words, it will determine if it has found the minimum of our cost function (loss function).

- Evolutionary algorithms.
- Closed-form solutions.
- Gradient-based methods (first and second order).

In this section, we will mostly focus on first-order gradient-based methods.

### 3.1 Optimizers

#### 3.1.1 Stochastic Gradient Descent (SGD)

SGD updates parameters using mini-batches:

---

**Algorithm 1** Stochastic Gradient Descent

---

- 1: **Input:** Learning rate  $\eta$ , initial parameter  $\theta$
- 2: **while** Stopping criterion not met **do**
- 3:     Sample a mini-batch of  $m$  examples  $(x^{(i)}, y^{(i)})$
- 4:     Compute gradient estimate:

$$\hat{g} = \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} L(f(x^{(i)}; \theta), y^{(i)})$$

- 5:     Update parameters:

$$\theta \leftarrow \theta - \eta \hat{g}$$

- 6: **end while**
- 

#### 3.1.2 Momentum

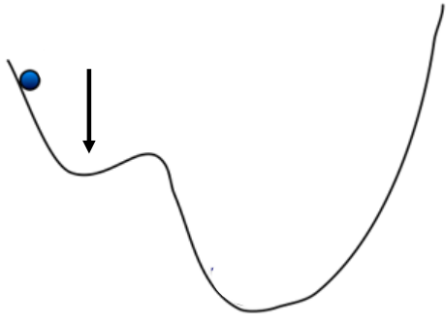
The **gradient with momentum** is an optimization method that improves the convergence of descent gradient methods by adding a notion of accumulation of past gradients. This helps to overcome oscillations and to accelerate convergence in directions where the gradient varies little. This method can also be combined with other optimization techniques, such as warm restart and adaptive learning rate adjustment.

The update of weights using the gradient with momentum is done according to the following rule :

$$\begin{aligned} v_{t+1} &= \rho v_t - \eta \nabla L(f(x; \theta_t), y) \\ \theta^{t+1} &= \theta^t + v_{t+1} \end{aligned}$$

where  $\rho$  is the momentum coefficient.

## Gradient Normal



## Gradient avec moment

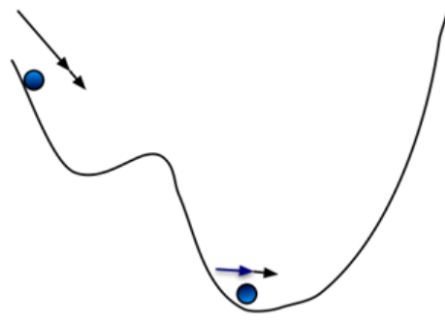


Figure 8: Gradient avec moment

### 3.1.3 Nesterov Accelerated Gradient (NAG)

Une variante du gradient avec moment est le **moment de Nesterov**, qui prévoit le déplacement avant de calculer le gradient. Cette approche permet d'anticiper le comportement de la fonction de perte et d'améliorer encore davantage la convergence. Elle aide à réduire les oscillations et à améliorer la vitesse d'apprentissage dans des zones planes de l'espace des solutions.

Nesterov Momentum refines the momentum method by first applying a partial update before computing the gradient, allowing for more accurate gradient adjustments.

$$\begin{aligned}\tilde{\theta} &= \theta + \alpha v, \\ g &= \nabla_{\theta} L(\tilde{\theta}), \\ v &\leftarrow \alpha v - \eta g, \quad \theta \leftarrow \theta + v.\end{aligned}$$

### 3.1.4 ADAGRAD (Adaptive Gradient)

Adagrad adapts the learning rate for each parameter for each parameter using accumulated squared gradients, making it suitable for convex problems but potentially slowing down over time due to accumulated gradients.

$$r \leftarrow r + g \odot g,$$

Here  $r$  accumulates past squared gradients element-wise.

$$\Delta\theta = \frac{\eta}{\sqrt{r} + \epsilon} \odot g,$$

Here  $\epsilon$  is a small constant for numerical stability.

$$\theta \leftarrow \theta - \Delta\theta.$$

### 3.1.5 RMSPROP

TODO



### 3.1.6 ADAM (Adaptive Moment Estimation)

This optimiser combines RMSPROP and momentum, computing adaptive learning rates for each parameter by using both first-order moment estimates (momentum) and second-order moment estimates (variance).

We compute the first moment estimate  $m$  (moving average of gradients) and the second moment estimate  $v$  (moving average of squared gradients) as

$$m \leftarrow \beta_1 m + (1 - \beta_1)g, \quad v \leftarrow \beta_2 v + (1 - \beta_2)g \odot g,$$

Where  $\beta_*$  is the decay rate.

$$\hat{m} = \frac{m}{1 - \beta_1^t}, \quad \hat{v} = \frac{v}{1 - \beta_2^t},$$

The bias correction terms  $\hat{m}$  and  $\hat{v}$  are then used to adjust for the initialization bias.

$$\Delta\theta = \frac{\eta}{\sqrt{\hat{v}} + \epsilon} \odot \hat{m}, \quad \theta \leftarrow \theta - \Delta\theta.$$

Adam is widely used due to its robustness and ability to handle sparse gradients.

### 3.1.7 ADAMW

The successor of ADAM, which is also very popular. This method decouples weight decay from gradient updates, improving generalization.

