

Super Pense-bête VIP : Intelligence artificielle

Afshine AMIDI et Shervine AMIDI

8 septembre 2019

Table des matières

1 Modèles basés sur le réflex	2
1.1 Prédicteurs linéaires	2
1.1.1 Classification	2
1.1.2 Régression	2
1.2 Minimisation de la fonction objectif	2
1.3 Prédicteurs non linéaires	3
1.4 Algorithme du gradient stochastique	3
1.5 Peaufinage de modèle	3
1.6 Apprentissage non supervisé	4
1.6.1 k -moyennes (en anglais <i>k-means</i>)	4
1.6.2 Analyse des composantes principales	5
2 Modèles basés sur les états	5
2.1 Optimisation de parcours	5
2.1.1 Parcours d'arbre	5
2.1.2 Parcours de graphe	6
2.1.3 Apprentissage des coûts	7
2.1.4 Algorithme A*	7
2.1.5 Relaxation	8
2.2 Processus de décision markovien	8
2.2.1 Notations	8
2.2.2 Applications	9
2.2.3 Cas des transitions et récompenses inconnues	10
2.3 Jeux	10
2.3.1 Accélération de minimax	11
2.3.2 Jeux simultanés	11
2.3.3 Jeux à somme non nulle	12

3 Modèles basés sur les variables	12
3.1 Problèmes de satisfaction de contraintes	12
3.1.1 Graphes de facteurs	12
3.1.2 Mise en ordre dynamique	13
3.1.3 Méthodes approximatives	13
3.1.4 Transformations sur les graphes de facteurs	14
3.2 Réseaux bayésiens	14
3.2.1 Introduction	14
3.2.2 Programmes probabilistes	15
3.2.3 Inférence	15
4 Modèles basés sur la logique	16
4.1 Bases	16
4.2 Base de connaissance	17
4.3 Logique propositionnelle	18
4.4 Calcul des prédictats du premier ordre	18

1 Modèles basés sur le réflex

1.1 Prédicteurs linéaires

Dans cette section, nous allons explorer les modèles basés sur le réflex qui peuvent s'améliorer avec l'expérience s'appuyant sur des données ayant une correspondance entrée-sortie.

□ **Vecteur caractéristique** – Le vecteur caractéristique (en anglais *feature vector*) d'une entrée x est noté $\phi(x)$ et se décompose en :

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \vdots \\ \phi_d(x) \end{bmatrix} \in \mathbb{R}^d$$

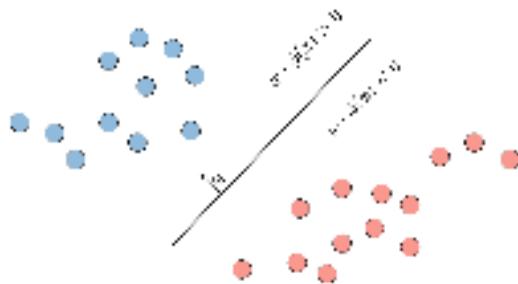
□ **Score** – Le score $s(x,w)$ d'un exemple $(\phi(x),y) \in \mathbb{R}^d \times \mathbb{R}$ associé à un modèle linéaire de paramètres $w \in \mathbb{R}^d$ est donné par le produit scalaire :

$$s(x,w) = w \cdot \phi(x)$$

1.1.1 Classification

□ **Classifieur linéaire** – Étant donnés un vecteur de paramètres $w \in \mathbb{R}^d$ et un vecteur caractéristique $\phi(x) \in \mathbb{R}^d$, le classifieur linéaire binaire f_w est donné par :

$$f_w(x) = \text{sign}(s(x,w)) = \begin{cases} +1 & \text{if } w \cdot \phi(x) > 0 \\ -1 & \text{if } w \cdot \phi(x) < 0 \\ ? & \text{if } w \cdot \phi(x) = 0 \end{cases}$$



□ **Marge** – La marge (en anglais *margin*) $m(x,y,w) \in \mathbb{R}$ d'un exemple $(\phi(x),y) \in \mathbb{R}^d \times \{-1, +1\}$ associée à un modèle linéaire de paramètre $w \in \mathbb{R}^d$ quantifie la confiance associée à une prédiction : plus cette valeur est grande, mieux c'est. Cette quantité est donnée par :

$$m(x,y,w) = s(x,w) \times y$$

1.1.2 Régression

□ **Régression linéaire** – Étant donnés un vecteur de paramètres $w \in \mathbb{R}^d$ et un vecteur caractéristique $\phi(x) \in \mathbb{R}^d$, le résultat d'une régression linéaire de paramètre w , notée f_w , est donné par :

$$f_w(x) = s(x, w)$$

□ **Résidu** – Le résidu $\text{res}(x, y, w) \in \mathbb{R}$ est défini comme étant la différence entre la prédiction $f_w(x)$ et la vraie valeur y :

$$\text{res}(x, y, w) = f_w(x) - y$$

1.2 Minimisation de la fonction objectif

□ **Fonction objectif** – Une fonction objectif (en anglais *loss function*) $\text{Loss}(x, y, w)$ traduit notre niveau d'insatisfaction avec les paramètres w du modèle dans la tâche de prédiction de la sortie y à partir de l'entrée x . C'est une quantité que l'on souhaite minimiser pendant la phase d'entraînement.

□ **Cas de la classification** – Trouver la classe d'un exemple x appartenant à $y \in \{-1, +1\}$ peut être fait par le biais d'un modèle linéaire de paramètre w à l'aide du prédicteur $f_w(x) \triangleq \text{sign}(s(x, w))$. La qualité de cette prédiction peut alors être évaluée au travers de la marge $m(x, y, w)$ intervenant dans les fonctions objectif suivantes :

Fonction objectif	Zéro-un	Hinge	Logistique
$\text{Loss}(x, y, w)$	$1_{\{m(x, y, w) \leq 0\}}$	$\max(1 - m(x, y, w), 0)$	$\log(1 + e^{-m(x, y, w)})$
Illustration			

□ **Cas de la régression** – Prédire la valeur $y \in \mathbb{R}$ associée à l'exemple x peut être fait par le biais d'un modèle linéaire de paramètre w à l'aide du prédicteur $f_w(x) \triangleq s(x, w)$. La qualité de cette prédiction peut alors être évaluée au travers du résidu $\text{res}(x, y, w)$ intervenant dans les fonctions objectif suivantes :

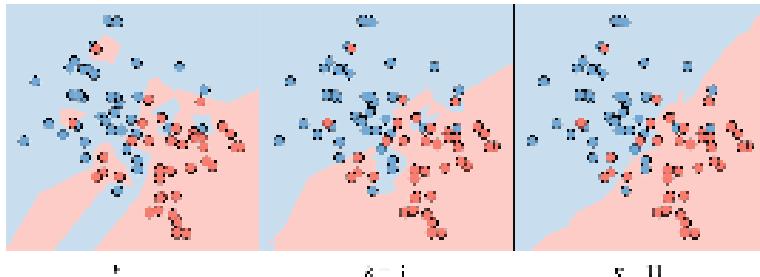
Nom	Erreur quadratique	Erreur absolue
$\text{Loss}(x, y, w)$	$(\text{res}(x, y, w))^2$	$ \text{res}(x, y, w) $
Illustration		

□ **Processus de minimisation de la fonction objectif** – Lors de l'entraînement d'un modèle, on souhaite minimiser la valeur de la fonction objectif évaluée sur l'ensemble d'entraînement :

$$\text{TrainLoss}(w) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} \text{Loss}(x,y,w)$$

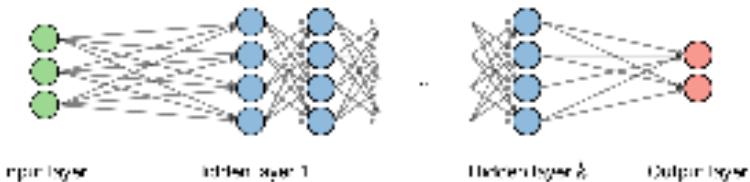
1.3 Prédicteurs non linéaires

□ **k plus proches voisins** – L'algorithme des k plus proches voisins (en anglais *k-nearest neighbors* ou *k-NN*) est une approche non paramétrique où la réponse associée à un exemple est déterminée par la nature de ses k plus proches voisins de l'ensemble d'entraînement. Cette démarche peut être utilisée pour la classification et la régression.



Remarque : plus le paramètre k est grand, plus le biais est élevé. À l'inverse, la variance devient plus élevée lorsque l'on réduit la valeur k .

□ **Réseaux de neurones** – Les réseaux de neurones (en anglais *neural networks*) constituent un type de modèle basés sur des couches (en anglais *layers*). Parmi les types de réseaux populaires, on peut compter les réseaux de neurones convolutionnels et récurrents (abréviés respectivement en CNN et RNN *en anglais*). Une partie du vocabulaire associé aux réseaux de neurones est détaillée dans la figure ci-dessous :



En notant i la i -ème couche du réseau et j son j -ième neurone, on a :

$$z_j^{[i]} = w_j^{[i]T} x + b_j^{[i]}$$

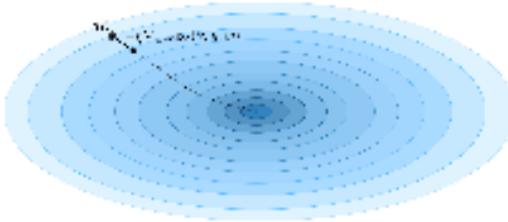
où l'on note w , b , x , z le coefficient, le biais ainsi que la variable de sortie respectivement.

1.4 Algorithme du gradient stochastique

□ **Descente de gradient** – En notant $\eta \in \mathbb{R}$ le taux d'apprentissage (en anglais learning rate ou step size), la règle de mise à jour des coefficients pour cet algorithme utilise la fonction objectif

$\text{Loss}(x, y, w)$ de la manière suivante :

$$w \leftarrow w - \eta \nabla_w \text{Loss}(x, y, w)$$



□ **Mises à jour stochastiques** – L'algorithme du gradient stochastique (en anglais *stochastic gradient descent* ou *SGD*) met à jour les paramètres du modèle en parcourant les exemples $(\phi(x), y) \in \mathcal{D}_{\text{train}}$ de l'ensemble d'entraînement un à un. Cette méthode engendre des mises à jour rapides à calculer mais qui manquent parfois de robustesse.

□ **Mises à jour par lot** – L'algorithme du gradient par lot (en anglais *batch gradient descent* ou *BGD*) met à jour les paramètres du modèle en utilisant des lots entiers d'exemples (e.g. la totalité de l'ensemble d'entraînement) à la fois. Cette méthode calcule des directions de mise à jour des coefficients plus stable au prix d'un plus grand nombre de calculs.

1.5 Peaufinage de modèle

□ **Classe d'hypothèses** – Une classe d'hypothèses \mathcal{F} est l'ensemble des prédicteurs candidats ayant un $\phi(x)$ fixé et dont le paramètre w peut varier :

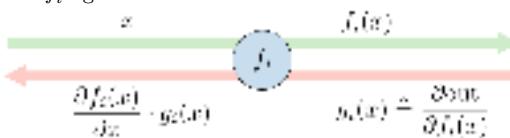
$$\mathcal{F} = \{f_w : w \in \mathbb{R}^d\}$$

□ **Fonction logistique** – La fonction logistique σ , aussi appelée en anglais *sigmoid function*, est définie par :

$$\forall z \in]-\infty, +\infty[, \quad \sigma(z) = \frac{1}{1 + e^{-z}}$$

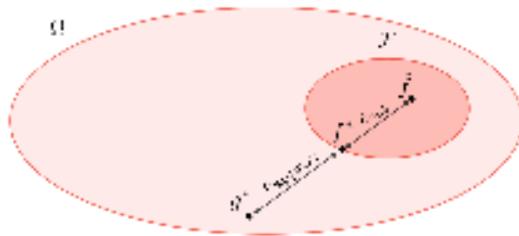
Remarque : la dérivée de cette fonction s'écrit $\sigma'(z) = \sigma(z)(1 - \sigma(z))$.

□ **Rétropropagation du gradient (en anglais *backpropagation*)** – La propagation avant (en anglais *forward pass*) est effectuée via f_i , valeur correspondant à l'expression appliquée à l'étape i . La propagation de l'erreur vers l'arrière (en anglais *backward pass*) se fait via $g_i = \frac{\partial \text{out}}{\partial f_i}$ et décrit la manière dont f_i agit sur la sortie du réseau.

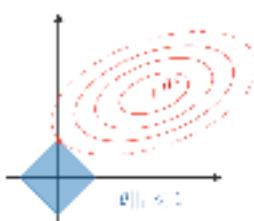
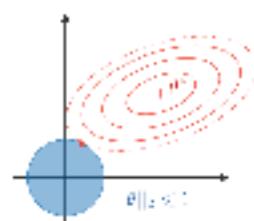
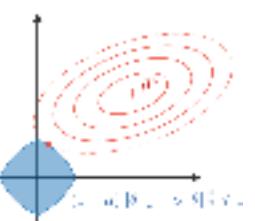


□ **Erreur d'approximation et d'estimation** – L'erreur d'approximation ϵ_{approx} représente la distance entre la classe d'hypothèses \mathcal{F} et le prédicteur optimal g^* . De son côté, l'erreur

d'estimation ϵ_{est} quantifie la qualité du prédicteur \hat{f} par rapport au meilleur prédicteur f^* de la classe d'hypothèses \mathcal{F} .



Régularisation – Le but de la régularisation est d'empêcher le modèle de surapprendre (en anglais *overfit*) les données en s'occupant ainsi des problèmes de variance élevée. La table suivante résume les différents types de régularisation couramment utilisés :

LASSO	Ridge	Elastic Net
<ul style="list-style-type: none"> - Réduit les coefficients à 0 - Bénéfique pour la sélection de variables 	Rapetissit les coefficients	Compromis entre sélection de variables et coefficients de faible magnitude
 $\ \theta\ _1 \leq 1$	 $\ \theta\ _2 \leq 1$	 $\ \theta\ _2 \leq \sqrt{\alpha} + \sqrt{1-\alpha}\ \theta\ _1$
$\dots + \lambda \ \theta\ _1$ $\lambda \in \mathbb{R}$	$\dots + \lambda \ \theta\ _2^2$ $\lambda \in \mathbb{R}$	$\dots + \lambda \left[(1-\alpha) \ \theta\ _1 + \alpha \ \theta\ _2^2 \right]$ $\lambda \in \mathbb{R}, \alpha \in [0,1]$

Hyperparamètres – Les hyperparamètres sont les paramètres de l'algorithme d'apprentissage et incluent parmi d'autres le type de caractéristiques utilisé ainsi que le paramètre de régularisation λ , le nombre d'itérations T le taux d'apprentissage η .

Vocabulaire – Lors de la sélection d'un modèle, on divise les données en 3 différentes parties :

Données d'entraînement	Données de validation	Données de test
<ul style="list-style-type: none"> - Le modèle y est entraîné - Constitue normalement 80 du jeu de données 	<ul style="list-style-type: none"> - Le modèle y est évalué - Constitue normalement 20 du jeu de données - Aussi appelé données de développement (en anglais <i>hold-out</i> ou <i>development set</i>) 	<ul style="list-style-type: none"> - Le modèle y donne ses prédictions - Données jamais observées

Une fois que le modèle a été choisi, il est entrainé sur le jeu de données entier et testé sur l'ensemble de test (qui n'a jamais été vu). Ces derniers sont représentés dans la figure ci-dessous :



1.6 Apprentissage non supervisé

Les méthodes d'apprentissage non supervisé visent à découvrir la structure (parfois riche) des données.

1.6.1 k -moyennes (en anglais k -means)

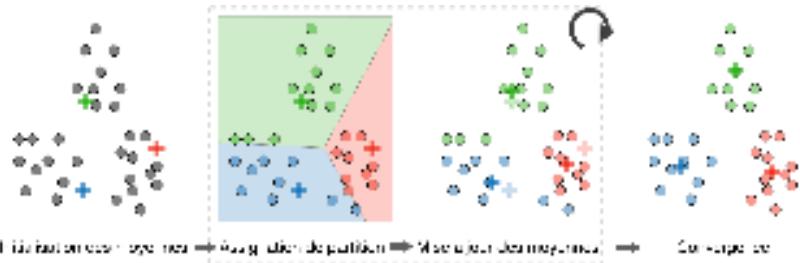
Partitionnement – Étant donné un ensemble d'entraînement $\mathcal{D}_{\text{train}}$, le but d'un algorithme de partitionnement (en anglais *clustering*) est d'assigner chaque point $\phi(x_i)$ à une partition $z_i \in \{1, \dots, k\}$.

Fonction objectif – La fonction objectif d'un des principaux algorithmes de partitionnement, k -moyennes, est donné par :

$$\text{Loss}_{k\text{-means}}(x, \mu) = \sum_{i=1}^n \|\phi(x_i) - \mu_{z_i}\|^2$$

Algorithme – Après avoir aléatoirement initialisé les centroides de partitions $\mu_1, \mu_2, \dots, \mu_k \in \mathbb{R}^n$, l'algorithme k -moyennes répète l'étape suivante jusqu'à convergence :

$$z_i = \arg \min_j \|\phi(x_i) - \mu_j\|^2 \quad \text{and} \quad \mu_j = \frac{\sum_{i=1}^m 1_{\{z_i=j\}} \phi(x_i)}{\sum_{i=1}^m 1_{\{z_i=j\}}}$$



1.6.2 Analyse des composantes principales

□ **Valeur propre, vecteur propre** – Étant donnée une matrice $A \in \mathbb{R}^{n \times n}$, λ est dite être une valeur propre de A s'il existe un vecteur $z \in \mathbb{R}^n \setminus \{0\}$, appelé vecteur propre, tel que :

$$Az = \lambda z$$

□ **Théorème spectral** – Soit $A \in \mathbb{R}^{n \times n}$. Si A est symétrique, alors A est diagonalisable par une matrice réelle orthogonale $U \in \mathbb{R}^{n \times n}$. En notant $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$, on a :

$$\exists \Lambda \text{ diagonal}, \quad A = U \Lambda U^T$$

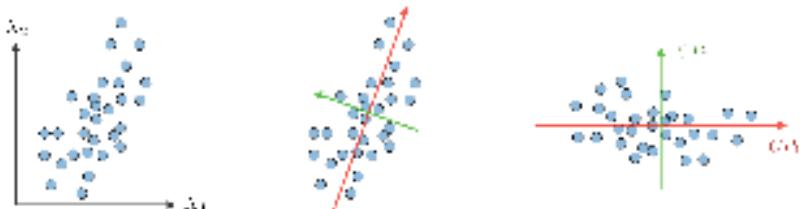
Remarque : le vecteur propre associé à la plus grande valeur propre est appelé le vecteur propre principal de la matrice A .