### 3.1.8 AMSGRAD

This optimiser ensures non-increasing second moment estimates to address convergence issues in ADAM.

### 3.1.9 ADABOUND

This optimiser dynamically bounds the learning rate to transition smoothly between adaptive methods and SGD.

## 3.2 Normalization

Normalization accelerates training and stabilizes gradient flow by standardizing activations.

Normalizing inputs (or activations) helps reduce the impact of differing value ranges, smoothing the loss landscape and improving gradient flow.

As we can see in figure 11, the landscape of normalized data will be much easier to traverse during training. Without it, there are some locations that could be complicated to get out of and some points are much further away than others. In the normalized case, everything is "relatively" close.

This will also make initialization less critical and leads to faster convergence from various starting points.

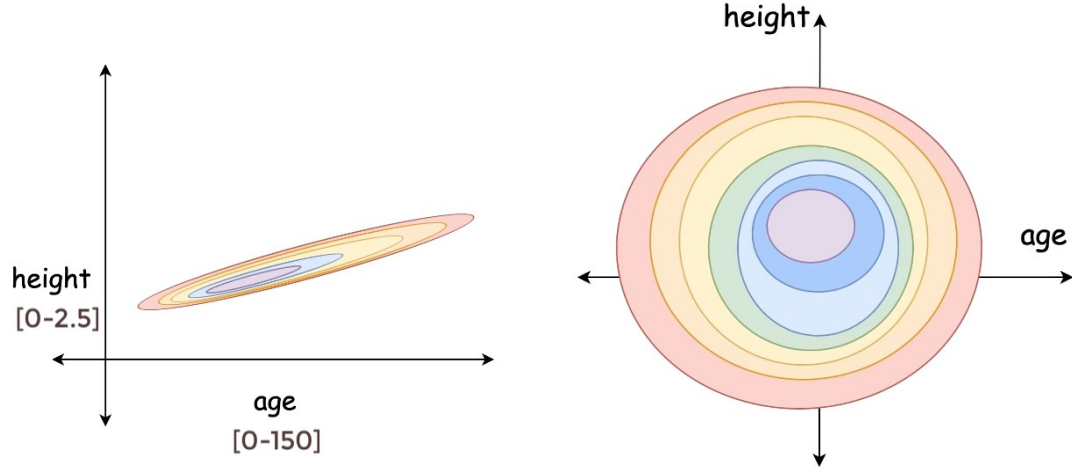


Figure 9: Gradient landscape without and with normalization

Once the parameters are normalised, the learned parameters  $\gamma$  and  $\beta$  are applied:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}, \quad y_i = \gamma \hat{x}_i + \beta.$$

### 3.2.1 Batch Normalization

**Reference:** Ioffe and Szegedy (ICML 2015)

For a mini-batch of size  $B$  with  $F$  features per channel, we normalize over each channels in all samples

$$\mu_c = \frac{1}{B \times F} \sum_{b,f} x_{bcf}, \quad \sigma_c^2 = \frac{1}{B \times F} \sum_{b,f} (x_{bcf} - \mu_c)^2.$$

### 3.2.2 Layer Normalization

**Reference:** Ba, Kiros, and Hinton

Normalizes over all features for each sample

$$\mu_b = \frac{1}{C \times F} \sum_{c,f} x_{bcf}, \quad \sigma_b^2 = \frac{1}{C \times F} \sum_{c,f} (x_{bcf} - \mu_b)^2.$$

### 3.2.3 Instance Normalization

Normalizes each channel per sample

$$\mu_{bc} = \frac{1}{F} \sum_f x_{bcf}, \quad \sigma_{bc}^2 = \frac{1}{F} \sum_f (x_{bcf} - \mu_{bc})^2.$$

### 3.2.4 RMS Normalization

Normalizes using the root mean square

$$\mu_b^2 = \frac{1}{C \times F} \sum_{c,f} x_{bcf}^2, \quad \hat{x}_{bcf} = \frac{x_{bcf}}{\sqrt{\mu_b^2 + \epsilon}}.$$

### 3.2.5 Group Normalization

Divides channels into  $G$  groups and normalizes each group separately.

### 3.2.6 Weight Normalization

TODO

## 4 Regularization

A central problem in machine learning is how to make an algorithm that will perform well not just on the training data, but also on new inputs. Many strategies used in machine learning are explicitly designed to reduce the test error, possibly at the expense of increased training error. These strategies are known collectively as regularization. A great many forms of regularization are available to the deep learning practitioner.

### 4.1 L2 Regularization

TODO..

### 4.2 L1 Regularization

TODO..

### 4.3 Early Stopping

This algorithm allows us to stop the training when we see the performance on the validation set reducing.

#### 4.3.1 Variants

There exist techniques that are not very used recently but are still worth mentioning such as

- **Early Stopping with Retraining:** This technique is made to not waste certain example in the validation set and uses validated examples for a minor performance boost. It's not very popular because it's not the goal to grab this little performance gain and the data base are very big now.
- **Early Stopping with Surrogate Loss:** Employs an alternate loss function.

### 4.4 Dropout Training

Randomly disables certain connections during training to reduce overfitting.

### 4.5 Stochastic Depth

Randomly drop entire layers during training, particularly effective in ResNets.

### 4.6 Transfer Learning

Train a general model and fine-tune it for specific tasks.

## 4.7 Multi-Task Learning

Build a general model and plug in smaller task-specific modules.

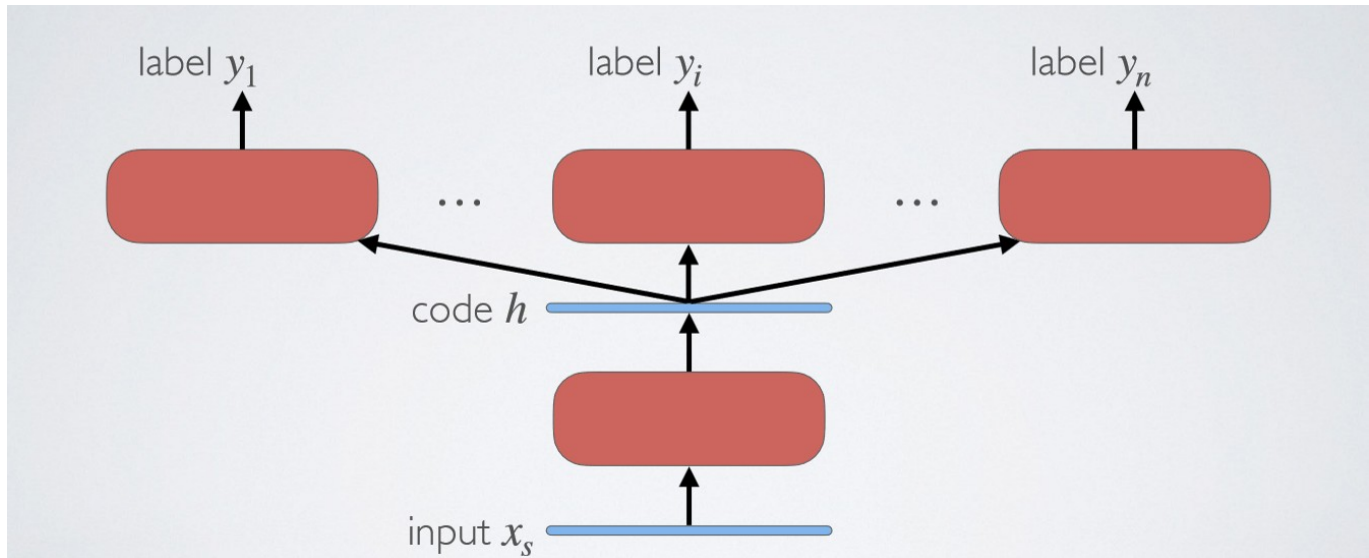


Figure 10: Multi-Task Learning architecture

## 4.8 Label Smoothing

Replace hard labels (0 and 1) with smoothed values to improve generalization. This results in a cluster of labels together.

## 4.9 Data Augmentation

Increase dataset size by applying transformations (translation, rotation, etc.). This technique had a HUGE impact on the image, audio, video and other tasks with those type of data but NONE to others like text because there is no way found so far to augment text.

### 4.9.1 Data Augmentation for Images

**RandAugment** This technique is among the most popular. In general, it applies transformations such as: translation, rotation, scaling, equalization, posterization and solarization, etc...

However, more aggressive augmentation techniques exist..

**MixUp** Blends two images and their labels. This technique generally yields low performance.

**Random Erasing / Cutout** Removes a random portion of the image.

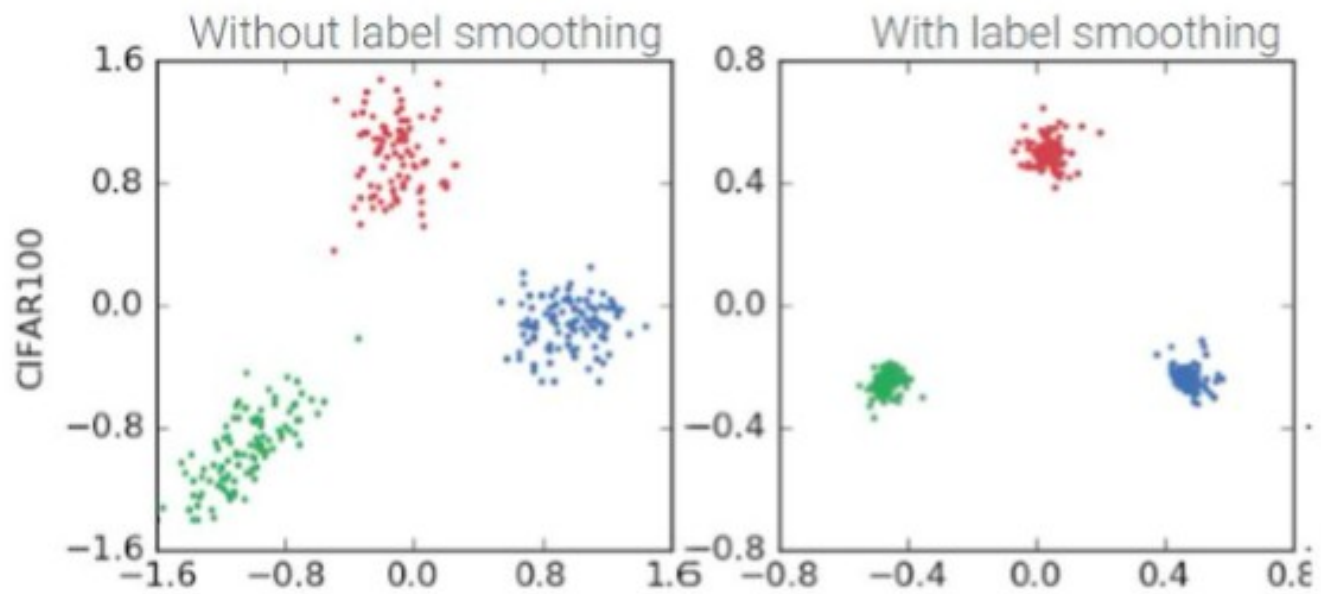


Figure 11: Source: When does label smoothing help?