□ **Algorithme** – La procédure d'analyse des composantes principales (en anglais *Principal Component Analysis* ou *PCA*) est une technique de réduction de dimension qui projette les données sur k dimensions en maximisant la variance des données de la manière suivante :

- Étape 1 : Normaliser les données pour avoir une moyenne de 0 et un écart-type de 1.

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{\sigma_j} \quad \text{where} \quad \mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)} \quad \text{and} \quad \sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2$$

- Étape 2 : Calculer $\Sigma = \frac{1}{m} \sum_{i=1}^m x^{(i)} x^{(i)T} \in \mathbb{R}^{n \times n}$, qui est symétrique avec des valeurs propres réelles.
- Étape 3 : Calculer $u_1, \dots, u_k \in \mathbb{R}^n$ les k valeurs propres principales orthogonales de Σ , i.e. les vecteurs propres orthogonaux des k valeurs propres les plus grandes.
- Étape 4 : Projeter les données sur $\text{span}_{\mathbb{R}}(u_1, \dots, u_k)$. Cette procédure maximise la variance sur tous les espaces à k dimensions.



Données dans l'espace initial \Rightarrow Vecteurs composantes principales \Rightarrow Composantes principales du CVA

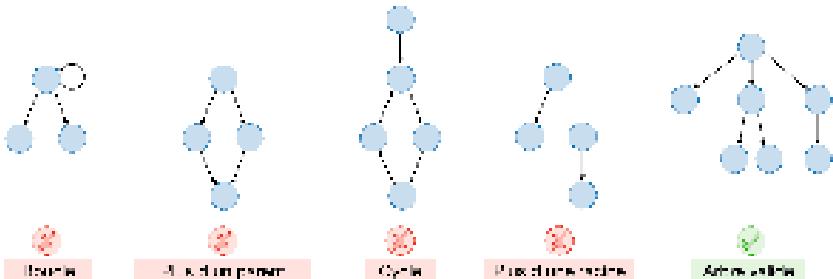
2 Modèles basés sur les états

2.1 Optimisation de parcours

Dans cette section, nous supposons qu'en effectuant une action a à partir d'un état s , on arrive de manière déterministe à l'état $\text{Succ}(s,a)$. Le but de cette étude est de déterminer une séquence d'actions $(a_1, a_2, a_3, a_4, \dots)$ démarrant d'un état initial et aboutissant à un état final. Pour y parvenir, notre objectif est de minimiser le coût associé à ces actions à l'aide de modèles basés sur les états (*state-based model* en anglais).

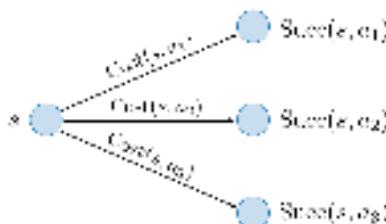
2.1.1 Parcours d'arbre

Cette catégorie d'algorithmes explore tous les états et actions possibles. Même si leur consommation en mémoire est raisonnable et peut supporter des espaces d'états de taille très grande, ce type d'algorithmes est néanmoins susceptible d'engendrer des complexités en temps exponentielles dans le pire des cas.



□ Problème de recherche – Un problème de recherche est défini par :

- un état de départ s_{start}
- des actions $\text{Actions}(s)$ pouvant être effectuées depuis l'état s
- le coût de l'action $\text{Cost}(s,a)$ depuis l'état s pour effectuer l'action a
- le successeur $\text{Succ}(s,a)$ de l'état s après avoir effectué l'action a
- la connaissance d'avoir atteint ou non un état final $\text{IsEnd}(s)$

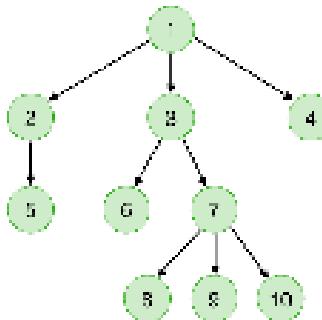


L'objectif est de trouver un chemin minimisant le coût total des actions utilisées.

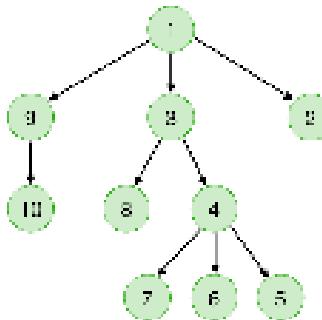
□ Retour sur trace – L'algorithme de retour sur trace (en anglais *backtracking search*) est un algorithme récursif explorant naïvement toutes les possibilités jusqu'à trouver le chemin de coût minimal. Ici, le coût des actions peut aussi bien être positif que négatif.

□ Parcours en largeur (BFS) – L'algorithme de parcours en largeur (en anglais *breadth-first search* ou *BFS*) est un algorithme de parcours de graphe traversant chaque niveau de manière successive. On peut le coder de manière itérative à l'aide d'une queue stockant à chaque étape

les prochains nœuds à visiter. Cet algorithme suppose que le coût de toutes les actions est égal à une constante $c \geq 0$.



□ Parcours en profondeur (DFS) – L'algorithme de parcours en profondeur (en anglais *depth-first search* ou *DFS*) est un algorithme de parcours de graphe traversant chaque chemin qu'il emprunte aussi loin que possible. On peut le coder de manière récursive, ou itérative à l'aide d'une pile qui stocke à chaque étape les prochains nœuds à visiter. Cet algorithme suppose que le coût de toutes les actions est égal à 0.



□ Approfondissement itératif – L'astuce de l'approfondissement itératif (en anglais *iterative deepening*) est une modification de l'algorithme de DFS qui l'arrête après avoir atteint une certaine profondeur, garantissant l'optimalité de la solution trouvée quand toutes les actions ont un même coût constant $c \geq 0$.

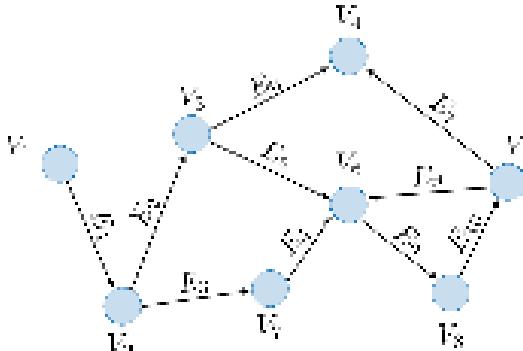
□ Récapitulatif des algorithmes de parcours d'arbre – En notant b le nombre d'actions par état, d la profondeur de la solution et D la profondeur maximale, on a :

Algorithme	Coût des actions	Espace	Temps
Retour sur trace	peu importe	$\mathcal{O}(D)$	$\mathcal{O}(b^D)$
Parcours en largeur	$c \geq 0$	$\mathcal{O}(b^d)$	$\mathcal{O}(b^d)$
Parcours en profondeur	0	$\mathcal{O}(D)$	$\mathcal{O}(b^D)$
DFS-approfondissement itératif	$c \geq 0$	$\mathcal{O}(d)$	$\mathcal{O}(b^d)$

2.1.2 Parcours de graphe

Cette catégorie d'algorithmes basés sur les états vise à trouver des chemins optimaux avec une complexité moins grande qu'exponentielle. Dans cette section, nous allons nous concentrer sur la programmation dynamique et la recherche à coût uniforme.

- **Graph** – Un graphe se compose d'un ensemble de sommets V (aussi appelés nœuds) et d'arêtes E (appelés arcs lorsque le graphe est orienté).



Remarque : un graphe est dit être acyclique lorsqu'il ne contient pas de cycle.

- **État** – Un état contient le résumé des actions passées suffisant pour choisir les actions futures de manière optimale.

- **Programmation dynamique** – La programmation dynamique (en anglais *dynamic programming* ou *DP*) est un algorithme de recherche de type retour sur trace qui utilise le principe de mémoisation (i.e. les résultats intermédiaires sont enregistrés) et ayant pour but de trouver le chemin à coût minimal allant de l'état s à l'état final s_{end} . Cette procédure peut potentiellement engendrer des économies exponentielles si on la compare aux algorithmes de parcours de graphe traditionnels, et a la propriété de ne marcher que dans le cas de graphes acycliques. Pour un état s donné, le coût futur est calculé de la manière suivante :

$$\text{FutureCost}(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ \min_{a \in \text{Actions}(s)} [\text{Cost}(s,a) + \text{FutureCost}(\text{Succ}(s,a))] & \text{otherwise} \end{cases}$$

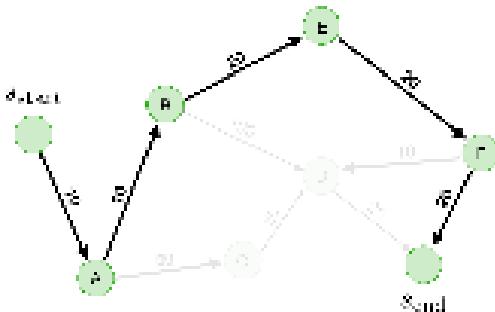
start	-	-	-	-	-	-
-	-	-	-	-	4	-
-	4	-	-	-	4	-
-	-	-	-	-	-	-
-	4	-	-	-	-	-
-	-	-	4	-	-	4
-	-	-	-	-	-	4

Remarque : la figure ci-dessus illustre une approche ascendante alors que la formule nous donne l'intuition d'une résolution avec une approche descendante.

□ **Types d'états** – La table ci-dessous présente la terminologie relative aux états dans le contexte de la recherche à coût uniforme :

État	Explication
Exploré \mathcal{E}	États pour lesquels le chemin optimal a déjà été trouvé
Frontière \mathcal{F}	États rencontrés mais pour lesquels on se demande toujours comment s'y rendre avec un coût minimal
Inexploré \mathcal{U}	États non rencontrés jusqu'à présent

□ **Recherche à coût uniforme** – La recherche à coût uniforme (*uniform cost search* ou *UCS* en anglais) est un algorithme de recherche qui a pour but de trouver le chemin le plus court entre les états s_{start} et s_{end} . Celui-ci explore les états s en les triant par coût croissant de $\text{PastCost}(s)$ et repose sur le fait que toutes les actions ont un coût non négatif.



Remarque 1 : UCS fonctionne de la même manière que l'algorithme de Dijkstra.

Remarque 2 : cet algorithme ne marche pas sur une configuration contenant des actions à coût négatif. Quelqu'un pourrait penser à ajouter une constante positive à tous les coûts, mais cela ne résoudrait rien puisque le problème résultant serait différent.

□ **Théorème de correction** – Lorsqu'un état s passe de la frontière \mathcal{F} à l'ensemble exploré \mathcal{E} , sa priorité est égale à $\text{PastCost}(s)$, représentant le chemin de coût minimal allant de s_{start} à s .

□ **Récapitulatif des algorithmes de parcours de graphe** – En notant N le nombre total d'états dont n sont explorés avant l'état final s_{end} , on a :

Algorithm	Acyclicité	Coûts	Temps/Espace
Programmation dynamique	oui	peu importe	$\mathcal{O}(N)$
Recherche à coût uniforme	non	$c \geq 0$	$\mathcal{O}(n \log(n))$

Remarque : ce décompte de la complexité suppose que le nombre d'actions possibles à partir de chaque état est constant.

2.1.3 Apprentissage des coûts

Supposons que nous ne sommes pas donnés les valeurs de $\text{Cost}(s,a)$. Nous souhaitons estimer ces quantités à partir d'un ensemble d'apprentissage de chemins à coût minimaux d'actions (a_1, a_2, \dots, a_k) .

□ **Perceptron structuré** – L'algorithme du perceptron structuré vise à apprendre de manière itérative les coûts des paires état-action. À chaque étape, il :

- fait décroître le coût estimé de chaque état-action du vrai chemin minimisant y donné par la base d'apprentissage,
- fait croître le coût estimé de chaque état-action du chemin y' prédit comme étant minimisant par les paramètres appris par l'algorithme.

Remarque : plusieurs versions de cette algorithme existent, l'une d'elles réduisant ce problème à l'apprentissage du coût de chaque action a et l'autre paramétrisant chaque $\text{Cost}(s,a)$ à un vecteur de paramètres pouvant être appris.

2.1.4 Algorithme A*

□ **Fonction heuristique** – Une heuristique est une fonction h opérant sur les états s , où chaque $h(s)$ vise à estimer $\text{FutureCost}(s)$, le coût du chemin optimal allant de s à s_{end} .



□ **Algorithme** – A^* est un algorithme de recherche visant à trouver le chemin le plus court entre un état s et un état final s_{end} . Il le fait en explorant les états s triés par ordre croissant de $\text{PastCost}(s) + h(s)$. Cela revient à utiliser l'algorithme UCS où chaque arête est associée au coût $\text{Cost}'(s,a)$ donné par :

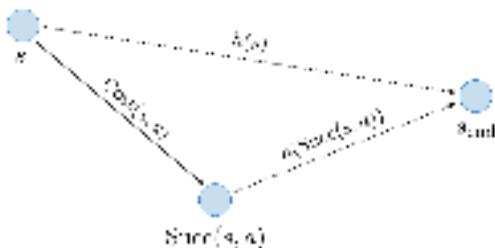
$$\text{Cost}'(s,a) = \text{Cost}(s,a) + h(\text{Succ}(s,a)) - h(s)$$

Remarque : cet algorithme peut être vu comme une version biaisée de UCS explorant les états estimés comme étant plus proches de l'état final.

□ **Consistance** – Une heuristique h est dite consistante si elle satisfait les deux propriétés suivantes :

- Pour tous états s et actions a ,

$$h(s) \leq \text{Cost}(s,a) + h(\text{Succ}(s,a))$$



— L'état final vérifie la propriété :

$$h(s_{\text{end}}) = 0$$



□ Correction – Si h est consistante, alors A^* renvoie le chemin de coût minimal.

□ Admissibilité – Une heuristique h est dite admissible si l'on a :

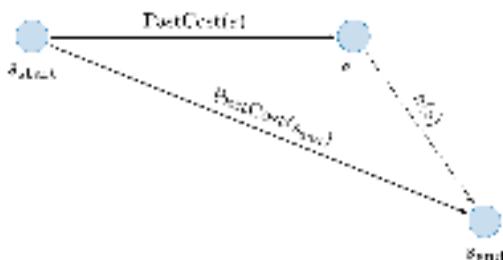
$$h(s) \leq \text{FutureCost}(s)$$

□ Théorème – Soit $h(s)$ une heuristique. On a :

$$h(s) \text{ consistante} \implies h(s) \text{ admissible}$$

□ Efficacité – A^* explore les états s satisfaisant l'équation :

$$\text{PastCost}(s) \leq \text{PastCost}(s_{\text{end}}) - h(s)$$



Remarque : avoir $h(s)$ élevé est préférable puisque cette équation montre que le nombre d'états s à explorer est alors réduit.

2.1.5 Relaxation

C'est un type de procédure permettant de produire des heuristiques consistantes. L'idée est de trouver une fonction de coût facile à exprimer en enlevant des contraintes au problème, et ensuite l'utiliser en tant qu'heuristique.

□ Relaxation d'un problème de recherche – La relaxation d'un problème de recherche P aux coûts Cost est noté P_{rel} avec coûts Cost_{rel} , et vérifie la relation :

$$\text{Cost}_{\text{rel}}(s,a) \leq \text{Cost}(s,a)$$

□ Relaxation d'une heuristique – Étant donné la relaxation d'un problème de recherche P_{rel} , on définit l'heuristique relaxée $h(s) = \text{FutureCost}_{\text{rel}}(s)$ comme étant le chemin de coût minimal allant de s à un état final dans le graphe de fonction de coût $\text{Cost}_{\text{rel}}(s,a)$.

□ Consistance de la relaxation d'heuristiques – Soit P_{rel} une relaxation d'un problème de recherche. Par théorème, on a :

$$h(s) = \text{FutureCost}_{\text{rel}}(s) \implies h(s) \text{ consistante}$$

□ Compromis lors du choix d'heuristique – Le choix d'heuristique se repose sur un compromis entre :

- Complexité de calcul : $h(s) = \text{FutureCost}_{\text{rel}}(s)$ doit être facile à calculer. De manière préférable, cette fonction peut s'exprimer de manière explicite et elle permet de diviser le problème en sous-parties indépendantes.
- Approximation adéquate : l'heuristique $h(s)$ devrait être assez proche de $\text{FutureCost}(s)$ ce qui veut dire qu'il ne faudrait pas enlever trop de contraintes.

□ Heuristique max – Soient $h_1(s), h_2(s)$ deux heuristiques. On a la propriété suivante :

$$h_1(s), h_2(s) \text{ consistante} \implies h(s) = \max\{h_1(s), h_2(s)\} \text{ consistante}$$

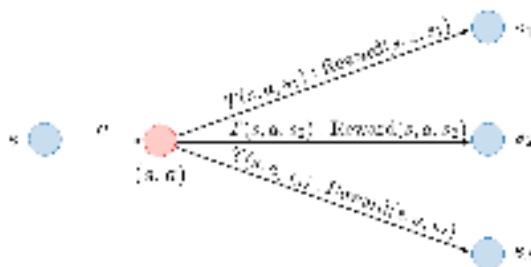
2.2 Processus de décision markovien

Dans cette section, on suppose qu'effectuer l'action a à partir de l'état s peut mener de manière probabiliste à plusieurs états s'_1, s'_2, \dots . Dans le but de trouver ce qu'il faudrait faire entre un état initial et un état final, on souhaite trouver une stratégie maximisant la quantité des récompenses en utilisant un outil adapté à l'imprévisibilité et l'incertitude : les processus de décision markoviens.

2.2.1 Notations

□ Définition – 'objectif d'un processus de décision markovien (en anglais *Markov decision process* ou *MDP*) est de maximiser la quantité de récompenses. Un tel problème est défini par :

- un état de départ s_{start}
- l'ensemble des actions $\text{Actions}(s)$ pouvant être effectuées à partir de l'état s
- la probabilité de transition $T(s, a, s')$ de l'état s vers l'état s' après avoir pris l'action a
- la récompense $\text{Reward}(s, a, s')$ pour être passé de l'état s à l'état s' après avoir pris l'action a
- la connaissance d'avoir atteint ou non un état final $\text{IsEnd}(s)$
- un facteur de dévaluation $0 \leq \gamma \leq 1$



□ Probabilités de transition – La probabilité de transition $T(s, a, s')$ représente la probabilité de transitionner vers l'état s' après avoir effectué l'action a en étant dans l'état s . Chaque $s' \mapsto T(s, a, s')$ est une loi de probabilité :

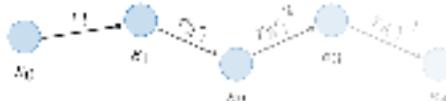
$$\forall s,a, \quad \sum_{s' \in \text{States}} T(s,a,s') = 1$$

□ Politique – Une politique π est une fonction liant chaque état s à une action a , i.e.

$$\boxed{\pi : s \mapsto a}$$

□ Utilité – L'utilité d'un chemin (s_0, \dots, s_k) est la somme des récompenses dévaluées récoltées sur ce chemin. En d'autres termes,

$$\boxed{u(s_0, \dots, s_k) = \sum_{i=1}^k r_i \gamma^{i-1}}$$



Remarque : la figure ci-dessus illustre le cas $k = 4$.