**CutMix** Combines regions from different images with proportional labels.

## 5 Convolutional Neural Networks (CNNs)

A Convolutional Neural Network (CNN) is a type of neural network designed specifically for pattern recognition in image or audio data (mimicking the way the visual cortex processes visual information). CNNs leverage convolution operations to extract hierarchical features from images, efficiently handling high-dimensional data while exploiting spatial structures, such as the relationship between nearby pixels. This architecture builds invariance to transformations like translation, making it highly effective for image-related tasks. A typical CNN consists of alternating convolutional and pooling layers, followed by fully connected layers.

### 5.1 Key Features

**Local Connectivity** Each neuron connects only to a local region of the input, called the *receptive field*. This allows CNNs to focus on localized patterns (e.g., edges, textures), reducing the number of parameters compared to fully connected networks (FFNs).

**Parameter Sharing** The same set of weights (kernel) is used across different spatial locations of the input, significantly reducing computational complexity.

**Pooling (Dimensionality Reduction)** Reduces the number of hidden units in the model and helps achieve translational invariance, improving generalization.

For an image  $x$ , a kernel  $k$ , and output  $y$ :

$$(y * k)_{i,j} = \sum_{p,q} x_{i+p,j+q} \cdot k_{r-p,r-q}.$$

### 5.2 Convolution Layer

For an input of size  $N \times N \times D_{in}$ , kernel of size  $K \times K$ , stride  $S$ , padding  $P$ , and  $D_{out}$  filters:

$$N_{out} = \left\lfloor \frac{N - K + 2P}{S} \right\rfloor + 1,$$

with total parameters:

$$\text{Total Parameters} = D_{out} \times (K \times K \times D_{in} + (\text{bias term})).$$

### 5.3 Pooling Layer

For input of size  $N \times N \times D$ , pooling window  $F \times F$ , and stride  $S$ :

$$N_{out} = \left\lfloor \frac{N - F}{S} \right\rfloor + 1,$$

with zero trainable parameters. There are two main types of pooling

- **Max Pooling:**

$$y_{i,j} = \max_{(p,q) \in R} x_{i+p,j+q}.$$

- **Average Pooling:**

$$y_{i,j} = \frac{1}{|R|} \sum_{(p,q) \in R} x_{i+p,j+q}.$$

## 5.4 Stride and Padding

**Stride** Controls the step size of the convolutional filter when moving across the input.

**Padding** Adds extra pixels around the input to control output size. Special cases include:

- **Valid Convolution:** No padding, reducing spatial dimensions.
- **Same Convolution (Half Padding):** Preserves input dimensions.
- **Full Convolution:** Pads the input to ensure maximum overlap.

## 5.5 Receptive Field

The *receptive field* of a neuron in a convolutional neural network (CNN) is the region of the input that influences that neuron's output. As you progress deeper into the network, the receptive field generally increases, allowing the network to capture larger context. Understanding the receptive field is essential because it helps you design architectures that capture enough context from the input.

In simpler terms, it is the size of the patch in the input image that contributes to a particular feature in a deeper layer. For layer  $i$  with kernel  $k_i$  and stride  $s_i$ , the receptive field can be computed recursively

$$r_i = r_{i-1} + (k_i - 1) \times S_{i-1},$$

where  $S_{i-1} = \prod_{j=1}^{i-1} s_j$  is the effective stride. Also, for the first layer applied directly on the input, the receptive field is simply

$$r_1 = k_1.$$

### Example

- Layer 1:  $k_1 = 3, s_1 = 1 \Rightarrow r_1 = 3$ .
- Layer 2:  $k_2 = 3, s_2 = 2 \Rightarrow r_2 = 3 + 2 \times 1 = 5$ .

## 5.6 Backpropagation

TODO



## 5.7 CNN Variants

### 5.7.1 ResNet (Residual Networks)

Deep networks suffer from the *vanishing gradient problem*, where gradients become too small to update weights effectively. ResNet introduces skip connections, allowing layers to learn residual mappings:

$$F(x) = H(x) - x$$

where  $H(x)$  is the desired mapping.

This technique enables the training of very deep networks (e.g., ResNet-50, ResNet-152).

There are two type of residual blocks

**Basic Block**    TODO

**Bottleneck Block**    TODO

### 5.7.2 SENet (Squeeze-and-Excitation Networks)

SENet introduces a channel-wise attention mechanism.

1. **Squeeze:** Global average pooling condenses spatial information.
2. **Excitation:** A small fully connected network learns per-channel weights.
3. **Recalibration:** Multiplies channel-wise weights with the input feature map.