□ Q-value – La fonction de valeur des états-actions (*Q-value* en anglais) d'une politique π évaluée à l'état s avec l'action a , aussi notée $Q_\pi(s,a)$, est l'espérance de l'utilité partant de l'état s avec l'action a et adoptant ensuite la politique π . Cette fonction est définie par :

$$\boxed{Q_\pi(s,a) = \sum_{s' \in \text{States}} T(s,a,s') [\text{Reward}(s,a,s') + \gamma V_\pi(s')]}$$

□ Fonction de valeur des états d'une politique – La fonction de valeur des états d'une politique π évaluée à l'état s , aussi notée $V_\pi(s)$, est l'espérance de l'utilité partant de l'état s et adoptant ensuite la politique π . Cette fonction est définie par :

$$\boxed{V_\pi(s) = Q_\pi(s, \pi(s))}$$

Remarque : $V_\pi(s)$ vaut 0 si s est un état final.

2.2.2 Applications

□ Évaluation d'une politique – Étant donnée une politique π , on peut utiliser l'algorithme itératif d'évaluation de politiques (en anglais *policy evaluation*) pour estimer V_π :

- Initialisation : pour tous les états s , on a

$$\boxed{V_\pi^{(0)}(s) \leftarrow 0}$$

- Itération : pour t allant de 1 à T_{PE} , on a

$$\forall s, \quad \boxed{V_\pi^{(t)}(s) \leftarrow Q_\pi^{(t-1)}(s, \pi(s))}$$

avec

$$Q_{\pi}^{(t-1)}(s, \pi(s)) = \sum_{s' \in \text{States}} T(s, \pi(s), s') \left[\text{Reward}(s, \pi(s), s') + \gamma V_{\pi}^{(t-1)}(s') \right]$$

Remarque : en notant S le nombre d'états, A le nombre d'actions par états, S' le nombre de successeurs et T le nombre d'itérations, la complexité en temps est alors de $\mathcal{O}(TPESS')$.

□ **Q-value optimale** – La Q -value optimale $Q_{\text{opt}}(s, a)$ d'un état s avec l'action a est définie comme étant la Q -value maximale atteinte avec n'importe quelle politique. Elle est calculée avec la formule :

$$Q_{\text{opt}}(s, a) = \sum_{s' \in \text{States}} T(s, a, s') \left[\text{Reward}(s, a, s') + \gamma V_{\text{opt}}(s') \right]$$

□ **Valeur optimale** – La valeur optimale $V_{\text{opt}}(s)$ d'un état s est définie comme étant la valeur maximum atteinte par n'importe quelle politique. Elle est calculée avec la formule :

$$V_{\text{opt}}(s) = \max_{a \in \text{Actions}(s)} Q_{\text{opt}}(s, a)$$

□ **Politique optimale** – La politique optimale π_{opt} est définie comme étant la politique liée aux valeurs optimales. Elle est définie par :

$$\forall s, \quad \pi_{\text{opt}}(s) = \operatorname{argmax}_{a \in \text{Actions}(s)} Q_{\text{opt}}(s, a)$$

□ **Itération sur la valeur** – L'algorithme d'itération sur la valeur (en anglais *value iteration*) vise à trouver la valeur optimale V_{opt} ainsi que la politique optimale π_{opt} en deux temps :

— Initialisation : pour tout état s , on a

$$V_{\text{opt}}^{(0)}(s) \leftarrow 0$$

— Itération : pour t allant de 1 à T_{VI} , on a

$$\forall s, \quad V_{\text{opt}}^{(t)}(s) \leftarrow \max_{a \in \text{Actions}(s)} Q_{\text{opt}}^{(t-1)}(s, a)$$

avec

$$Q_{\text{opt}}^{(t-1)}(s, a) = \sum_{s' \in \text{States}} T(s, a, s') \left[\text{Reward}(s, a, s') + \gamma V_{\text{opt}}^{(t-1)}(s') \right]$$

Remarque : si $\gamma < 1$ ou si le graphe associé au processus de décision markovien est acyclique, alors l'algorithme d'itération sur la valeur est garanti de converger vers la bonne solution.

2.2.3 Cas des transitions et récompenses inconnues

On suppose maintenant que les probabilités de transition et les récompenses sont inconnues.

□ Monte-Carlo basé sur modèle – La méthode de Monte-Carlo basée sur modèle (en anglais *model-based Monte Carlo*) vise à estimer $T(s,a,s')$ et $\text{Reward}(s,a,s')$ en utilisant des simulations de Monte-Carlo avec :

$$\widehat{T}(s,a,s') = \frac{\# \text{ de fois où } (s,a,s') \text{ se produit}}{\# \text{ de fois où } (s,a) \text{ se produit}}$$

and

$$\widehat{\text{Reward}}(s,a,s') = r \text{ dans } (s,a,r,s')$$

Ces estimations sont ensuite utilisées pour trouver les Q -values, ainsi que Q_π et Q_{opt} .

Remarque : la méthode de Monte-Carlo basée sur modèle est dite "hors politique" (en anglais "off-policy") car l'estimation produite ne dépend pas de la politique utilisée.

□ Monte-Carlo sans modèle – La méthode de Monte-Carlo sans modèle (en anglais *model-free Monte Carlo*) vise à directement estimer Q_π de la manière suivante :

$$\widehat{Q}_\pi(s,a) = \text{moyenne de } u_t \text{ où } s_{t-1} = s, a_t = a$$

où u_t désigne l'utilité à partir de l'étape t d'un épisode donné.

Remarque : la méthode de Monte-Carlo sans modèle est dite "sur politique" (en anglais "on-policy") car l'estimation produite dépend de la politique π utilisée pour générer les données.

□ Formulation équivalente – En introduisant la constante $\eta = \frac{1}{1+(\#\text{mises à jour } (s,a))}$ et pour chaque triplet (s,a,u) de la base d'apprentissage, la formule de récurrence de la méthode de Monte-Carlo sans modèle s'écrit à l'aide de la combinaison convexe :

$$\widehat{Q}_\pi(s,a) \leftarrow (1 - \eta)\widehat{Q}_\pi(s,a) + \eta u$$

ainsi qu'une formulation mettant en valeur une sorte de gradient :

$$\widehat{Q}_\pi(s,a) \leftarrow \widehat{Q}_\pi(s,a) - \eta(\widehat{Q}_\pi(s,a) - u)$$

□ SARSA – État-action-récompense-état-action (en anglais *state-action-reward-state-action* ou *SARSA*) est une méthode de bootstrap qui estime Q_π en utilisant à la fois des données réelles et estimées dans sa formule de mise à jour. Pour chaque (s,a,r,s',a') , on a :

$$\widehat{Q}_\pi(s,a) \leftarrow (1 - \eta)\widehat{Q}_\pi(s,a) + \eta \left[r + \gamma \widehat{Q}_\pi(s',a') \right]$$

Remarque : l'estimation donnée par SARSA est mise à jour à la volée contrairement à celle donnée par la méthode de Monte-Carlo sans modèle où la mise à jour est uniquement effectuée à la fin de l'épisode.

□ Q-learning – Le Q -apprentissage (en anglais *Q-learning*) est un algorithme hors politique (en anglais *off-policy*) donnant une estimation de Q_{opt} . Pour chaque (s,a,r,s',a') , on a :

$$\widehat{Q}_{\text{opt}}(s,a) \leftarrow (1 - \eta)\widehat{Q}_{\text{opt}}(s,a) + \eta \left[r + \gamma \max_{a' \in \text{Actions}(s')} \widehat{Q}_{\text{opt}}(s',a') \right]$$

□ **Epsilon-glouton** – La politique epsilon-gloutonne (en anglais *epsilon-greedy*) est un algorithme essayant de trouver un compromis entre l'exploration avec probabilité ϵ et l'exploitation avec probabilité $1 - \epsilon$. Pour un état s , la politique π_{act} est calculée par :

$$\boxed{\pi_{\text{act}}(s) = \begin{cases} \underset{a \in \text{Actions}}{\text{argmax}} \hat{Q}_{\text{opt}}(s,a) & \text{avec proba } 1 - \epsilon \\ \text{random from Actions}(s) & \text{avec proba } \epsilon \end{cases}}$$

2.3 Jeux

Dans les jeux (e.g. échecs, backgammon, Go), d'autres agents sont présents et doivent être pris en compte au moment d'élaborer une politique.

□ **Arbre de jeu** – Un arbre de jeu est un arbre détaillant toutes les issues possibles d'un jeu. En particulier, chaque nœud représente un point de décision pour un joueur et chaque chemin liant la racine à une des feuilles traduit une possible instance du jeu.

□ **Jeu à somme nulle à deux joueurs** – C'est un type jeu où chaque état est entièrement observé et où les joueurs jouent de manière successive. On le définit par :

- un état de départ s_{start}
- de possibles actions $\text{Actions}(s)$ partant de l'état s
- du successeur $\text{Succ}(s,a)$ de l'état s après avoir effectué l'action a
- la connaissance d'avoir atteint ou non un état final $\text{IsEnd}(s)$
- l'utilité de l'agent $\text{Utility}(s)$ à l'état final s
- le joueur $\text{Player}(s)$ qui contrôle l'état s

Remarque : nous assumerons que l'utilité de l'agent a le signe opposé de celui de son adversaire.

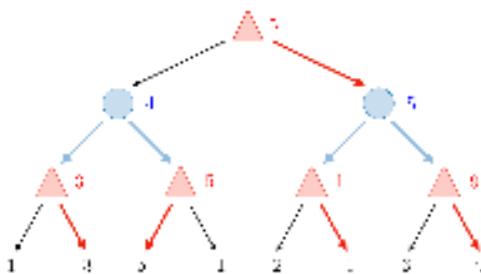
□ **Types de politiques** – Il y a deux types de politiques :

- Les politiques déterministes, notées $\pi_p(s)$, qui représentent pour tout s l'action que le joueur p prend dans l'état s .
- Les politiques stochastiques, notées $\pi_p(s,a) \in [0,1]$, qui sont décrites pour tout s et a par la probabilité que le joueur p prenne l'action a dans l'état s .

□ **Expectimax** – Pour un état donné s , la valeur d'expectimax $V_{\text{exptmax}}(s)$ est l'utilité maximum sur l'ensemble des politiques utilisées par l'agent lorsque celui-ci joue avec un adversaire de politique connue π_{opp} . Cette valeur est calculée de la manière suivante :

$$\boxed{V_{\text{exptmax}}(s) = \begin{cases} \underset{a \in \text{Actions}(s)}{\max} V_{\text{exptmax}}(\text{Succ}(s,a)) & \text{IsEnd}(s) \\ \sum_{a \in \text{Actions}(s)} \pi_{\text{opp}}(s,a) V_{\text{exptmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{opp} \end{cases}}$$

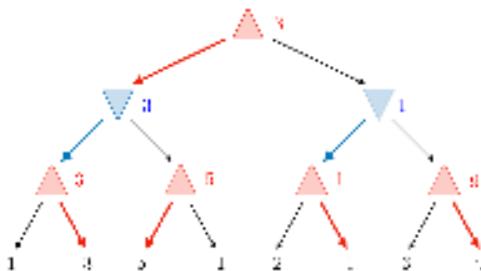
Remarque : expectimax est l'analogie de l'algorithme d'itération sur la valeur pour les MDPs.



□ **Minimax** – Le but des politiques minimax est de trouver une politique optimale contre un adversaire que l'on assume effectuer toutes les pires actions, i.e. toutes celles qui minimisent l'utilité de l'agent. La valeur correspondante est calculée par :

$$V_{\text{minimax}}(s) = \begin{cases} \text{Utility}(s) & \text{max}_{a \in \text{Actions}(s)} V_{\text{minimax}}(\text{Succ}(s,a)) & \text{IsEnd}(s) \\ \min_{a \in \text{Actions}(s)} V_{\text{minimax}}(\text{Succ}(s,a)) & & \text{Player}(s) = \text{agent} \\ & & \text{Player}(s) = \text{opp} \end{cases}$$

Remarque : on peut déduire π_{\max} et π_{\min} à partir de la valeur minimax V_{minimax} .



□ **Propriétés de minimax** – En notant V la fonction de valeur, il y a 3 propriétés sur minimax qu'il faut avoir à l'esprit :

- *Propriété 1* : si l'agent changeait sa politique en un quelconque π_{agent} , alors il ne s'en sortirait pas mieux.

$$\boxed{\forall \pi_{\text{agent}}, \quad V(\pi_{\max}, \pi_{\min}) \geq V(\pi_{\text{agent}}, \pi_{\min})}$$

- *Propriété 2* : si son adversaire change sa politique de π_{\min} à π_{opp} , alors il ne s'en sortira pas mieux.

$$\boxed{\forall \pi_{\text{opp}}, \quad V(\pi_{\max}, \pi_{\min}) \leq V(\pi_{\max}, \pi_{\text{opp}})}$$

- *Propriété 3* : si l'on sait que son adversaire ne joue pas les pires actions possibles, alors la politique minimax peut ne pas être optimale pour l'agent.

$$\boxed{\forall \pi, \quad V(\pi_{\max}, \pi) \leq V(\pi_{\text{exptmax}}, \pi)}$$

À la fin, on a la relation suivante :

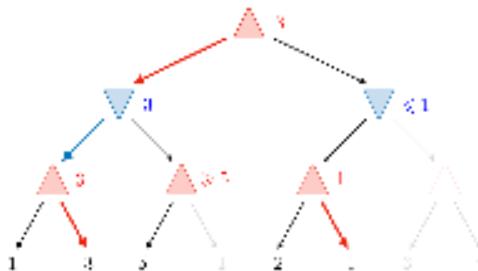
$$\boxed{V(\pi_{\text{exptmax}}, \pi_{\min}) \leq V(\pi_{\max}, \pi_{\min}) \leq V(\pi_{\max}, \pi) \leq V(\pi_{\text{exptmax}}, \pi)}$$

2.3.1 Accélération de minimax

Fonction d'évaluation – Une fonction d'évaluation estime de manière approximative la valeur $V_{\text{minimax}}(s)$ selon les paramètres du problème. Elle est notée $\text{Eval}(s)$.

Remarque : l'analogue de cette fonction utilisé dans les problèmes de recherche est $\text{FutureCost}(s)$.

Élagage alpha-bêta – L'élagage alpha-bêta (en anglais *alpha-beta pruning*) est une méthode exacte d'optimisation employée sur l'algorithme de minimax et a pour but d'éviter l'exploration de parties inutiles de l'arbre de jeu. Pour ce faire, chaque joueur garde en mémoire la meilleure valeur qu'il puisse espérer (appelée α chez le joueur maximisant et β chez le joueur minimisant). À une étape donnée, la condition $\beta < \alpha$ signifie que le chemin optimal ne peut pas passer par la branche actuelle puisque le joueur qui précédait avait une meilleure option à sa disposition.



TD learning – L'apprentissage par différence de temps (en anglais *temporal difference learning* ou *TD learning*) est une méthode utilisée lorsque l'on ne connaît pas les transitions/récompenses. La valeur est alors basée sur la politique d'exploration. Pour pouvoir l'utiliser, on a besoin de connaître les règles du jeu $\text{Succ}(s,a)$. Pour chaque (s,a,r,s') , la mise à jour des coefficients est faite de la manière suivante :

$$w \leftarrow w - \eta [V(s,w) - (r + \gamma V(s',w))] \nabla_w V(s,w)$$

2.3.2 Jeux simultanés

Ce cas est opposé aux jeux joués tour à tour. Il n'y a pas d'ordre pré-déterminé sur le mouvement du joueur.

Jeu simultané à un mouvement – Soient deux joueurs A et B , munis de possibles actions. On note $V(a,b)$ l'utilité de A si A choisit l'action a et B l'action b . V est appelée la matrice de profit (en anglais *payoff matrix*).

Stratégies – Il y a principalement deux types de stratégies :

- Une stratégie pure est une seule action :

$$a \in \text{Actions}$$

- Une stratégie mixte est une loi de probabilité sur les actions :

$$\forall a \in \text{Actions}, \quad 0 \leq \pi(a) \leq 1$$

Évaluation de jeu – La valeur d'un jeu $V(\pi_A, \pi_B)$ quand le joueur A suit π_A et le joueur B suit π_B est telle que :

$$V(\pi_A, \pi_B) = \sum_{a,b} \pi_A(a)\pi_B(b)V(a,b)$$

□ **Théorème minimax** – Soient π_A et π_B des stratégies mixtes. Pour chaque jeu à somme nulle à deux joueurs ayant un nombre fini d'actions, on a :

$$\max_{\pi_A} \min_{\pi_B} V(\pi_A, \pi_B) = \min_{\pi_B} \max_{\pi_A} V(\pi_A, \pi_B)$$

2.3.3 Jeux à somme non nulle

□ **Matrice de profit** – On définit $V_p(\pi_A, \pi_B)$ l'utilité du joueur p .

□ **Équilibre de Nash** – Un équilibre de Nash est défini par (π_A^*, π_B^*) tel qu'aucun joueur n'a d'intérêt de changer sa stratégie. On a :

$$\forall \pi_A, V_A(\pi_A^*, \pi_B^*) \geq V_A(\pi_A, \pi_B^*) \quad \text{et} \quad \forall \pi_B, V_B(\pi_A^*, \pi_B^*) \geq V_B(\pi_A^*, \pi_B)$$

Remarque : dans un jeu à nombre de joueurs et d'actions finis, il existe au moins un équilibre de Nash.

3 Modèles basés sur les variables

3.1 Problèmes de satisfaction de contraintes

Dans cette section, notre but est de trouver des affectations de poids maximisants dans des problèmes impliquant des modèles basés sur les variables. Un avantage comparé aux modèles basés sur les états est que ces algorithmes sont plus commodes lorsqu'il s'agit de transcrire des contraintes spécifiques à certains problèmes.

3.1.1 Graphes de facteurs

Définition – Un graphe de facteurs, aussi appelé champ aléatoire de Markov, est un ensemble de variables $X = (X_1, \dots, X_n)$ où $X_i \in \text{Domain}_i$ muni de m facteurs f_1, \dots, f_m où chaque $f_j(X) \geq 0$.



Arité – Le nombre de variables dépendant d'un facteur f_j est appelé son arité.

*Remarque : les facteurs d'arité 1 et 2 sont respectivement appelés *unaire* et *binaire*.*

Affectation de poids – Chaque affectation $x = (x_1, \dots, x_n)$ donne un poids $\text{Weight}(x)$ défini comme étant le produit de tous les facteurs f_j appliqués à cette affectation. Son expression est donnée par :

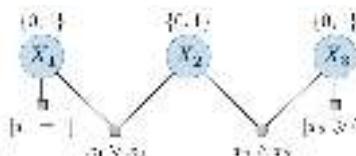
$$\boxed{\text{Weight}(x) = \prod_{j=1}^m f_j(x)}$$

Problème de satisfaction de contraintes – Un problème de satisfaction de contraintes (en anglais *constraint satisfaction problem* ou *CSP*) est un graphe de facteurs où tous les facteurs sont binaires ; on les appelle "contraintes".

$$\boxed{\forall j \in [1, m], \quad f_j(x) \in \{0, 1\}}$$

Ici, on dit que l'affectation x satisfait la contrainte j si et seulement si $f_j(x) = 1$.

Affectation consistante – Une affectation x d'un CSP est dite consistante si et seulement si $\text{Weight}(x) = 1$, i.e. toutes les contraintes sont satisfaites.



3.1.2 Mise en ordre dynamique

Facteurs dépendants – L'ensemble des facteurs dépendants de la variable X_i dont l'affectation partielle est x est appelé $D(x, X_i)$ et désigne l'ensemble des facteurs liant X_i à des variables déjà affectées.

Recherche avec retour sur trace – L'algorithme de recherche avec retour sur trace (en anglais *backtracking search*) est utilisé pour trouver l'affectation de poids maximum d'un graphe de facteurs. À chaque étape, une variable non assignée est choisie et ses valeurs sont explorées par récursivité. On peut utiliser un processus de mise en ordre dynamique sur le choix des variables et valeurs et/ou d'anticipation (i.e. élimination précoce d'options non consistantes) pour explorer le graphe de manière plus efficace. La complexité temporelle dans tous les cas reste néanmoins exponentielle : $O(|\text{Domain}|^n)$.

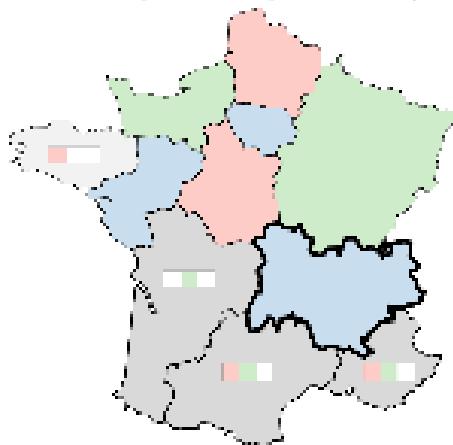
Vérification en avant – La vérification en avant (*forward checking* en anglais) est une heuristique d'anticipation à une étape qui enlève des variables voisines les valeurs impossibles de manière préemptive. Cette méthode a les caractéristiques suivantes :

- Après l'affectation d'une variable X_i , les valeurs non consistantes sont éliminées du domaine de tous ses voisins.
- Si l'un de ces domaines devient vide, la recherche locale s'arrête.
- Si l'on enlève l'affectation d'une valeur X_i , on doit restaurer le domaine de ses voisins.

Variable la plus contrainte – L'heuristique de la variable la plus contrainte (en anglais *most constrained variable* ou *MCV*) sélectionne la prochaine variable sans affectation ayant le moins de valeurs consistentes. Cette procédure a pour effet de faire échouer les affectations impossibles plus tôt dans la recherche, permettant un élagage plus efficace.

Valeur la moins contraignante – L'heuristique de la valeur la moins contraignante (en anglais *least constrained value* ou *LCV*) sélectionne pour une variable donnée la prochaine valeur maximisant le nombre de valeurs consistentes chez les variables voisines. De manière intuitive, on peut dire que cette procédure choisit en premier les valeurs qui sont le plus susceptible de marcher.

Remarque : en pratique, cette heuristique est utile quand tous les facteurs sont des contraintes.



L'exemple ci-dessus est une illustration du problème de coloration de graphe à 3 couleurs en utilisant l'algorithme de recherche avec retour sur trace couplé avec les heuristiques de MCV, de LCV ainsi que de vérification en avant à chaque étape.

Arc-consistance – On dit que l'arc-consistance de la variable X_l par rapport à X_k est vérifiée lorsque pour tout $x_l \in \text{Domain}_l$:

- les facteurs unaires de X_l sont non-nuls,
- il existe au moins un $x_k \in \text{Domain}_k$ tel que n'importe quel facteur entre X_l et X_k est non nul.

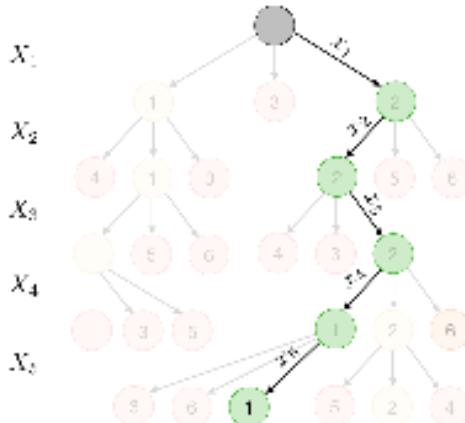
□ **AC-3** – L'algorithme d'AC-3 est une heuristique qui applique le principe de vérification en avant à toutes les variables susceptibles d'être concernées. Après l'affectation d'une variable, cet algorithme effectue une vérification en avant et applique successivement l'arc-consistance avec tous les voisins de variables pour lesquels le domaine change.

Remarque : AC-3 peut être codé de manière itérative ou récursive.

3.1.3 Méthodes approximatives

□ **Recherche en faisceau** – L'algorithme de recherche en faisceau (en anglais *beam search*) est une technique approximative qui étend les affectations partielles de n variables de facteur de branchement $b = |\text{Domain}|$ en explorant les K meilleurs chemins qui s'offrent à chaque étape. La largeur du faisceau $K \in \{1, \dots, b^n\}$ détermine la balance entre efficacité et précision de l'algorithme. Sa complexité en temps est de $O(n \cdot Kb \log(Kb))$.

L'exemple ci-dessous illustre une recherche en faisceau de paramètres $K = 2$, $b = 3$ et $n = 5$.



Remarque : $K = 1$ correspond à la recherche gloutonne alors que $K \rightarrow +\infty$ est équivalent à effectuer un parcours en largeur.

□ **Modes conditionnels itérés** – L'algorithme des modes conditionnels itérés (en anglais *iterated conditional modes* ou *ICM*) est une technique itérative et approximative qui modifie l'affectation d'un graphe de facteurs une variable à la fois jusqu'à convergence. À l'étape i , X_i prend la valeur v qui maximise le produit de tous les facteurs connectés à cette variable.

Remarque : il est possible qu'ICM reste bloqué dans un minimum local.

□ **Échantillonnage de Gibbs** – La méthode d'échantillonnage de Gibbs (en anglais *Gibbs sampling*) est une technique itérative et approximative qui modifie les affectations d'un graphe de facteurs une variable à la fois jusqu'à convergence. À l'étape i :

- on assigne à chaque élément $u \in \text{Domain}_i$ un poids $w(u)$ qui est le produit de tous les facteurs connectés à cette variable,
- on échantillonne v de la loi de probabilité engendrée par w et on l'associe à X_i .

Remarque : la méthode d'échantillonnage de Gibbs peut être vue comme étant la version probabiliste de ICM. Cette méthode a l'avantage de pouvoir échapper aux potentiels minimum locaux dans la plupart des situations.

3.1.4 Transformations sur les graphes de facteurs

□ Indépendance – Soit A, B une partition des variables X . On dit que A et B sont indépendants s'il n'y a pas d'arête connectant A et B et on écrit :

$$A \text{ et } B \text{ indépendants} \iff A \perp\!\!\!\perp B$$

Remarque : l'indépendance est une propriété importante car elle nous permet de décomposer la situation en sous-problèmes que l'on peut résoudre en parallèle.

□ Indépendance conditionnelle – On dit que A et B ont conditionnellement indépendants par rapport à C si le fait de conditionner sur C produit un graphe dans lequel A et B sont indépendants. Dans ce cas, on écrit :

$$A \text{ et } B \text{ cond. indép. par rapport à } C \iff A \perp\!\!\!\perp B | C$$

□ Conditionnement – Le conditionnement est une transformation visant à rendre des variables indépendantes et ainsi diviser un graphe de facteurs en pièces plus petites qui peuvent être traitées en parallèle et utiliser le retour sur trace. Pour conditionner par rapport à une variable $X_i = v$, on :

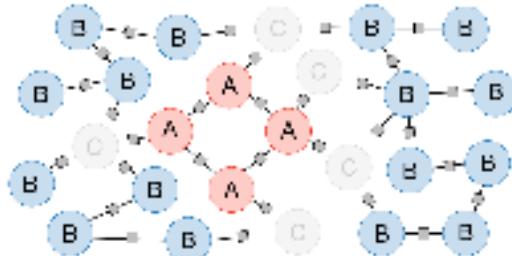
- considère toutes les facteurs f_1, \dots, f_k qui dépendent de X_i
- enlève X_i et f_1, \dots, f_k
- ajoute $g_j(x)$ pour $j \in \{1, \dots, k\}$ défini par :

$$g_j(x) = f_j(x \cup \{X_i : v\})$$

□ Couverture de Markov – Soit $A \subseteq X$ une partie des variables. On définit $\text{MarkovBlanket}(A)$ comme étant les voisins de A qui ne sont pas dans A .

□ Proposition – Soit $C = \text{MarkovBlanket}(A)$ et $B = X \setminus (A \cup C)$. On a alors :

$$A \perp\!\!\!\perp B | C$$



□ Élimination – L'élimination est une transformation consistant à enlever X_i d'un graphe de facteurs pour ensuite résoudre un sous-problème conditionné sur sa couverture de Markov où l'on :

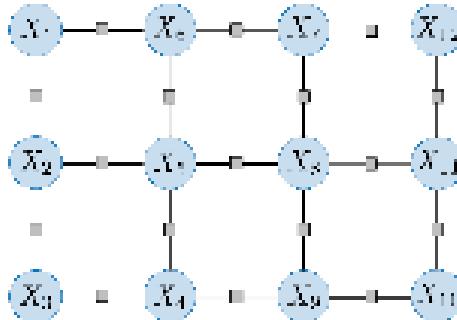
- considère tous les facteurs $f_{i,1}, \dots, f_{i,k}$ qui dépendent de X_i
- enlève X_i et $f_{i,1}, \dots, f_{i,k}$
- ajoute $f_{\text{new},i}(x)$ défini par :

$$f_{\text{new},i}(x) = \max_{x_i} \prod_{l=1}^k f_{i,l}(x)$$

□ **Largeur arborescente** – La largeur arborescente (en anglais *treenwidth*) d'un graphe de facteurs est l'arité maximum de n'importe quel facteur créé par élimination avec le meilleur ordre de variable. En d'autres termes,

$$\text{Treewidth} = \min_{\text{orderings}} \max_{i \in \{1, \dots, n\}} \text{arity}(f_{\text{new},i})$$

L'exemple ci-dessous illustre le cas d'un graphe de facteurs ayant une largeur arborescente égale à 3.



Remarque : trouver le meilleur ordre de variable est un problème NP-difficile.

3.2 Réseaux bayésiens

Dans cette section, notre but est de calculer des probabilités conditionnelles. Quelle est la probabilité d'un événement étant donné des observations ?

3.2.1 Introduction

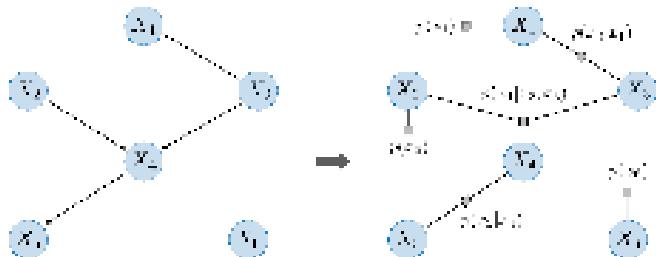
□ **Explication** – Supposons que les causes C_1 et C_2 influencent un effet E . Le conditionnement sur l'effet E et une des causes (disons C_1) change la probabilité de l'autre cause (disons C_2). Dans ce cas, on dit que C_1 a expliqué C_2 .

□ **Graphe orienté acyclique** – Un graphe orienté acyclique (en anglais *directed acyclic graph* ou *DAG*) est un graphe orienté fini sans cycle orienté.

□ **Réseau bayésien** – Un réseau bayésien (en anglais *Bayesian network*) est un DAG qui définit une loi de probabilité jointe sur les variables aléatoires $X = (X_1, \dots, X_n)$ comme étant le produit des lois de probabilités conditionnelles locales (une pour chaque noeud) :

$$P(X_1 = x_1, \dots, X_n = x_n) \triangleq \prod_{i=1}^n p(x_i | x_{\text{Parents}(i)})$$

Remarque : les réseaux bayésiens sont des graphes de facteurs imprégnés de concepts de probabilité.



□ Normalisation locale – Pour chaque $x_{\text{Parents}(i)}$, tous les facteurs sont localement des lois de probabilité conditionnelles. Elles doivent donc vérifier :

$$\sum_{x_i} p(x_i|x_{\text{Parents}(i)}) = 1$$

De ce fait, les sous-réseaux bayésiens et les distributions conditionnelles sont consistants.

Remarque : les lois locales de probabilité conditionnelles sont de vraies lois de probabilité conditionnelles.

□ Marginalisation – La marginalisation d'un nœud sans enfant entraîne un réseau bayésien sans ce nœud.

3.2.2 Programmes probabilistes

□ Concept – Un programme probabiliste rend aléatoire l'affectation de variables. De ce fait, on peut imaginer des réseaux bayésiens compliqués pour la génération d'affectations sans avoir à écrire de manière explicite les probabilités associées.

Remarque : quelques exemples de programmes probabilistes incluent parmi d'autres le modèle de Markov caché (en anglais hidden Markov model ou HMM), HMM factoriel, le modèle bayésien naïf (en anglais naive Bayes), l'allocation de Dirichlet latente (en anglais latent Dirichlet allocation ou LDA), le modèle à blocs stochastiques (en anglais stochastic block model).

□ Récapitulatif – La table ci-dessous résume les programmes probabilistes les plus fréquents ainsi que leur champ d'application associé :

Programme	Algorithm	Illustration	Exemple
Modèle de Markov	$X_i \sim p(X_i X_{i-1})$		Modélisation du langage
Modèle de Markov caché (HMM)	$H_t \sim p(H_t H_{t-1})$ $E_t \sim p(E_t H_t)$		Suivi d'objet

HMM factoriel	$H_t^o \sim_{\{a,b\}} p(H_t^o H_{t-1}^o)$ $E_t \sim p(E_t H_t^a, H_t^b)$		Suivi de plusieurs objets
Bayésien naïf	$Y \sim p(Y)$ $W_i \sim p(W_i Y)$		Classification de document
Allocation de Dirichlet latente (LDA)	$\alpha \in \mathbb{R}^K$ distribution $Z_i \sim p(Z_i \alpha)$ $W_i \sim p(W_i Z_i)$		Modélisation de sujet

3.2.3 Inférence

□ **Stratégie générale pour l'inférence probabiliste** – La stratégie que l'on utilise pour calculer la probabilité $P(Q|E = e)$ d'une requête Q étant donnée l'observation $E = e$ est la suivante :

- Étape 1 : on enlève les variables qui ne sont pas les ancêtres de la requête Q ou de l'observation E par marginalisation
- Étape 2 : on convertit le réseau bayésien en un graphe de facteurs
- Étape 3 : on conditionne sur l'observation $E = e$
- Étape 4 : on enlève les nœuds déconnectés de la requête Q par marginalisation
- Étape 5 : on lance un algorithme d'inférence probabiliste (manuel, élimination de variables, échantillonnage de Gibbs, filtrage particulaire)

□ **Algorithme progressif-rétrogressif** – L'algorithme progressif-rétrogressif (en anglais *forward-backward*) calcule la valeur exacte de $P(H = h_k | E = e)$ pour chaque $k \in \{1, \dots, L\}$ dans le cas d'un HMM de taille L . Pour ce faire, on procède en 3 étapes :

- Étape 1 : pour $i \in \{1, \dots, L\}$, calculer $F_i(h_i) = \sum_{h_{i-1}} F_{i-1}(h_{i-1})p(h_i|h_{i-1})p(e_i|h_i)$
- Étape 2 : pour $i \in \{L, \dots, 1\}$, calculer $B_i(h_i) = \sum_{h_{i+1}} B_{i+1}(h_{i+1})p(h_{i+1}|h_i)p(e_{i+1}|h_{i+1})$
- Étape 3 : pour $i \in \{1, \dots, L\}$, calculer $S_i(h_i) = \frac{F_i(h_i)B_i(h_i)}{\sum_{h_i} F_i(h_i)B_i(h_i)}$

avec la convention $F_0 = B_{L+1} = 1$. À partir de cette procédure et avec ces notations, on obtient

$$P(H = h_k | E = e) = S_k(h_k)$$

Remarque : cet algorithme interprète une affectation comme étant un chemin où chaque arête $h_{i-1} \rightarrow h_i$ a un poids $p(h_i|h_{i-1})p(e_i|h_i)$.

□ Échantillonnage de Gibbs – L'algorithme d'échantillonnage de Gibbs (en anglais *Gibbs sampling*) est une méthode itérative et approximative qui utilise un petit ensemble d'affectations (particules) pour représenter une loi de probabilité. Pour une affectation aléatoire x , l'échantillonnage de Gibbs effectue les étapes suivantes pour $i \in \{1, \dots, n\}$ jusqu'à convergence :

- Pour tout $u \in \text{Domain}_i$, on calcule le poids $w(u)$ de l'affectation x où $X_i = u$
- On échantillonne v de la loi de probabilité engendrée par $w : v \sim P(X_i = v | X_{-i} = x_{-i})$
- On pose $X_i = v$

Remarque : X_{-i} veut dire $X \setminus \{X_i\}$ et x_{-i} représente l'affectation correspondante.

□ Filtrage particulaire – L'algorithme de filtrage particulaire (en anglais *particle filtering*) approxime la densité postérieure de variables d'états à partir des variables observées en suivant K particules à la fois. En commençant avec un ensemble de particules C de taille K , on répète les 3 étapes suivantes :

- Étape 1 : proposition - Pour chaque particule $x_{t-1} \in C$, on échantillonne x avec loi de probabilité $p(x|x_{t-1})$ et on ajoute x à un ensemble C' .
- Étape 2 : pondération - On associe chaque x de l'ensemble C' au poids $w(x) = p(e_t|x)$, où e_t est l'observation vue à l'instant t .
- Étape 3 : é-échantillonnage - On échantillonne K éléments de l'ensemble C' en utilisant la loi de probabilité engendrée par w et on les met dans C : ce sont les particules courantes x_t .

Remarque : une version plus coûteuse de cet algorithme tient aussi compte des particules passée à l'étape de proposition.