This enhances feature representation with minimal computational cost.

### 5.7.3 Dilated Convolutions

Useful when spatial redundancy is high. Expands the receptive field without increasing computational cost.

### 5.7.4 EfficientNet

*TODO: Explanation needed.*

### 5.7.5 ConvNeXt

*TODO: Explanation needed.*

## 6 Recurrent Neural Networks (RNNs)

*TODO: Explanation needed.*

## 7 Transformers

Transformers are effective for language modeling due to their ability to process data in parallel and capture long-range dependencies through attention.

Here are some key points when fine-tuning this type of model:

- The dimension of attention keys is critical.
- Larger models generally achieve better performance.
- Dropout is beneficial.
- Positional encodings (sinusoidal or learned) produce similar results.

### 7.1 Attention Mechanism

*TODO: This concept needs a dedicated sub section to clearly explain it*

### 7.2 Temperature in Transformers

Temperature is a scaling factor used to control the probability distribution in the self-attention mechanism and during text generation. It modulates the model's "confidence" in its predictions by adjusting the sharpness of the distributions. Temperature scales the softmax in both self-attention and text generation:

#### Self-Attention

$$e_{ij} = \frac{Q_i \cdot K_j^T}{T \sqrt{d_k}},$$

where  $T$  adjusts the sharpness.

#### Text Generation

$$P(y) = \text{softmax} \left( \frac{z}{T} \right).$$

In other words, the parameter  $T$  influences the attention distribution:

- **Low Temperature** ( $T < 1$ ): Results in a sharper distribution, concentrating more strongly on a few tokens with high scores.
- **High Temperature** ( $T > 1$ ): Produces a more diffuse distribution, allowing broader but less pronounced attention.

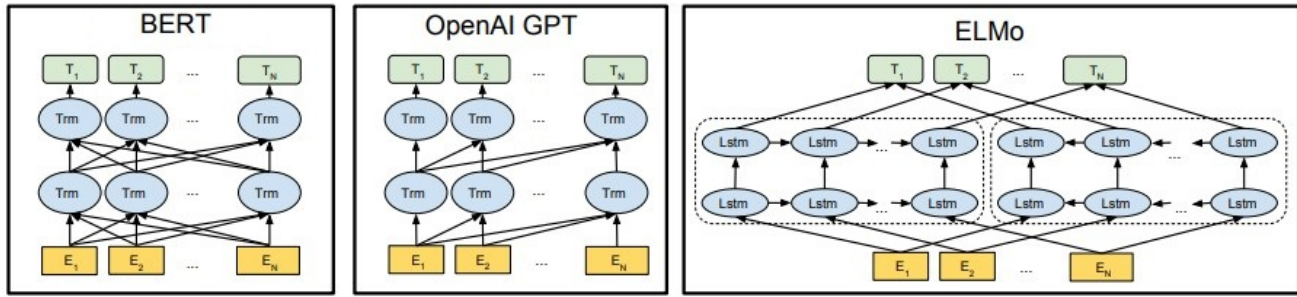


Figure 12: Architecture des Transformers. Source : *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* (Devlin et al.)

## 7.3 Transformer Variants

### 7.3.1 GPT (Generative Pre-trained Transformer)

GPT est un modèle basé sur un décodeur Transformer unidirectionnel. Il est entraîné de manière autoregressive pour prédire le token suivant dans une séquence. Son principal atout est la capacité à générer du texte de façon cohérente et fluide.

Les versions plus récentes (GPT-2, GPT-3) utilisent des contextes de taille fixe mais augmentent en taille de modèle pour améliorer la qualité de génération.

**GPT3** Emerging abilities - Zero-shot & Few-shot Prompting - Scale-up at some point make the performance explode

### 7.3.2 BERT (Bidirectional Encoder Representations from Transformers)

BERT (Bidirectional Encoder Representations from Transformers) is a Transformer-based model introduced by Google AI [?]. It revolutionized NLP by introducing a powerful bidirectional language representation model. It is designed to pre-train deep bidirectional representations by using self-supervised learning, allowing it to achieve state-of-the-art performance on a variety of natural language processing (NLP) tasks.

Ce modèle est pré-entraîné à l'aide de la tâche de *Masked Language Modeling* (MLM) ainsi que de la prédiction de la prochaine phrase (NSP).

**Masked Language Modeling (MLM)** BERT is pre-trained using a self-supervised learning approach called *Masked Language Modeling* (MLM). In this method:

- 15% of the input tokens are randomly selected for masking.
- Of these, 80% are replaced with a special [MASK] token, 10% are replaced with a random token, and 10% remain unchanged.
- The model is trained to predict the original tokens based on their surrounding context.

This forces BERT to develop a deep bidirectional understanding of text.

**Next Sentence Prediction (NSP)** To enhance sentence-level understanding, BERT also uses the *Next Sentence Prediction* (NSP) task:

- The model is given two sentences: A and B.
- It predicts whether B is the actual next sentence following A or a randomly selected sentence.

This task helps BERT learn sentence relationships, which is useful for tasks like question answering and natural language inference.