□ Maximum de vraisemblance – Si l'on ne connaît pas les lois de probabilité locales, on peut les trouver en utilisant le maximum de vraisemblance.

$$\max_{\theta} \prod_{x \in \mathcal{D}_{\text{train}}} p(X = x; \theta)$$

□ Lissage de Laplace – Pour chaque loi de probabilité d et affectation partielle $(x_{\text{Parents}(i)}, x_i)$, on ajoute λ à $\text{count}_d(x_{\text{Parents}(i)}, x_i)$ et on normalise ensuite pour obtenir des probabilités.

□ Espérance-maximisation – L'algorithme d'espérance-maximisation (en anglais *expectation-maximization* ou *EM*) est une méthode efficace utilisée pour estimer le paramètre θ via l'estimation du maximum de vraisemblance en construisant de manière répétée une borne inférieure de la vraisemblance (étape E) et en optimisant cette borne inférieure (étape M) :

- Étape E : on évalue la probabilité postérieure $q(h)$ que chaque point e vienne d'une partition particulière h avec :

$$q(h) = P(H = h | E = e; \theta)$$

- Étape M : on utilise la probabilité postérieure $q(h)$ en tant que poids de la partition h sur les points e pour déterminer θ through via le maximum de vraisemblance.

4 Modèles basés sur la logique

4.1 Bases

□ **Syntaxe de la logique propositionnelle** – En notant f et g formules et $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ opérateurs, on peut écrire les expressions logiques suivantes :

Nom	Symbole	Signification	Illustration
Affirmation	f	f	
Négation	$\neg f$	non f	
Conjonction	$f \wedge g$	f et g	
Disjonction	$f \vee g$	f ou g	
Implication	$f \rightarrow g$	si f alors g	
Biconditionnel	$f \leftrightarrow g$	f , c'est à dire g	

Remarque : n'importe quelle formule peut être construite de manière récursive à partir de ces opérateurs.

□ **Modèle** – Un modèle w dénote une combinaison de valeurs binaires liées à des symboles propositionnels.

Exemple : l'ensemble de valeurs de vérité $w = \{A : 0, B : 1, C : 0\}$ est un modèle possible pour les symboles propositionnels A , B and C .

□ **Interprétation** – L'interprétation $\mathcal{I}(f,w)$ outputs whether model w satisfies formula f :

$$\boxed{\mathcal{I}(f,w) \in \{0,1\}}$$

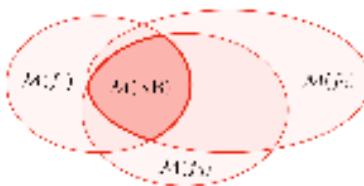
□ **Ensemble de modèles** – $\mathcal{M}(f)$ dénote l'ensemble des modèles w qui satisfont la formule f . Sa définition mathématique est donnée par :

$$\boxed{\forall w \in \mathcal{M}(f), \quad \mathcal{I}(f, w) = 1}$$

4.2 Base de connaissance

□ **Définition** – La base de connaissance KB est la conjonction de toutes les formules considérées jusqu'à présent. L'ensemble des modèles de la base de connaissance est l'intersection de l'ensemble des modèles satisfaisant chaque formule. En d'autres termes :

$$\boxed{\mathcal{M}(\text{KB}) = \bigcap_{f \in \text{KB}} \mathcal{M}(f)}$$



□ **Interprétation en termes de probabilités** – La probabilité que la requête f soit évaluée à 1 peut être vue comme la proportion des modèles w de la base de connaissance KB qui satisfait f , i.e. :

$$\boxed{P(f|\text{KB}) = \frac{\sum_{w \in \mathcal{M}(\text{KB}) \cap \mathcal{M}(f)} P(W=w)}{\sum_{w \in \mathcal{M}(\text{KB})} P(W=w)}}$$

□ **Satisfaisabilité** – La base de connaissance KB est dite satisfaisable si au moins un modèle w satisfait toutes ses contraintes. En d'autres termes :

$$\boxed{\text{KB satisfaisable} \iff \mathcal{M}(\text{KB}) \neq \emptyset}$$

Remarque : $\mathcal{M}(\text{KB})$ dénote l'ensemble des modèles compatibles avec toutes les contraintes de la base de connaissance.

□ **Relation entre formules et base de connaissance** – On définit les propriétés suivantes entre la base de connaissance KB et une nouvelle formule f :

Nom	Formulation mathématique	Illustration	Notes
KB déduit f	$\mathcal{M}(\text{KB}) \cap \mathcal{M}(f) = \mathcal{M}(\text{KB})$		- f n'apporte aucune nouvelle information - Aussi écrit $\text{KB} \models f$
KB contredit f	$\mathcal{M}(\text{KB}) \cap \mathcal{M}(f) = \emptyset$		- Aucun modèle ne satisfait les contraintes après l'ajout de f - Équivalent à $\text{KB} \models \neg f$
f est contingent à KB	$\mathcal{M}(\text{KB}) \cap \mathcal{M}(f) \neq \emptyset$ et $\mathcal{M}(\text{KB}) \cap \mathcal{M}(f) \neq \mathcal{M}(\text{KB})$		- f ne contredit pas KB - f ajoute une quantité d'information non triviale à KB

□ **Vérification de modèles** – Un algorithme de vérification de modèles (*model checking* en anglais) prend comme argument une base de connaissance KB et nous renseigne si celle-ci est satisfaisable ou pas.

Remarque : DPLL et WalkSat sont des exemples populaires d'algorithmes de vérification de modèles.

□ **Règle d'inférence** – Une règle d'inférence de prémisses f_1, \dots, f_k et de conclusion g s'écrit :

$$\frac{f_1, \dots, f_k}{g}$$

□ **Algorithme de chaînage avant** – Partant d'un ensemble de règles d'inférence Rules, l'algorithme de chaînage avant (en anglais *forward inference algorithm*) parcourt tous les f_1, \dots, f_k et ajoute g à la base de connaissance KB si une règle parvient à une telle conclusion. Cette démarche est répétée jusqu'à ce qu'aucun autre ajout ne puisse être fait à KB.

□ **Dérivation** – On dit que KB dérive f (noté $\text{KB} \vdash f$) par le biais des règles Rules soit si f est déjà dans KB ou si elle se fait ajouter pendant l'application du chaînage avant utilisant les règles Rules.

□ **Propriétés des règles d'inférence** – Un ensemble de règles d'inférence Rules peut avoir les propriétés suivantes :

Name	Formulation mathématique	Notes
Validité	$\{f : \text{KB} \vdash f\} \subseteq \{f : \text{KB} \models f\}$	- Les formules inférées sont déduites par KB - Peut être vérifiée une règle à la fois - "Rien que la vérité"
Complétude	$\{f : \text{KB} \vdash f\} \supseteq \{f : \text{KB} \models f\}$	- Les formules déduites par KB sont soit déjà dans la base de connaissance, soit inférées de celle-ci - "La vérité dans sa totalité"

4.3 Logique propositionnelle

Dans cette section, nous allons parcourir les modèles logiques utilisant des formules logiques et des règles d'inférence. L'idée est de trouver le juste milieu entre expressivité et efficacité.

□ Clause de Horn – En notant p_1, \dots, p_k et q des symboles propositionnels, une clause de Horn s'écrit :

$$(p_1 \wedge \dots \wedge p_k) \longrightarrow q$$

Remarque : quand $q = \text{false}$, cette clause de Horn est "négative", autrement elle est appelée "stricte".

□ Modus ponens – Sur les symboles propositionnels f_1, \dots, f_k et p , la règle de modus ponens est écrite :

$$\frac{f_1, \dots, f_k, \quad (f_1 \wedge \dots \wedge f_k) \longrightarrow p}{p}$$

Remarque : l'application de cette règle se fait en temps linéaire, puisque chaque exécution génère une clause contenant un symbole propositionnel.

□ Complétude – Modus ponens est complet lorsqu'on le munit des clauses de Horn si l'on suppose que KB contient uniquement des clauses de Horn et que p est un symbole propositionnel qui est déduit. L'application de modus ponens dérivera alors p .

□ Forme normale conjonctive – La forme normale conjonctive (en anglais *conjunctive normal form* ou *CNF*) d'une formule est une conjonction de clauses, chacune d'entre elles étant une disjonction de formules atomiques.

Remarque : en d'autres termes, les CNFs sont des \wedge de \vee .

□ Représentation équivalente – Chaque formule en logique propositionnelle peut être écrite de manière équivalente sous la forme d'une formule CNF. Le tableau ci-dessous présente les propriétés principales permettant une telle conversion :

Nom de la règle		Début	Résultat
Élimine	\leftrightarrow	$f \leftrightarrow g$	$(f \rightarrow g) \wedge (g \rightarrow f)$
	\rightarrow	$f \rightarrow g$	$\neg f \vee g$
	$\neg\neg$	$\neg\neg f$	f
Distribue	\neg sur \wedge	$\neg(f \wedge g)$	$\neg f \vee \neg g$
	\neg sur \vee	$\neg(f \vee g)$	$\neg f \wedge \neg g$
	\vee sur \wedge	$f \vee (g \wedge h)$	$(f \vee g) \wedge (f \vee h)$

□ Règle de résolution – Pour des symboles propositionnels f_1, \dots, f_n , et g_1, \dots, g_m ainsi que p , la règle de résolution s'écrit :

$$\frac{f_1 \vee \dots \vee f_n \vee p, \quad \neg p \vee g_1 \vee \dots \vee g_m}{f_1 \vee \dots \vee f_n \vee g_1 \vee \dots \vee g_m}$$

Remarque : l'application de cette règle peut prendre un temps exponentiel, vu que chaque itération génère une clause constituée d'une partie des symboles propositionnels.

□ Inférence basée sur la règle de résolution – L'algorithme d'inférence basée sur la règle de résolution se déroule en plusieurs étapes :

- Étape 1 : Conversion de toutes les formules vers leur forme CNF
- Étape 2 : Application répétée de la règle de résolution
- Étape 3 : Renvoyer "non satisfaisable" si et seulement si False est dérivé

4.4 Calcul des prédictats du premier ordre

L'idée ici est d'utiliser des variables et ainsi permettre une représentation des connaissances plus compacte.

□ Modèle – Un modèle w en calcul des prédictats du premier ordre lie :

- des symboles constants à des objets
- des prédictats à n -uplets d'objets

□ Clause de Horn – En notant x_1, \dots, x_n variables et a_1, \dots, a_k, b formules atomiques, une clause de Horn pour le calcul des prédictats du premier ordre a la forme :

$$\boxed{\forall x_1, \dots, \forall x_n, (a_1 \wedge \dots \wedge a_k) \rightarrow b}$$

□ Substitution – Une substitution θ lie les variables aux termes et $\text{Subst}(\theta, f)$ désigne le résultat de la substitution θ sur f .

□ Unification – Une unification prend deux formules f et g et renvoie la substitution θ la plus générale les rendant égales :

$$\boxed{\text{Unify}[f, g] = \theta \quad \text{t.q.} \quad \text{Subst}[\theta, f] = \text{Subst}[\theta, g]}$$

Note : $\text{Unify}[f, g]$ renvoie Fail si un tel θ n'existe pas.

□ Modus ponens – En notant x_1, \dots, x_n variables, a_1, \dots, a_k et a'_1, \dots, a'_k formules atomiques et en notant $\theta = \text{Unify}(a'_1 \wedge \dots \wedge a'_k, a_1 \wedge \dots \wedge a_k)$, modus ponens pour le calcul des prédictats du premier ordre s'écrit :

$$\boxed{\frac{a'_1, \dots, a'_k \quad \forall x_1, \dots, \forall x_n (a_1 \wedge \dots \wedge a_k) \rightarrow b}{\text{Subst}[\theta, b]}}$$

□ Complétude – Modus ponens est complet pour le calcul des prédictats du premier ordre lorsqu'il agit uniquement sur les clauses de Horn.

□ Règle de résolution – En notant $f_1, \dots, f_n, g_1, \dots, g_m, p, q$ formules et en posant $\theta = \text{Unify}(p, q)$, la règle de résolution pour le calcul des prédictats du premier ordre s'écrit :

$$\boxed{\frac{f_1 \vee \dots \vee f_n \vee p, \neg q \vee g_1 \vee \dots \vee g_m}{\text{Subst}[\theta, f_1 \vee \dots \vee f_n \vee g_1 \vee \dots \vee g_m]}}$$

□ Semi-décidabilité – Le calcul des prédictats du premier ordre, même restreint aux clauses de Horn, n'est que semi-décidable.

- si $\text{KB} \models f$, l'algorithme de chaînage avant sur des règles d'inférence complètes prouvera f en temps fini
- si $\text{KB} \not\models f$, aucun algorithme ne peut le prouver en temps fini

Super Pense-bête VIP : Machine Learning

Afshine AMIDI et Shervine AMIDI

6 octobre 2018

Table des matières

1	Apprentissage supervisé	2
1.1	Introduction à l'apprentissage supervisé	2
1.2	Notations et concepts généraux	2
1.2.1	Régression linéaire	3
1.2.2	Classification et régression logistique	3
1.2.3	Modèles linéaires généralisés	3
1.3	Support Vector Machines	3
1.4	Apprentissage génératif	4
1.4.1	Gaussian Discriminant Analysis	4
1.4.2	Naive Bayes	4
1.5	Méthode à base d'arbres et d'ensembles	4
1.6	Autres approches non-paramétriques	5
1.7	Théorie d'apprentissage	5
2	Apprentissage non-supervisé	6
2.1	Introduction à l'apprentissage non-supervisé	6
2.2	Partitionnement	6
2.2.1	Espérance-Maximisation	6
2.2.2	Partitionnement k -means	6
2.2.3	Regroupement hiérarchique	6
2.2.4	Indicateurs d'évaluation de clustering	6
2.3	Réduction de dimension	7
2.3.1	Analyse des composantes principales	7
2.3.2	Analyse en composantes indépendantes	7
3	Apprentissage profond	8
3.1	Réseau de neurones	8
3.2	Réseaux de neurones convolutionnels	8
3.3	Réseaux de neurones récurrents	8
3.4	Reinforcement Learning	9

4 Astuces de Machine Learning	10
4.1 Indicateurs dans le contexte de la classification	10
4.2 Indicateurs dans le contexte de la régression	10
4.3 Sélection de modèle	10
4.4 Diagnostics	11
5 Rappels	12
5.1 Probabilités et Statistiques	12
5.1.1 Introduction aux probabilités à l'analyse combinatoire	12
5.1.2 Probabilité conditionnelle	12
5.1.3 Variable aléatoires	13
5.1.4 Variables aléatoires conjointement distribuées	13
5.1.5 Estimation des paramètres	14
5.2 Algèbre linéaire et Analyse	14
5.2.1 Notations générales	14
5.2.2 Opérations matricielles	15
5.2.3 Propriétés matricielles	15
5.2.4 Analyse matricielle	16

1 Apprentissage supervisé

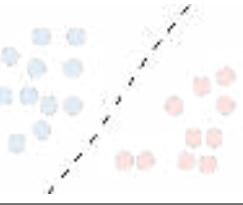
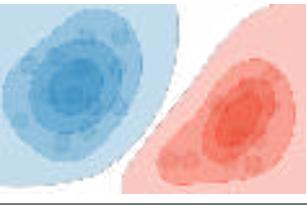
1.1 Introduction à l'apprentissage supervisé

Étant donné un ensemble de points $\{x^{(1)}, \dots, x^{(m)}\}$ associés à un ensemble d'issues $\{y^{(1)}, \dots, y^{(m)}\}$, on veut construire un classifieur qui apprend à prédire y depuis x .

□ Type de prédition – Les différents types de modèle prédictifs sont résumés dans le tableau ci-dessous :

	Régression	Classifieur
Issue	Continu	Classe
Exemples	Régression linéaire	Régression logistique, SVM, Naive Bayes

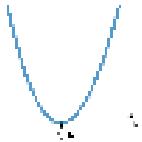
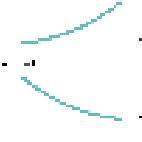
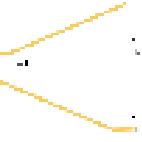
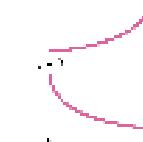
□ Type de modèle – Les différents modèles sont présentés dans le tableau ci-dessous :

	Modèle discriminatif	Modèle génératif
But	Estimer directement $P(y x)$	Estimer $P(x y)$ puis déduire $P(y x)$
Ce qui est appris	Frontière de décision	Distribution de proba des données
Illustration		
Exemples	Régressions, SVMs	GDA, Naive Bayes

1.2 Notations et concepts généraux

□ Hypothèse – Une hypothèse est notée h_θ et est le modèle que l'on choisit. Pour une entrée donnée $x^{(i)}$, la prédiction donnée par le modèle est $h_\theta(x^{(i)})$.

□ Fonction de loss – Une fonction de loss est une fonction $L : (z,y) \in \mathbb{R} \times Y \mapsto L(z,y) \in \mathbb{R}$ prenant comme entrée une valeur prédictée z correspondant à une valeur réelle y , et nous renseigne sur la ressemblance de ces deux valeurs. Les fonctions de loss courantes sont récapitulées dans le tableau ci-dessous :

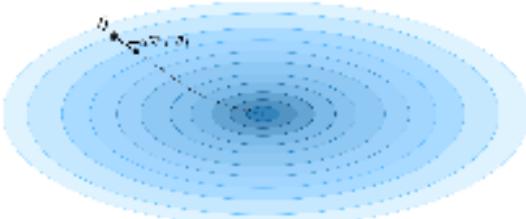
Moindres carrés	Logistique	Hinge loss	Cross-entropie
$\frac{1}{2}(y - z)^2$	$\log(1 + \exp(-yz))$	$\max(0, 1 - yz)$	$-\left[y \log(z) + (1 - y) \log(1 - z)\right]$
			
Régression linéaire	Régression logistique	SVM	Réseau de neurones

□ **Fonction de coût** – La fonction de coût J est communément utilisée pour évaluer la performance d'un modèle, et est définie avec la fonction de loss L par :

$$J(\theta) = \sum_{i=1}^m L(h_\theta(x^{(i)}), y^{(i)})$$

□ **Algorithme du gradient** – En notant $\alpha \in \mathbb{R}$ le taux d'apprentissage (en anglais *learning rate*), la règle de mise à jour de l'algorithme est exprimée en fonction du taux d'apprentissage et de la fonction de cost J de la manière suivante :

$$\theta \leftarrow \theta - \alpha \nabla J(\theta)$$



Remarque : L'algorithme du gradient stochastique (en anglais SGD - Stochastic Gradient Descent) met à jour le paramètre à partir de chaque élément du jeu d'entraînement, tandis que l'algorithme du gradient de batch le fait sur chaque lot d'exemples.