**Fine-tuning** After pre-training, BERT can be fine-tuned for specific NLP applications by adding task-specific layers on top of the Transformer encoders. Common applications include:

- **Text Classification:** Sentiment analysis, spam detection, topic classification.
- **Named Entity Recognition (NER):** Identifying people, locations, and organizations in text.
- **Question Answering (QA):** Extracting answers from passages, such as in the SQuAD dataset.
- **Text Summarization:** Generating concise versions of longer documents.

Several improved versions of BERT have been introduced over the years.

**RoBERTa** Removes NSP and trains with larger datasets and longer sequences.

**ALBERT** Reduces parameter size using factorized embeddings and cross-layer parameter sharing.

**DistilBERT** A smaller, faster, and more efficient variant of BERT with fewer layers.

That said, at the moment there isn't much followup on that model as all research are going toward decoder only style architectures like ChatGPT and company. That said, Prof. Aaron mentioned that if he had to bet what is the futur for LLMs in long term, he thinks BERT is because of it's much better performance for same model size.

### 7.3.3 ELMo (Embeddings from Language Models)

ELMo est basé sur des réseaux LSTM bidirectionnels plutôt que sur des Transformers. Il génère des embeddings contextuels pour chaque token en prenant en compte l'ensemble de la phrase. Bien qu'il ait été révolutionnaire pour fournir des représentations riches en contexte, il est moins efficace pour traiter de très longues séquences par rapport aux modèles Transformer modernes.

### 7.3.4 Transformer-XL

Transformer-XL étend le modèle autoregressif en introduisant une récursion au niveau des segments, ce qui permet de réutiliser les états cachés d'un segment à l'autre. Ce mécanisme, associé aux embeddings positionnels relatifs, permet de traiter des contextes beaucoup plus longs que les Transformers standards.

### 7.3.5 T5 (Text-to-Text Transfer Transformer)

T5 adopte une approche encodeur-décodeur en formulant toutes les tâches NLP comme un problème de conversion de texte en texte. Il est particulièrement efficace pour des tâches telles que la traduction, le résumé et la question-réponse. Sa flexibilité en fait un modèle très pertinent dans de nombreux contextes d'applications actuels.

## 8 Variational Auto-Encoders (VAEs)

This section introduces the central concepts of Variational Auto-Encoders (VAEs). We discuss how high-dimensional data is modeled using a low-dimensional latent space, derive a tractable variational objective, and present the methods that enable end-to-end learning in these models.

### 8.1 Manifold Hypothesis

In a machine learning context, the manifold hypothesis is the idea that although real world data (like image, audio or text) may exist in a high-dimensional space, the meaningful variations in the data are actually confined to a much lower-dimensional plane or manifold embedded within that high dimensional space. In other words, while each data point is represented by a large number of features, the true degree of freedom (the intrinsic dimensions) are far fewer... There is a limited number of possible values the data points can actually take.

### 8.2 Latent Variable Representation

We represent data  $x$  using latent variables  $z$  via a deterministic mapping:

$$x = g(z), \quad z \sim \mathcal{N}(0, I)$$

Here,  $z$  encapsulates the underlying factors of variation, and the decoder  $g$  reconstructs the high-dimensional data from  $z$ .

#### 8.2.1 Maximum Likelihood

The goal would then be to maximise a mapping such that

$$\log p(x) = \log \int p(x|z)p(z) dz$$

However, the integral over  $z$  is generally intractable.

### 8.3 The Variational Objective

To bypass the intractability of directly computing  $\log p(x)$ , VAEs optimize an approximated lower bound on the log-likelihood.

#### 8.3.1 ELL (Expected Complete-Data Log-Likelihood)

An intermediate formulation approximates the complete-data log-likelihood:

$$\mathbb{E}_{q(z|x)}[\log p(x, z)]$$

This expectation serves as a proxy by averaging over the latent variable distribution.

### 8.3.2 KL Divergence

The KL divergence measures the difference between the approximate posterior and the prior:

$$D_{\text{KL}}(q(z|x)||p(z|x)) = - \int q(z|x) \log \frac{q(z|x)}{p(z|x)} dz$$

This term regularizes the latent space, ensuring that  $q(z|x)$  remains close to the simple prior  $p(z|x)$ .

### 8.3.3 ELBO

The Evidence Lower Bound (ELBO) is defined as:

$$\text{ELBO} = \mathbb{E}_{q(z|x)}[\log p(x|z)] - D_{\text{KL}}(q(z|x)||p(z))$$

Maximizing the ELBO leads to a practical training objective that both reconstructs  $x$  and regularizes  $q(z|x)$ .

## 8.4 Backpropagation

The entire VAE, including both encoder and decoder, is trained end-to-end via backpropagation using gradients derived from the variational objective.

### 8.4.1 The Reparameterization Trick

To enable gradient flow through sampling, we reparameterize  $z$  as:

$$z = \mu(x) + \sigma(x)\epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

This neat transformation allows us to treat the random sampling as a differentiable operation.

## 8.5 VAE Variants

### 8.5.1 VAE-IAF

To achieve a richer posterior, we apply a series of invertible transformations:

$$z_T = f_T \circ \dots \circ f_1(z_0)$$

Inverse Autoregressive Flow (IAF) enables the approximate posterior to capture complex dependencies.

### 8.5.2 IWAE (Weighted Importances)

An improved variational bound is obtained by importance weighting:

$$\log p(x) \approx \log \frac{1}{K} \sum_{k=1}^K \frac{p(x, z_k)}{q(z_k|x)}$$

Multiple samples tighten the bound and enhance the model's performance.