□ **Vraisemblance** – La vraisemblance d'un modèle $L(\theta)$ de paramètre θ est utilisée pour trouver le paramètre optimal θ par le biais du maximum de vraisemblance. En pratique, on utilise la log vraisemblance $\ell(\theta) = \log(L(\theta))$ qui est plus facile à optimiser. On a :

$$\theta^{\text{opt}} = \arg \max_{\theta} L(\theta)$$

□ **Algorithme de Newton** – L'algorithme de Newton est une méthode numérique qui trouve θ tel que $\ell'(\theta) = 0$. La règle de mise à jour est :

$$\theta \leftarrow \theta - \frac{\ell'(\theta)}{\ell''(\theta)}$$

Remarque : la généralisation multidimensionnelle, aussi connue sous le nom de la méthode de Newton-Raphson, a la règle de mise à jour suivante :

$$\theta \leftarrow \theta - (\nabla_{\theta}^2 \ell(\theta))^{-1} \nabla_{\theta} \ell(\theta)$$

1.2.1 Régression linéaire

On suppose ici que $y|x; \theta \sim \mathcal{N}(\mu, \sigma^2)$

□ **Équations normales** – En notant X la matrice de design, la valeur de θ qui minimize la fonction de coût a une solution de forme fermée tel que :

$$\boxed{\theta = (X^T X)^{-1} X^T y}$$

□ **Algorithme LMS** – En notant α le taux d'apprentissage, la règle de mise à jour d'algorithme des moindres carrés (LMS) pour un jeu de données d'entraînement de m points, aussi connu sous le nom de règle de Widrow-Hoff, est donné par :

$$\boxed{\forall j, \quad \theta_j \leftarrow \theta_j + \alpha \sum_{i=1}^m [y^{(i)} - h_{\theta}(x^{(i)})] x_j^{(i)}}$$

Remarque : la règle de mise à jour est un cas particulier de l'algorithme du gradient.

□ **LWR** – Locally Weighted Regression, souvent noté LWR, est une variante de la régression linéaire appliquant un coefficient à chaque exemple dans sa fonction de coût via $w^{(i)}(x)$, qui est défini avec un paramètre $\tau \in \mathbb{R}$ de la manière suivante :

$$\boxed{w^{(i)}(x) = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)}$$

1.2.2 Classification et régression logistique

□ **Sigmoïde** – La sigmoïde g , aussi connue sous le nom de fonction logistique, est définie par :

$$\boxed{\forall z \in \mathbb{R}, \quad g(z) = \frac{1}{1 + e^{-z}} \in]0, 1[}$$

□ **Régression logistique** – On suppose ici que $y|x; \theta \sim \text{Bernoulli}(\phi)$. On a la forme suivante :

$$\boxed{\phi = p(y=1|x; \theta) = \frac{1}{1 + \exp(-\theta^T x)} = g(\theta^T x)}$$

Remarque : il n'y a pas de solution fermée dans le cas de la régression logistique.

□ **Régression softmax** – Une régression softmax, aussi appelée un régression logistique multi-classe, est utilisée pour généraliser la régression logistique lorsqu'il y a plus de 2 classes à prédire. Par convention, on fixe $\theta_K = 0$, ce qui oblige le paramètre de Bernoulli ϕ_i de chaque classe i à être égal à :

$$\boxed{\phi_i = \frac{\exp(\theta_i^T x)}{\sum_{j=1}^K \exp(\theta_j^T x)}}$$

1.2.3 Modèles linéaires généralisés

□ Famille exponentielle – Une classe de distributions est issue de la famille exponentielle lorsqu'elle peut être écrite en termes d'un paramètre naturel, aussi appelé paramètre canonique ou fonction de lien η , d'une statistique suffisante $T(y)$ et d'une fonction de log-partition $a(\eta)$ de la manière suivante :

$$p(y; \eta) = b(y) \exp(\eta T(y) - a(\eta))$$

Remarque : on aura souvent $T(y) = y$. Aussi, $\exp(-a(\eta))$ peut être vu comme un paramètre de normalisation s'assurant que les probabilités somment à un.

Les distributions exponentielles les plus communément rencontrées sont récapitulées dans le tableau ci-dessous :

Distribution	η	$T(y)$	$a(\eta)$	$b(y)$
Bernoulli	$\log\left(\frac{\phi}{1-\phi}\right)$	y	$\log(1 + \exp(\eta))$	1
Gaussian	μ	y	$\frac{\eta^2}{2}$	$\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{y^2}{2}\right)$
Poisson	$\log(\lambda)$	y	e^η	$\frac{1}{y!}$
Geometric	$\log(1 - \phi)$	y	$\log\left(\frac{e^\eta}{1 - e^\eta}\right)$	1

□ Hypothèses pour les GLMs – Les modèles linéaires généralisés (GLM) ont pour but de prédire une variable aléatoire y comme une fonction de $x \in \mathbb{R}^{n+1}$ et reposent sur les 3 hypothèses suivantes :

$$(1) \quad y|x; \theta \sim \text{ExpFamily}(\eta) \quad (2) \quad h_\theta(x) = E[y|x; \theta] \quad (3) \quad \eta = \theta^T x$$

Remarque : la méthode des moindres carrés ordinaires et la régression logistique sont des cas spéciaux des modèles linéaires généralisés.

1.3 Support Vector Machines

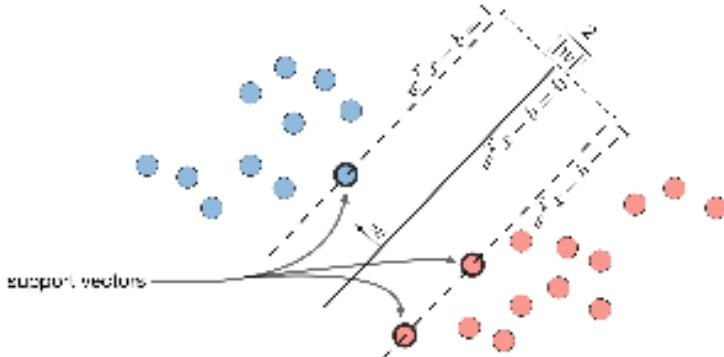
Le but des support vector machines est de trouver la ligne qui maximise la distance minimum à la ligne.

□ Classifieur à marges optimales – Le classifieur à marges optimales h est tel que :

$$h(x) = \text{sign}(w^T x - b)$$

où $(w, b) \in \mathbb{R}^n \times \mathbb{R}$ est une solution du problème d'optimisation suivant :

$$\min \frac{1}{2} \|w\|^2 \quad \text{tel que} \quad y^{(i)}(w^T x^{(i)} - b) \geq 1$$



Remarque : la ligne est définie par $w^T x - b = 0$.

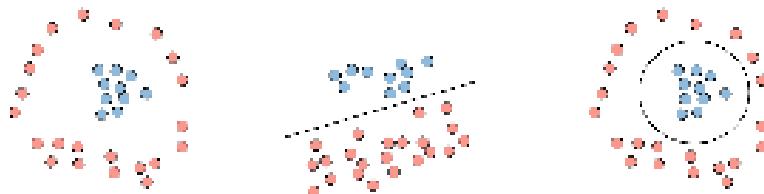
□ **Hinge loss** – Le hinge loss est utilisé dans le cadre des SVMs et est défini de la manière suivante :

$$L(z,y) = [1 - yz]_+ = \max(0, 1 - yz)$$

□ **Noyau** – Étant donné un feature mapping ϕ , on définit le noyau K par :

$$K(x,z) = \phi(x)^T \phi(z)$$

En pratique, le noyau K défini par $K(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$ est nommé noyau gaussien et est communément utilisé.



Separation non linéaire \longrightarrow Mapping de noyau ϕ \longrightarrow Ligne de décision dans l'espace initial

Remarque : on dit que l'on utilise "l'astuce du noyau" (en anglais kernel trick) pour calculer la fonction de coût en utilisant le noyau parce qu'il se trouve que l'on n'a pas besoin de trouver le mapping explicite, qui est souvent compliqué. Il suffit de connaître les valeurs de $K(x,z)$.

□ **Lagrangien** – On définit le lagrangien $L(w,b)$ par :

$$\mathcal{L}(w,b) = f(w) + \sum_{i=1}^l \beta_i h_i(w)$$

Remarque : les coefficients β_i sont appelés les multiplicateurs de Lagrange.

1.4 Apprentissage génératif

Un modèle génératif essaie d'abord d'apprendre comment les données sont générées en estimant $P(x|y)$, nous permettant ensuite d'estimer $P(y|x)$ par le biais du théorème de Bayes.

1.4.1 Gaussian Discriminant Analysis

Cadre – Le Gaussian Discriminant Analysis suppose que y et $x|y = 0$ et $x|y = 1$ sont tels que :

$$y \sim \text{Bernoulli}(\phi)$$

$$x|y = 0 \sim \mathcal{N}(\mu_0, \Sigma) \quad \text{et} \quad x|y = 1 \sim \mathcal{N}(\mu_1, \Sigma)$$

Estimation – Le tableau suivant récapitule les estimations que l'on a trouvées lors de la maximisation de la vraisemblance :

$\hat{\phi}$	$\hat{\mu}_j \quad (j = 0, 1)$	$\hat{\Sigma}$
$\frac{1}{m} \sum_{i=1}^m \mathbf{1}_{\{y^{(i)}=1\}}$	$\frac{\sum_{i=1}^m \mathbf{1}_{\{y^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^m \mathbf{1}_{\{y^{(i)}=j\}}}$	$\frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T$

1.4.2 Naive Bayes

Hypothèse – Le modèle de Naive Bayes suppose que les caractéristiques de chaque point sont toutes indépendantes :

$$P(x|y) = P(x_1, x_2, \dots | y) = P(x_1|y)P(x_2|y)\dots = \prod_{i=1}^n P(x_i|y)$$

Solutions – Maximiser la log vraisemblance donne les solutions suivantes, où $k \in \{0, 1\}, l \in [1, L]$

$$P(y = k) = \frac{1}{m} \times \#\{j | y^{(j)} = k\}$$

et

$$P(x_i = l | y = k) = \frac{\#\{j | y^{(j)} = k \text{ et } x_i^{(j)} = l\}}{\#\{j | y^{(j)} = k\}}$$

Remarque : Naive Bayes est couramment utilisé pour la classification de texte et pour la détection de spams.

1.5 Méthode à base d'arbres et d'ensembles

Ces méthodes peuvent être utilisées pour des problèmes de régression et de classification.

CART – Les arbres de classification et de régression (en anglais *CART - Classification And Regression Trees*), aussi connus sous le nom d'arbres de décision, peuvent être représentés sous la forme d'arbres binaires. Ils ont l'avantage d'être très interprétables.

□ Random forest – C'est une technique à base d'arbres qui utilise un très grand nombre d'arbres de décisions construits à partir d'ensembles de caractéristiques aléatoirement sélectionnées. Contrairement à un simple arbre de décision, il n'est pas interprétable du tout mais le fait qu'il ait une bonne performance en fait un algorithme populaire.

Remarque : les random forests sont un type de méthode ensembliste.

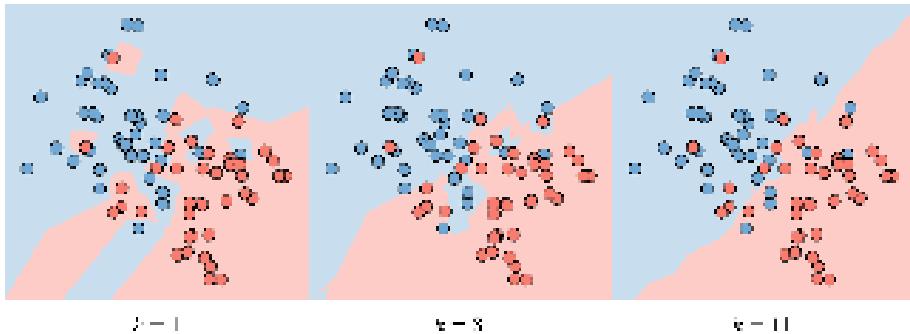
□ Boosting – L'idée des méthodes de boosting est de combiner plusieurs modèles faibles pour former un modèle meilleur. Les principales méthodes de boosting sont récapitulées dans le tableau ci-dessous :

Boosting adaptatif	Boosting par gradient
<ul style="list-style-type: none"> - De grands coefficients sont mis sur les erreurs pour s'améliorer à la prochaine étape de boosting - Connue sous le nom d'Adaboost 	<ul style="list-style-type: none"> - Les modèles faibles sont entraînés sur les erreurs résiduelles

1.6 Autres approches non-paramétriques

□ *k*-nearest neighbors – L'algorithme des k plus proches voisins (en anglais *k-nearest neighbors*), aussi connu sous le nom de *k*-NN, est une approche non-paramétrique où la réponse d'un point est déterminée par la nature de ses k voisins du jeu de données d'entraînement. Il peut être utilisé dans des cadres de classification et de régression.

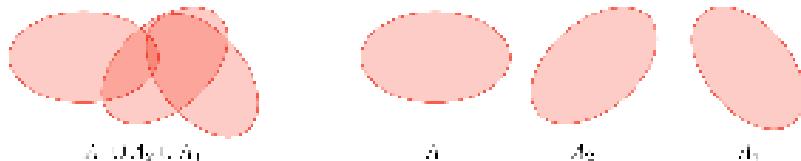
Remarque : Plus le paramètre k est élevé, plus le biais est élevé, et plus le paramètre k est faible, plus la variance est élevée.



1.7 Théorie d'apprentissage

□ Inégalité de Boole – Soit A_1, \dots, A_k k événements. On a :

$$P(A_1 \cup \dots \cup A_k) \leq P(A_1) + \dots + P(A_k)$$



□ Inégalité d'Hoeffding – Soit Z_1, \dots, Z_m m variables iid tirées d'une distribution de Bernoulli de paramètre ϕ . Soit $\hat{\phi}$ leur moyenne empirique et $\gamma > 0$ fixé. On a :

$$P(|\phi - \hat{\phi}| > \gamma) \leq 2 \exp(-2\gamma^2 m)$$

Remarque : cette inégalité est aussi connue sous le nom de borne de Chernoff.

□ Erreur de training – Pour un classifieur donné h , on définit l'erreur d'entraînement $\widehat{\epsilon}(h)$, aussi connu sous le nom de risque empirique ou d'erreur empirique, par :

$$\widehat{\epsilon}(h) = \frac{1}{m} \sum_{i=1}^m 1_{\{h(x^{(i)}) \neq y^{(i)}\}}$$

□ Probablement Approximativement Correct (PAC) – PAC est un cadre dans lequel de nombreux résultats d'apprentissages ont été prouvés, et contient l'ensemble d'hypothèses suivant :

- les jeux d'entraînement et de test suivent la même distribution
- les exemples du jeu d'entraînement sont tirés indépendamment

□ Éclatement – Étant donné un ensemble $S = \{x^{(1)}, \dots, x^{(d)}\}$, et un ensemble de classifieurs \mathcal{H} , on dit que \mathcal{H} brise S si pour tout ensemble de labels $\{y^{(1)}, \dots, y^{(d)}\}$, on a :

$$\exists h \in \mathcal{H}, \quad \forall i \in [1, d], \quad h(x^{(i)}) = y^{(i)}$$

□ Théorème de la borne supérieure – Soit \mathcal{H} une hypothèse finie de classe telle que $|\mathcal{H}| = k$, soit δ , et soit m la taille fixée d'un échantillon. Alors, avec une probabilité d'au moins $1 - \delta$, on a :

$$\epsilon(\widehat{h}) \leq \left(\min_{h \in \mathcal{H}} \epsilon(h) \right) + 2 \sqrt{\frac{1}{2m} \log \left(\frac{2k}{\delta} \right)}$$

□ Dimension VC – La dimension de Vapnik-Chervonenkis (VC) d'une classe d'hypothèses de classes infinies donnée \mathcal{H} , que l'on note $\text{VC}(\mathcal{H})$, est la taille de l'ensemble le plus grand qui est brisé par \mathcal{H} .

Remarque : la dimension VC de $\mathcal{H} = \{\text{set of linear classifiers in 2 dimensions}\}$ est égale à 3.



□ Théorème (Vapnik) – Soit \mathcal{H} donné, avec $\text{VC}(\mathcal{H}) = d$ avec m le nombre d'exemples d'entraînement. Avec une probabilité d'au moins $1 - \delta$, on a :

$$\epsilon(\widehat{h}) \leq \left(\min_{h \in \mathcal{H}} \epsilon(h) \right) + O \left(\sqrt{\frac{d}{m} \log \left(\frac{m}{d} \right)} + \frac{1}{m} \log \left(\frac{1}{\delta} \right) \right)$$

2 Apprentissage non-supervisé

2.1 Introduction à l'apprentissage non-supervisé

□ Motivation – Le but de l'apprentissage non-supervisé est de trouver des formes cachées dans un jeu de données non-labelées $\{x^{(1)}, \dots, x^{(m)}\}$.

□ Inégalité de Jensen – Soit f une fonction convexe et X une variable aléatoire. On a l'inégalité suivante :

$$E[f(X)] \geq f(E[X])$$

2.2 Partitionnement

2.2.1 Espérance-Maximisation

□ Variables latentes – Les variables latentes sont des variables cachées/non-observées qui posent des difficultés aux problèmes d'estimation, et sont souvent notées z . Voici les cadres dans lesquelles les variables latentes sont le plus fréquemment utilisées :

Cadre	Variance latente z	$x z$	Commentaires
Mixture de k gaussiennes	Multinomial(ϕ)	$\mathcal{N}(\mu_j, \Sigma_j)$	$\mu_j \in \mathbb{R}^n, \phi \in \mathbb{R}^k$
Analyse factorielle	$\mathcal{N}(0, I)$	$\mathcal{N}(\mu + \Lambda z, \psi)$	$\mu_j \in \mathbb{R}^n$

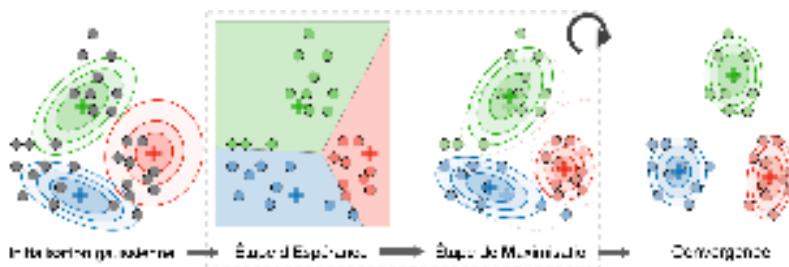
□ Algorithme – L'algorithme d'espérance-maximisation (EM) est une méthode efficace pour estimer le paramètre θ . Elle passe par le maximum de vraisemblance en construisant un borne inférieure sur la vraisemblance (E-step) et optimisant cette borne inférieure (M-step) de manière successive :

- E-step : Évaluer la probabilité postérieure $Q_i(z^{(i)})$ que chaque point $x^{(i)}$ provienne d'une partition particulière $z^{(i)}$ de la manière suivante :

$$Q_i(z^{(i)}) = P(z^{(i)} | x^{(i)}; \theta)$$

- M-step : Utiliser les probabilités postérieures $Q_i(z^{(i)})$ en tant que coefficients propres aux partitions sur les points $x^{(i)}$ pour ré-estimer séparément chaque modèle de partition de la manière suivante :

$$\theta_i = \underset{\theta}{\operatorname{argmax}} \sum_i \int_{z^{(i)}} Q_i(z^{(i)}) \log \left(\frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right) dz^{(i)}$$

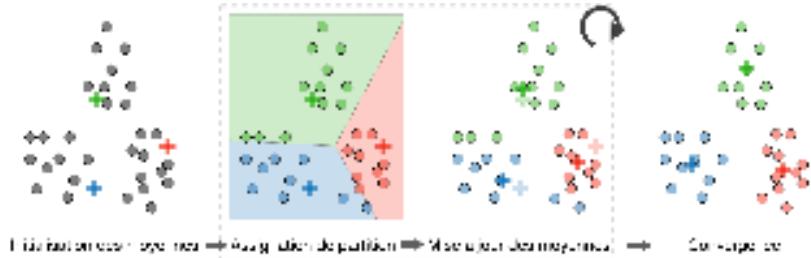


2.2.2 Partitionnement k -means

On note $c^{(i)}$ la partition du point i et μ_j le centre de la partition j .