### 8.5.3 SVG / VRNN

For sequential data, stochastic variational recurrent networks incorporate latent variables into recurrent architectures. This approach, known as SVG or VRNN, models temporal dependencies alongside latent representations.

## 9 Diffusion models

One interesting note in the course was diffusion models as spectral noise. You saw in that plot that when you do the process of adding the gaussian noise to the image, you lose the high frequency information of the image and you just keep the low frequency information.

The difference of the model diffusion and VAE is that VAE takes from the latent space in one shot but the diffusion models go from noisy to no noise iteratively. This gives time to the model to correct himself, one of the reasons why the diffusion models are so performing at the moment.

### 9.1 Controlling Diffusion models (Part 2)

Il y a un tradeoff entre la quality et la quantite des images generer

Tu plug un encoder text avant par exemple stable diffusion alors les deux captions "An image of a loaf of bread in the shape of a cat." "An image of a cat in the shape of a loaf of bread"

You get almost the same thing which is because the word encoder mix them up and cause the caption to lose the nuances.

Difficulté avec la spécialité dans la génération des images. Genre quelque chose à gauche et un autre à droite.

Aussi difficulté dans les interactions entre les objets et choses.

Dalle essaye toujours de toujours générer 10:10 dans les montres. C'est un biais car les photos de montre sur le net c'est très très souvent 10:10.

Diffusion discret sont le futur?? Mais pas encore de résultats empirique

Les modèles de diffusion discret opérer sur les nombres réels??

Toujours chaîne de Markov

## 10 Reinforcement Learning

### 10.1 Reward Hacking

### 10.2 Markov Decision Process

### 10.3 Policies

### 10.4 Bellman Equations

### 10.5 Value-Based RL

#### 10.5.1 Goal

Learn value functions

#### 10.5.2 Policy

You derive the policy from the value function, typically by choosing the action with the highest value (greedy policy).

#### 10.5.3 Q-Learning

#### 10.5.4 Deep Q-Learning

### 10.6 Policy-Based RL

#### 10.6.1 Goal

Learn policy parameters

#### 10.6.2 Policy

You directly learn a parameterized policy (e.g., neural network) that outputs actions or action probabilities.

## 10.7 Actor-Critic

### 10.7.1 Policy

You learn both a policy (the actor) and a value function (the critic). The actor selects actions, while the critic evaluates them.

## 11 Reinforcement Learning from Human Feedback (RLHF)

Language models are not aligned with user intent [Ouyang et al., 2022]. Finetuning can help!

There are two steps to this situation.

First step is pretraining on language modeling (lots of text; learn general things!)

Second step is finetuning with human feedback (many tasks, many labels, adapt to the tasks!)

We can generally do two type of finetuning. One is supervised training and the other by reinforcement (RLHF).

### 11.1 Flan-T5

### 11.2 Limitations of Finetuning

One problem is that we have open-ended questions that are difficult to evaluate what is "good" and what is "bad". Especially in creative tasks. If we finetune too much on those tasks there will be overfitting quickly.

Another problem is that language modeling penalizes all token-level mistakes equally, but some errors are worse than others.

For example when you evaluate the output of the model, we will use cross-entropy based on the probabilities output of the model. Why it doesn't punish the model too much even if he output a synonym is that in the probabilities, the synonyms will generally both at the same %. Therefore the model is not too much punished.

### 11.3 Human-in-the-loop

This method is by far the best way to finetune models but there are some big problems with this technique too.

One of the main problem is that this process is expensive. The solution that companies have been using is to model their preferences as a separate (NLP) problem. [Knox and Stone, 2009] - One example is chatgpt options to thumbs up/down the answers, regenerate answers and make the user choose between two answers.

Another problem is that human judgment is noisy and miscalibrated. A solution that was found was that instead of asking for direct ratings, the system should ask for pairwise comparisons, which can be more reliable. [Phelps et al., 2015; Clark et al., 2018]

### 11.4 Reward

Almost always this for RLHF

## 11.5 InstructGPT

Step 1 (Supervised training): Collect demonstration data and train a supervised policy. Step 2... Step 3...

## 11.6 LLM-as-a-judge

- PAS UTILISER POUR REINFORCEMENT!! JUST BENCHMARKING

Instead of using a human in the feed-back-loop, we can use a LLM to judge the awnsers. This method has been increasing in popularity in recent years. We generally choose the biggest model to make the judgments on the awnsers.

Here are some notes on that: - Models are heavily positionally biased - Models often rate on syntax response length

## 11.7 Benchmarks and comparisons

### 11.7.1 MT Bench

## 12 Self-Supervised Learning

The main idea behind self-supervision is to design auxiliary (“pretext”) tasks that generate supervisory signals directly from the structure of the data, thus eliminating the need for manual labels. These pretext tasks force the model to learn meaningful representations that are useful when transferred to downstream tasks.

### 12.1 Pretext Task Paradigm

At its core, the self-supervised framework leverages pretext tasks to generate supervisory signals from the data itself.

- **Data Transformation:** Raw inputs (e.g., images) are modified using operations such as cropping, permutation, or geometric transformation.
- **Task Definition:** The network is trained to predict the transformation, the relative context, or to compare different views of the same instance.
- **Representation Learning:** In solving the pretext task, the network learns high-level semantic or structural features that generalize well.