□ Algorithme – Après avoir aléatoirement initialisé les centroïdes de partitions $\mu_1, \mu_2, \dots, \mu_k \in \mathbb{R}^n$, l'algorithme k -means répète l'étape suivante jusqu'à convergence :

$$c^{(i)} = \arg \min_j \|x^{(i)} - \mu_j\|^2 \quad \text{et} \quad \mu_j = \frac{\sum_{i=1}^m 1_{\{c^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^m 1_{\{c^{(i)}=j\}}}$$



□ Fonction de distortion – Pour voir si l'algorithme converge, on regarde la fonction de distortion définie de la manière suivante :

$$J(c, \mu) = \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

2.2.3 Regroupement hiérarchique

□ Algorithme – C'est un algorithme de partitionnement avec une approche hiérarchique qui construit des partitions intriquées de manière successive.

□ Types – Il y a différents types d'algorithme de regroupement hiérarchique qui ont pour but d'optimiser différents fonctions objectif, récapitulés dans le tableau ci-dessous :

Ward linkage	Average linkage	Complete linkage
Minimize within cluster distance	Minimize average distance between cluster pairs	Minimize maximum distance of between cluster pairs

2.2.4 Indicateurs d'évaluation de clustering

Dans le cadre de l'apprentissage non-supervisé, il est souvent difficile d'évaluer la performance d'un modèle vu que les vrais labels ne sont pas connus (contrairement à l'apprentissage supervisé).

□ Coefficient silhouette – En notant a et b la distance moyenne entre un échantillon et tous les autres points d'une même classe, et entre un échantillon et tous les autres points de la prochaine partition la plus proche, le coefficient silhouette s d'un échantillon donné est défini de la manière suivante :

$$s = \frac{b - a}{\max(a, b)}$$

□ **Index de Calinski-Harabaz** – En notant k le nombre de partitions, B_k et W_k les matrices de dispersion entre-partitions et au sein d'une même partition sont définis respectivement par :

$$B_k = \sum_{j=1}^k n_{c(i)} (\mu_{c(i)} - \mu) (\mu_{c(i)} - \mu)^T, \quad W_k = \sum_{i=1}^m (x^{(i)} - \mu_{c(i)}) (x^{(i)} - \mu_{c(i)})^T$$

l'index de Calinski-Harabaz $s(k)$ renseigne sur la qualité des partitions, de sorte à ce qu'un score plus élevé indique des partitions plus denses et mieux séparées entre elles. Il est défini par :

$$s(k) = \frac{\text{Tr}(B_k)}{\text{Tr}(W_k)} \times \frac{N - k}{k - 1}$$

2.3 Réduction de dimension

2.3.1 Analyse des composantes principales

C'est une technique de réduction de dimension qui trouve les directions maximisant la variance, vers lesquelles les données sont projetées.

□ **Valeur propre, vecteur propre** – Soit une matrice $A \in \mathbb{R}^{n \times n}$, λ est dit être une valeur propre de A s'il existe un vecteur $z \in \mathbb{R}^n \setminus \{0\}$, appelé vecteur propre, tel que l'on a :

$$Az = \lambda z$$

□ **Théorème spectral** – Soit $A \in \mathbb{R}^{n \times n}$. Si A est symétrique, alors A est diagonalisable par une matrice réelle orthogonale $U \in \mathbb{R}^{n \times n}$. En notant $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$, on a :

$$\exists \Lambda \text{ diagonal}, \quad A = U \Lambda U^T$$

Remarque : le vecteur propre associé à la plus grande valeur propre est appelé le vecteur propre principal de la matrice A .

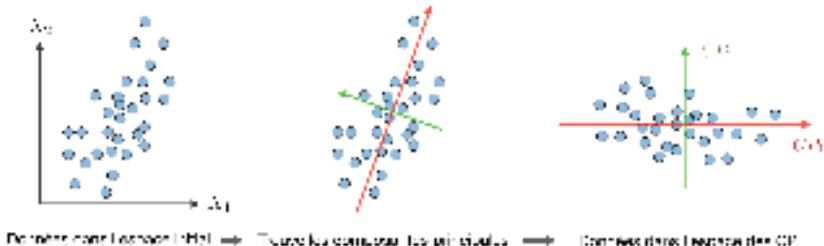
□ **Algorithme** – La procédure d'analyse des composantes principales (en anglais *PCA - Principal Component Analysis*) est une technique de réduction de dimension qui projette les données sur k dimensions en maximisant la variance des données de la manière suivante :

- Étape 1 : Normaliser les données pour avoir une moyenne de 0 et un écart-type de 1.

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{\sigma_j} \quad \text{où} \quad \mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)} \quad \text{et} \quad \sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2$$

- Étape 2 : Calculer $\Sigma = \frac{1}{m} \sum_{i=1}^m x^{(i)} x^{(i)T} \in \mathbb{R}^{n \times n}$, qui est symétrique et aux valeurs propres réelles.
- Étape 3 : Calculer $u_1, \dots, u_k \in \mathbb{R}^n$ les k valeurs propres principales orthogonales de Σ , i.e. les vecteurs propres orthogonaux des k valeurs propres les plus grandes.

- Étape 4 : Projeter les données sur $\text{span}_{\mathbb{R}}(u_1, \dots, u_k)$. Cette procédure maximise la variance sur tous les espaces à k dimensions.



2.3.2 Analyse en composantes indépendantes

C'est une technique qui vise à trouver les sources génératrices sous-jacentes.

- **Hypothèses** – On suppose que nos données x ont été générées par un vecteur source à n dimensions $s = (s_1, \dots, s_n)$, où les s_i sont des variables aléatoires indépendantes, par le biais d'une matrice de mélange et inversible A de la manière suivante :

$$x = As$$

Le but est de trouver la matrice de démêlange $W = A^{-1}$.

- **Algorithme d'ICA de Bell and Sejnowski** – Cet algorithme trouve la matrice de démêlange W en suivant les étapes ci-dessous :

- Écrire la probabilité de $x = As = W^{-1}s$ par :

$$p(x) = \prod_{i=1}^n p_s(w_i^T x) \cdot |W|$$

- Écrire la log vraisemblance de notre jeu de données d'entraînement $\{x^{(i)}, i \in [1, m]\}$ et en notant g la fonction sigmoïde par :

$$l(W) = \sum_{i=1}^m \left(\sum_{j=1}^n \log \left(g'(w_j^T x^{(i)}) \right) + \log |W| \right)$$

Par conséquent, l'algorithme du gradient stochastique est tel que pour chaque exemple du jeu d'entraînement $x^{(i)}$, on met à jour W de la manière suivante :

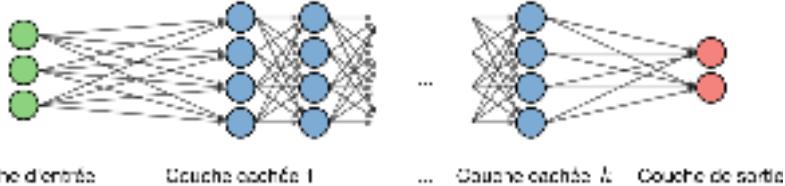
$$W \leftarrow W + \alpha \left(\begin{pmatrix} 1 - 2g(w_1^T x^{(i)}) \\ 1 - 2g(w_2^T x^{(i)}) \\ \vdots \\ 1 - 2g(w_n^T x^{(i)}) \end{pmatrix} x^{(i)T} + (W^T)^{-1} \right)$$

3 Apprentissage profond

3.1 Réseau de neurones

Les réseaux de neurones (en anglais *neural networks*) sont une classe de modèles qui sont construits à l'aide de couches de neurones. Les réseaux de neurones convolutionnels (en anglais *convolutional neural networks*) ainsi que les réseaux de neurones récurrents (en anglais *recurrent neural networks*) font partie des principaux types de réseaux de neurones.

Architecture – Le vocabulaire autour des architectures des réseaux de neurones est décrit dans la figure ci-dessous :

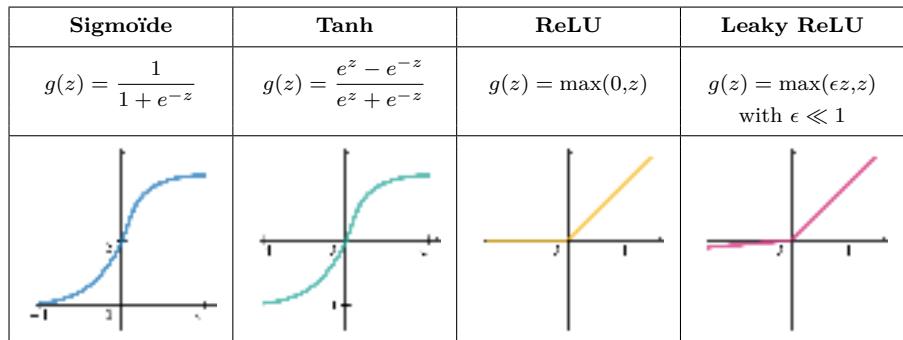


En notant i la $i^{\text{ème}}$ couche du réseau et j la $j^{\text{ème}}$ unité de couche cachée, on a :

$$z_j^{[i]} = w_j^{[i]T} x + b_j^{[i]}$$

où l'on note w , b , z le coefficient, le biais ainsi que la variable sortie respectivement.

Fonction d'activation – Les fonctions d'activation sont utilisées à la fin d'une unité de couche cachée pour introduire des complexités non linéaires au modèle. En voici les plus fréquentes :



Cross-entropy loss – Dans le contexte des réseaux de neurones, la fonction objectif de cross-entropie $L(z,y)$ est communément utilisée et est définie de la manière suivante :

$$L(z,y) = - \left[y \log(z) + (1-y) \log(1-z) \right]$$

Taux d'apprentissage – Le taux d'apprentissage (appelé en anglais *learning rate*), souvent noté α ou parfois η , indique la vitesse à laquelle les coefficients évoluent. Cette quantité peut être fixe ou variable. L'une des méthodes les plus populaires à l'heure actuelle s'appelle Adam, qui a un taux d'apprentissage qui s'adapte au fil du temps.

□ **Rétropropagation du gradient** – La rétropropagation du gradient (en anglais *backpropagation*) est une méthode destinée à mettre à jour les coefficients d'un réseau de neurones en comparant la sortie obtenue et la sortie désirée. La dérivée par rapport au coefficient w est calculée à l'aide du théorème de dérivation des fonctions composées, et s'écrit de la manière suivante :

$$\boxed{\frac{\partial L(z,y)}{\partial w} = \frac{\partial L(z,y)}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w}}$$

Ainsi, le coefficient est actualisé de la manière suivante :

$$\boxed{w \leftarrow w - \eta \frac{\partial L(z,y)}{\partial w}}$$

□ **Actualiser les coefficients** – Dans un réseau de neurones, les coefficients sont actualisés comme suit :

- Étape 1 : Prendre un groupe d'observations appartenant au données du training set.
- Étape 2 : Réaliser la propagation avant pour obtenir le loss correspondant.
- Étape 3 : Effectuer une rétropropagation du loss pour obtenir les gradients.
- Étape 4 : Utiliser les gradients pour actualiser les coefficients du réseau.

□ **Dropout** – Le dropout est une technique qui est destinée à empêcher le sur-ajustement sur les données de training en abandonnant des unités dans un réseau de neurones. En pratique, les neurones sont soit abandonnés avec une probabilité p ou gardés avec une probabilité $1 - p$.

3.2 Réseaux de neurones convolutionnels

□ **Pré-requis de la couche convolutionnelle** – Si l'on note W la taille du volume d'entrée, F la taille de la couche de neurones convolutionnelle, P la quantité de zero padding, alors le nombre de neurones N qui tient dans un volume donné est tel que :

$$\boxed{N = \frac{W - F + 2P}{S} + 1}$$

□ **Normalisation de batch** – C'est une étape possédant les paramètres γ, β qui normalise le batch $\{x_i\}$. En notant μ_B, σ_B^2 la moyenne et la variance de ce que l'on veut corriger au batch, ceci est fait de la manière suivante :

$$\boxed{x_i \leftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta}$$

Cela est normalement effectué après une couche fully-connected/couche convolutionnelle et avant une couche de non-linéarité et a pour but de permettre un taux d'apprentissage plus grand et de réduire une dépendance trop forte à l'initialisation.

3.3 Réseaux de neurones récurrents

□ **Types de porte** – Voici les différents types de porte que l'on rencontre dans un réseau de neurones récurrent typique :

Porte d'entrée	Porte d'oubli	Porte de sortie	Porte
Écrire ?	Supprimer ?	A quel point révéler ?	Combien écrire ?

□ **LSTM** – Un réseau de long court terme (en anglais *long short-term memory*, LSTM) est un type de modèle RNN qui empêche le phénomène de *vanishing gradient* en ajoutant des portes d'oubli.

3.4 Reinforcement Learning

Le but du reinforcement learning est pour un agent d'apprendre comment évoluer dans un environnement.

□ **Processus de décision markovien** – Un processus de décision markovien (MDP) est décrite par 5 quantités $(S, A, \{P_{sa}\}, \gamma, R)$, où :

- S est l'ensemble des états
- \mathcal{A} est l'ensemble des actions
- $\{P_{sa}\}$ sont les probabilités d'états de transition pour $s \in S$ et $a \in \mathcal{A}$
- $\gamma \in [0, 1[$ est le taux d'actualisation (en anglais *discount factor*)
- $R : S \times \mathcal{A} \rightarrow \mathbb{R}$ ou $R : S \rightarrow \mathbb{R}$ est la fonction de récompense que l'algorithme veut maximiser

□ **Politique** – Une politique π est une fonction $\pi : S \rightarrow \mathcal{A}$ qui lie les états aux actions.

Remarque : on dit que l'on effectue une politique donnée π si étant donné un état s , on prend l'action $a = \pi(s)$.

□ **Fonction de valeurs** – Pour une politique donnée π et un état donné s , on définit la fonction de valeurs V^π comme suit :

$$V^\pi(s) = E \left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots | s_0 = s, \pi \right]$$

□ **Équation de Bellman** – Les équations de Bellman optimales caractérisent la fonction de valeurs V^{π^*} de la politique optimale π^* :

$$V^{\pi^*}(s) = R(s) + \max_{a \in \mathcal{A}} \gamma \sum_{s' \in S} P_{sa}(s') V^{\pi^*}(s')$$

Remarque : on note que la politique optimale π^ pour un état donné s est tel que :*

$$\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} \sum_{s' \in S} P_{sa}(s') V^*(s')$$

□ **Algorithme d'itération sur la valeur** – L'algorithme d'itération sur la valeur est faite de deux étapes :

- On initialise la valeur :

$$V_0(s) = 0$$

- On itère la valeur en se basant sur les valeurs précédentes :

$$V_{i+1}(s) = R(s) + \max_{a \in \mathcal{A}} \left[\sum_{s' \in \mathcal{S}} \gamma P_{sa}(s') V_i(s') \right]$$

- **Maximum de vraisemblance** – Les estimations du maximum de vraisemblance pour les transitions de probabilité d'état sont comme suit :

$$P_{sa}(s') = \frac{\#\text{fois où l'action } a \text{ dans l'état } s \text{ est prise pour arriver à l'état } s'}{\#\text{fois où l'action } a \text{ dans l'état } s \text{ est prise}}$$

- **Q-learning** – Le Q-learning est une estimation non-paramétrique de Q, qui est faite de la manière suivante :

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

4 Astuces de Machine Learning

4.1 Indicateurs dans le contexte de la classification

Dans le contexte de la classification binaire, voici les principaux indicateurs à surveiller pour évaluer la performance d'un modèle.

Matrice de confusion – Une matrice de confusion est utilisée pour avoir une image complète de la performance d'un modèle. Elle est définie de la manière suivante :

		Classe prédictive	
		+	-
Classe vraie	+	TP True Positives	FN False Negatives Type II error
	-	FP False Positives Type I error	TN True Negatives

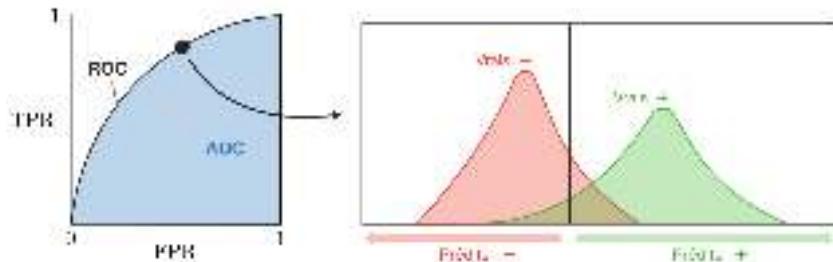
Indicateurs principaux – Les indicateurs suivants sont communément utilisés pour évaluer la performance des modèles de classification :

Indicateur	Formule	Interprétation
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$	Performance globale du modèle
Precision	$\frac{TP}{TP + FP}$	À quel point les prédictions positives sont précises
Recall Sensitivity	$\frac{TP}{TP + FN}$	Couverture des observations vraiment positives
Specificity	$\frac{TN}{TN + FP}$	Couverture des observations vraiment négatives
F1 score	$\frac{2TP}{2TP + FP + FN}$	Indicateur hybride pour les classes non-balancées

Courbe ROC – La fonction d'efficacité du récepteur, plus fréquemment appelée courbe ROC (de l'anglais *Receiver Operating Curve*), est une courbe représentant le taux de *True Positives* en fonction de taux de *False Positives* et obtenue en faisant varier le seuil. Ces indicateurs sont résumés dans le tableau suivant :

Indicateur	Formule	Equivalent
True Positive Rate TPR	$\frac{TP}{TP + FN}$	Recall, sensitivity
False Positive Rate FPR	$\frac{FP}{TN + FP}$	1-specificity

□ **AUC** – L'aire sous la courbe ROC, aussi notée AUC (de l'anglais *Area Under the Curve*) ou AUROC (de l'anglais *Area Under the ROC*), est l'aire sous la courbe ROC comme le montre la figure suivante :



4.2 Indicateurs dans le contexte de la régression

□ **Indicateurs de base** – Étant donné un modèle de régression f , les indicateurs suivants sont communément utilisés pour évaluer la performance d'un modèle :

Somme des carrés totale	Somme des carrés expliquée	Somme des carrés résiduelle
$SS_{\text{tot}} = \sum_{i=1}^m (y_i - \bar{y})^2$	$SS_{\text{reg}} = \sum_{i=1}^m (f(x_i) - \bar{y})^2$	$SS_{\text{res}} = \sum_{i=1}^m (y_i - f(x_i))^2$

□ **Coefficient de détermination** – Le coefficient de détermination, souvent noté R^2 ou r^2 , donne une mesure sur la qualité du modèle et est tel que :

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

□ **Indicateurs principaux** – Les indicateurs suivants sont communément utilisés pour évaluer la performance des modèles de régression, en prenant en compte le nombre de variables n qu'ils prennent en considération :

Cp de Mallow	AIC	BIC	R^2 ajusté
$\frac{SS_{\text{res}} + 2(n + 1)\hat{\sigma}^2}{m}$	$2[(n + 2) - \log(L)]$	$\log(m)(n + 2) - 2\log(L)$	$1 - \frac{(1 - R^2)(m - 1)}{m - n - 1}$

où L est la vraisemblance et $\hat{\sigma}^2$ est une estimation de la variance associée à chaque réponse.