### 12.2 Spatial Context and Structural Tasks

#### 12.2.1 Context Prediction

- **Task:** Given two patches from the same image (with a spatial gap and random jitter), the network predicts their relative spatial configuration.
- **Architecture:** Utilizes a *Siamese network* structure with shared weights to process each patch, and a late-fusion module that combines features (often from a fully connected layer) for classification into one of several spatial configurations.
- **Design Consideration:** Incorporates mechanisms such as patch gaps, jitter, and color preprocessing to prevent the network from learning trivial (low-level) solutions.

#### 12.2.2 Jigsaw Puzzles

- **Task:** An image is split into tiles (e.g., a  $3 \times 3$  grid) and then randomly permuted. The network is tasked with reassembling the puzzle.
- **Architecture:** Implements a multi-stream network where each tile is processed by a branch with tied weights, followed by a permutation classification layer that predicts the correct arrangement from a carefully chosen set of possibilities.
- **Outcome:** This method forces the model to understand both object parts and the overall structure.

#### 12.2.3 Rotation Prediction

- **Task:** The network must predict the rotation applied to an image (typically one of  $0^\circ$ ,  $90^\circ$ ,  $180^\circ$ , or  $270^\circ$ ).
- **Architecture:** Consists of a single-stream CNN that extracts features from the rotated image, feeding into a classification head that outputs a probability distribution over rotations.

- **Intuition:** The task compels the network to understand the configuration of objects, as accurate rotation prediction requires awareness of object semantics.

## 12.3 Contrastive Learning and Beyond

Contrastive methods and their extensions drive learning by comparing different views of the same instance against other samples, enforcing invariance across augmentations.

### 12.3.1 MoCo (Momentum Contrast)

- **Idea:** Build a dynamic dictionary using a large queue of negative examples and a momentum-updated encoder.
- **Architecture:**
  - Dual encoders: a query encoder for the current mini-batch and a key encoder updated via a momentum mechanism.
  - A contrastive loss (e.g., InfoNCE) that encourages the query representation to align with its positive key while differentiating from negatives stored in the queue.
- **Advantage:** Decouples the dictionary size from the mini-batch size while ensuring stable key representations.

### 12.3.2 SimCLR (Simple Framework for Contrastive Learning)

- **Idea:** Leverage strong data augmentations to create varied views of the same image and pull their representations together.
- **Architecture:** A standard CNN backbone is augmented with a projection head and trained with a contrastive loss very similar to InfoNCE using large batch sizes and temperature scaling.
- **Outcome:** Establishes a simplified yet highly competitive baseline in self-supervised learning.

### 12.3.3 CLIP (Contrastive Language–Image Pre-training)

- **Idea:** Align image representations with natural language descriptions using a bi-encoder architecture.
- **Architecture:** Includes an image encoder and a separate text encoder. A contrastive loss is used to maximize the similarity of matching image–text pairs and minimize it for non-matching pairs.
- **Impact:** Enables learning of rich, multi-modal representations that transfer effectively across vision and language tasks.

### 12.3.4 BYOL (Bootstrap Your Own Latent)

- **Idea:** Achieves self-supervised learning without explicit negative examples by maintaining an online and a target network.
- **Architecture:** Both networks (often with an added projection head) process augmented views of the same image. The target network is updated as a moving average of the online network.
- **Outcome:** Yields robust representations through careful design and momentum updates despite the absence of negative pairs.



### 12.3.5 Barlow Twins

- **Idea:** Reduce redundancy in feature dimensions by pushing the cross-correlation matrix between different augmentations toward the identity matrix.
- **Architecture:** Uses a shared backbone with a projection head for two augmented views. The loss function combines terms for invariance and decorrelation.
- **Advantage:** Avoids the need for explicit negative sampling while still enforcing meaningful representation learning.

### 12.3.6 DINO (Self-Distillation with No Labels)

- **Idea:** Employ a teacher–student framework, where the teacher (an exponential moving average of the student) provides soft targets that stabilize training.
- **Architecture:** Both the student and teacher networks process multiple augmented views, and a distillation loss aligns the student’s outputs with the teacher’s.
- **Outcome:** Generates strong clustering and attention maps, leading to highly transferable features.

### 12.3.7 JEPA (Joint Embedding Predictive Architecture) and I-JEPA (Improved JEPA)

- **Idea:** Go beyond simple instance discrimination by predicting the representation of one part of an image from its surrounding context.
- **Architecture:** Consists of a context encoder and a predictor network that work on partially observed images (or masked regions) to predict the representations for missing parts.
- **Enhancements in I-JEPA:** Refinements in both prediction and embedding strategies yield more robust and transferable features.
- **Significance:** Aligns with modern trends emphasizing joint embedding frameworks without the explicit need for negative sampling.

## 12.4 Knowledge Distillation

**Knowledge Distillation** has traditionally been used to transfer information from a “teacher” model to a “student” model in supervised settings. It allows further refinements by predicting content based on context and transferring rich teacher knowledge to student models.

Methods like DINO utilize a teacher–student formulation in a label-free setting, where the student is trained to mimic the softened outputs of the teacher.

This process generally enhances feature robustness and improves transferability by internalizing richer representations.