4.3 Sélection de modèle

□ **Vocabulaire** – Lors de la sélection d'un modèle, on divise les données en 3 différentes parties comme suit :

Training set	Validation set	Testing set
<ul style="list-style-type: none"> - Modèle est entraîné - Normalement 80% du dataset 	<ul style="list-style-type: none"> - Modèle est évaluée - Normalement 20% du dataset - Aussi appelé hold-out ou development set 	<ul style="list-style-type: none"> - Modèle donne des prédictions - Données jamais vues

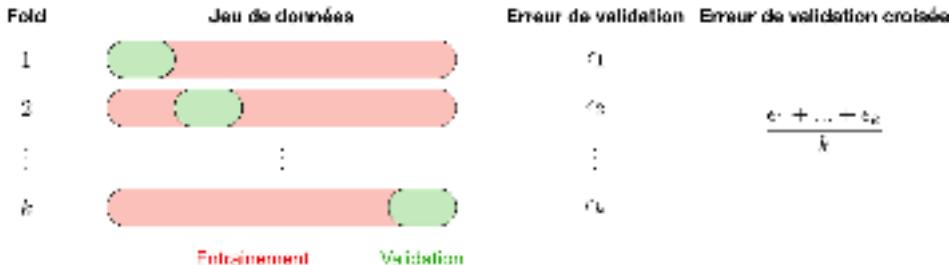
Une fois que le modèle a été choisi, il est entraîné sur le jeu de données entier et testé sur test set (qui n'a jamais été vu). Ces derniers sont représentés dans la figure ci-dessous :



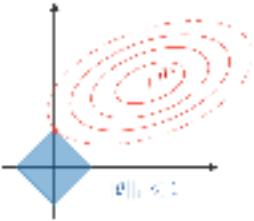
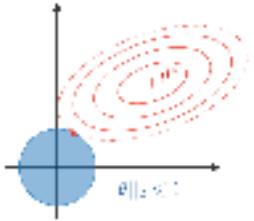
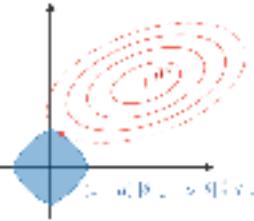
□ **Validation croisée** – La validation croisée, aussi notée CV (de l'anglais *Cross-Validation*), est une méthode qui est utilisée pour sélectionner un modèle qui ne s'appuie pas trop sur le training set de départ. Les différents types de validation croisée rencontrés sont résumés dans le tableau ci-dessous :

<i>k</i> -fold	Leave- <i>p</i> -out
<ul style="list-style-type: none"> - Entraînement sur $k - 1$ folds et évaluation sur le fold restant - Généralement $k = 5$ ou 10 	<ul style="list-style-type: none"> - Entraînement sur $n - p$ observations et évaluation sur les p restantes - Cas $p = 1$ est appelé <i>leave-one-out</i>

La méthode la plus utilisée est appelée validation croisée *k*-fold et partage le jeu de données d'entraînement en *k* folds, de manière à valider le modèle sur un fold tout en entraînant le modèle sur les $k - 1$ autres folds, tout ceci *k* fois. L'erreur est alors moyennée sur *k* folds et est appelée erreur de validation croisée.



□ **Régularisation** – La procédure de régularisation a pour but d'éviter que le modèle ne surapprenne (en anglais *overfit*) les données et ainsi vise à régler les problèmes de grande variance. Le tableau suivant récapitule les différentes techniques de régularisation communément utilisées.

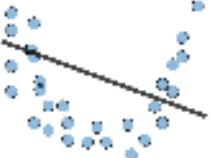
LASSO	Ridge	Elastic Net
- Réduit les coefficients à 0 - Bon pour la sélection de variables	Rend les coefficients plus petits	Compromis entre la sélection de variables et la réduction de coefficients
 $\ \theta\ _1 \leq 1$	 $\ \theta\ _2 \leq 1$	 $\ \theta - \alpha\ _1 + \alpha\ \theta\ _2 \leq C$
$\dots + \lambda \ \theta\ _1$ $\lambda \in \mathbb{R}$	$\dots + \lambda \ \theta\ _2^2$ $\lambda \in \mathbb{R}$	$\dots + \lambda \left[(1 - \alpha) \ \theta\ _1 + \alpha \ \theta\ _2^2 \right]$ $\lambda \in \mathbb{R}, \alpha \in [0,1]$

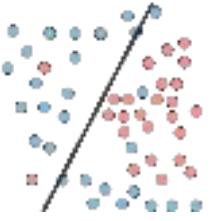
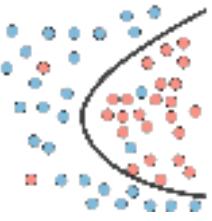
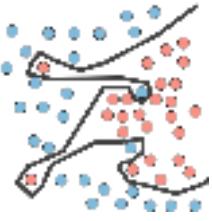
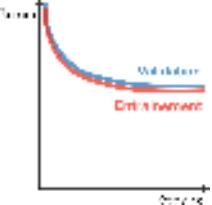
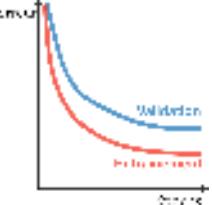
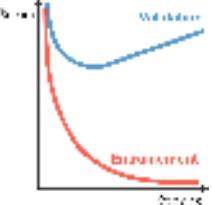
4.4 Diagnostics

Biais – Le biais d'un modèle est la différence entre l'espérance de la prédiction et du modèle correct pour lequel on essaie de prédire pour des observations données.

Variance – La variance d'un modèle est la variabilité des prédictions d'un modèle pour des observations données.

Compromis biais/variance – Plus le modèle est simple, plus le biais est grand et plus le modèle est complexe, plus la variance est grande.

	Underfitting	Just right	Overfitting
Symptômes	<ul style="list-style-type: none"> - Erreur de training élevé - Erreur de training proche de l'erreur de test - Biais élevé 	<ul style="list-style-type: none"> - Erreur de training légèrement inférieure à l'erreur de test 	<ul style="list-style-type: none"> - Erreur de training très faible - Erreur de training beaucoup plus faible que l'erreur de test - Variance élevée
Régression			

Classification			
Deep Learning			
Remèdes	<ul style="list-style-type: none"> - Complexifier le modèle - Ajouter plus de variables - Laisser le training pendant plus de temps 		<ul style="list-style-type: none"> - Effectuer une regularisation - Avoir plus de données

□ **Analyse de l'erreur** – L'analyse de l'erreur consiste à analyser la cause première de la différence en performance entre le modèle actuel et le modèle parfait.

□ **Analyse ablative** – L'analyse ablative consiste à analyser la cause première de la différence en performance entre le modèle actuel et le modèle de base.

5 Rappels

5.1 Probabilités et Statistiques

5.1.1 Introduction aux probabilités à l'analyse combinatoire

□ Univers de probabilités – L'ensemble de toutes les issues possibles d'une expérience aléatoire est appelé l'univers de probabilités d'une expérience aléatoire et est noté S .

□ Évènement – Toute partie E d'un univers est appelé un évènement. Ainsi, un évènement est un ensemble d'issues possibles d'une expérience aléatoire. Si l'issue de l'expérience aléatoire est contenue dans E , alors on dit que E s'est produit.

□ Axiomes de probabilités – Pour chaque évènement E , on note $P(E)$ la probabilité que l'évènement E se produise.

$$(1) \quad 0 \leq P(E) \leq 1 \quad (2) \quad P(S) = 1 \quad (3) \quad P\left(\bigcup_{i=1}^n E_i\right) = \sum_{i=1}^n P(E_i)$$

□ Permutation – Une permutation est un arrangement de r objets parmi n objets, dans un ordre donné. Le nombre de tels arrangements est donné par $P(n,r)$, défini par :

$$P(n,r) = \frac{n!}{(n-r)!}$$

□ Combinaison – Une combinaison est un arrangement de r objets parmi n objets, où l'ordre ne compte pas. Le nombre de tels arrangements est donné par $C(n,r)$, défini par :

$$C(n,r) = \frac{P(n,r)}{r!} = \frac{n!}{r!(n-r)!}$$

Remarque : on note que pour $0 \leq r \leq n$, on a $P(n,r) \geq C(n,r)$.

5.1.2 Probabilité conditionnelle

□ Théorème de Bayes – Pour des évènements A et B tels que $P(B) > 0$, on a :

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Remarque : on a $P(A \cap B) = P(A)P(B|A) = P(A|B)P(B)$.

□ Partition – Soit $\{A_i, i \in [1,n]\}$ tel que pour tout i , $A_i \neq \emptyset$. On dit que $\{A_i\}$ est une partition si l'on a :

$$\forall i \neq j, A_i \cap A_j = \emptyset \quad \text{et} \quad \bigcup_{i=1}^n A_i = S$$

Remarque : pour tout évènement B dans l'univers de probabilités, on a $P(B) = \sum_{i=1}^n P(B|A_i)P(A_i)$.

□ **Formule étendue du théorème de Bayes** – Soit $\{A_i, i \in \llbracket 1, n \rrbracket\}$ une partition de l'univers de probabilités. On a :

$$P(A_k|B) = \frac{P(B|A_k)P(A_k)}{\sum_{i=1}^n P(B|A_i)P(A_i)}$$

□ **Indépendance** – Deux événements A et B sont dits indépendants si et seulement si on a :

$$P(A \cap B) = P(A)P(B)$$

5.1.3 Variable aléatoires

□ **Variable aléatoire** – Une variable aléatoire, souvent notée X , est une fonction qui associe chaque élément de l'univers de probabilité à la droite des réels.

□ **Fonction de répartition** – La fonction de répartition F (en anglais *CDF - Cumulative distribution function*), qui est croissante monotone et telle que

$$\lim_{x \rightarrow -\infty} F(x) = 0 \quad \text{et} \quad \lim_{x \rightarrow +\infty} F(x) = 1$$

est définie de la manière suivante :

$$F(x) = P(X \leq x)$$

Remarque : on a $P(a < X \leq b) = F(b) - F(a)$.

□ **Densité de probabilité** – La densité de probabilité f (en anglais *PDF - Probability density function*) est la probabilité que X prenne des valeurs entre deux réalisations adjacentes d'une variable aléatoire.

□ **Relations vérifiées par les PDF et CDF** – Voici les propriétés importantes à savoir dans les cas discret (D) et continu (C).

Case	CDF F	PDF f	Propriétés du PDF
(D)	$F(x) = \sum_{x_i \leq x} P(X = x_i)$	$f(x_j) = P(X = x_j)$	$0 \leq f(x_j) \leq 1$ and $\sum_j f(x_j) = 1$
(C)	$F(x) = \int_{-\infty}^x f(y)dy$	$f(x) = \frac{dF}{dx}$	$f(x) \geq 0$ and $\int_{-\infty}^{+\infty} f(x)dx = 1$

□ **Variance** – La variance d'une variable aléatoire, souvent notée $\text{Var}(X)$ ou σ^2 , est une mesure de la dispersion de ses fonctions de distribution. Elle est déterminée de la manière suivante :

$$\text{Var}(X) = E[(X - E[X])^2] = E[X^2] - E[X]^2$$

□ **Écart-type** – L'écart-type d'une variable aléatoire, souvent notée σ , est une mesure de la dispersion de sa fonction de distribution, exprimée avec les mêmes unités que la variable aléatoire. Il est déterminé de la manière suivante :

$$\sigma = \sqrt{\text{Var}(X)}$$

□ **Espérance et moments de la distribution** – Voici les expressions de l'espérance $E[X]$, l'espérance généralisée $E[g(X)]$, $k^{\text{ème}}$ moment $E[X^k]$ et fonction caractéristique $\psi(\omega)$ dans les cas discret et continu.

Case	$E[X]$	$E[g(X)]$	$E[X^k]$	$\psi(\omega)$
(D)	$\sum_{i=1}^n x_i f(x_i)$	$\sum_{i=1}^n g(x_i) f(x_i)$	$\sum_{i=1}^n x_i^k f(x_i)$	$\sum_{i=1}^n f(x_i) e^{i\omega x_i}$
(C)	$\int_{-\infty}^{+\infty} x f(x) dx$	$\int_{-\infty}^{+\infty} g(x) f(x) dx$	$\int_{-\infty}^{+\infty} x^k f(x) dx$	$\int_{-\infty}^{+\infty} f(x) e^{i\omega x} dx$

Remarque : on a $e^{i\omega x} = \cos(\omega x) + i \sin(\omega x)$.

□ **Transformation de variables aléatoires** – Soit X, Y des variables liées par une certaine fonction. En notant f_X et f_Y les fonctions de distribution de X et Y respectivement, on a :

$$f_Y(y) = f_X(x) \left| \frac{dx}{dy} \right|$$

□ **Loi d'intégration de Leibniz** – Soit g une fonction de x et potentiellement c , et a, b , les limites de l'intervalle qui peuvent dépendre de c . On a :

$$\frac{\partial}{\partial c} \left(\int_a^b g(x) dx \right) = \frac{\partial b}{\partial c} \cdot g(b) - \frac{\partial a}{\partial c} \cdot g(a) + \int_a^b \frac{\partial g}{\partial c}(x) dx$$

□ **Inégalité de Tchebychev** – Soit X une variable aléatoire de moyenne μ . Pour $k, \sigma > 0$, on a l'inégalité suivante :

$$P(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2}$$

5.1.4 Variables aléatoires conjointement distribuées

□ **Densité conditionnelle** – La densité conditionnelle de X par rapport à Y , souvent notée $f_{X|Y}$, est définie de la manière suivante :

$$f_{X|Y}(x) = \frac{f_{XY}(x,y)}{f_Y(y)}$$

□ **Indépendance** – Deux variables aléatoires X et Y sont dits indépendantes si l'on a :

$$f_{XY}(x,y) = f_X(x)f_Y(y)$$

□ **Densité marginale et fonction de répartition** – À partir de la densité de probabilité f_{XY} , on a :

Cas	Densité marginale	Fonction de répartition
(D)	$f_X(x_i) = \sum_j f_{XY}(x_i, y_j)$	$F_{XY}(x, y) = \sum_{x_i \leq x} \sum_{y_j \leq y} f_{XY}(x_i, y_j)$
(C)	$f_X(x) = \int_{-\infty}^{+\infty} f_{XY}(x, y) dy$	$F_{XY}(x, y) = \int_{-\infty}^x \int_{-\infty}^y f_{XY}(x', y') dx' dy'$

□ **Covariance** – On définit la covariance de deux variables aléatoires X et Y , que l'on note σ_{XY}^2 ou plus souvent $\text{Cov}(X, Y)$, de la manière suivante :

$$\boxed{\text{Cov}(X, Y) \triangleq \sigma_{XY}^2 = E[(X - \mu_X)(Y - \mu_Y)] = E[XY] - \mu_X \mu_Y}$$

□ **Corrélation** – En notant σ_X, σ_Y les écart-types de X et Y , on définit la corrélation entre les variables aléatoires X et Y , que l'on note ρ_{XY} , de la manière suivante :

$$\boxed{\rho_{XY} = \frac{\sigma_{XY}^2}{\sigma_X \sigma_Y}}$$

Remarques : on note que pour toute variable aléatoire X, Y , on a $\rho_{XY} \in [-1, 1]$. Si X et Y sont indépendants, alors $\rho_{XY} = 0$.

□ **Distributions importantes** – Voici les distributions importantes à savoir :

Type	Distribution	PDF	$\psi(\omega)$	$E[X]$	$\text{Var}(X)$
(D)	$X \sim \mathcal{B}(n, p)$ Binomial	$P(X = x) = \binom{n}{x} p^x q^{n-x}$ $x \in \llbracket 0, n \rrbracket$	$(pe^{i\omega} + q)^n$	np	npq
	$X \sim \text{Po}(\mu)$ Poisson	$P(X = x) = \frac{\mu^x}{x!} e^{-\mu}$ $x \in \mathbb{N}$	$e^{\mu(e^{i\omega}-1)}$	μ	μ
(C)	$X \sim \mathcal{U}(a, b)$ Uniforme	$f(x) = \frac{1}{b-a}$ $x \in [a, b]$	$\frac{e^{i\omega b} - e^{i\omega a}}{(b-a)i\omega}$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
	$X \sim \mathcal{N}(\mu, \sigma)$ Gaussien	$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2} \left(\frac{x-\mu}{\sigma} \right)^2}$ $x \in \mathbb{R}$	$e^{i\omega\mu - \frac{1}{2}\omega^2\sigma^2}$	μ	σ^2
	$X \sim \text{Exp}(\lambda)$ Exponentiel	$f(x) = \lambda e^{-\lambda x}$ $x \in \mathbb{R}_+$	$\frac{1}{1 - \frac{i\omega}{\lambda}}$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$

5.1.5 Estimation des paramètres

□ **Échantillon aléatoire** – Un échantillon aléatoire est une collection de n variables aléatoires X_1, \dots, X_n qui sont indépendantes et identiquement distribuées avec X .

□ **Estimateur** – Un estimateur est une fonction des données qui est utilisée pour trouver la valeur d'un paramètre inconnu dans un modèle statistique.

□ **Biais** – Le biais d'un estimateur $\hat{\theta}$ est défini comme étant la différence entre l'espérance de la distribution de $\hat{\theta}$ et de la valeur vraie, i.e. :

$$\text{Bias}(\hat{\theta}) = E[\hat{\theta}] - \theta$$

Remarque : un estimateur est dit non biaisé lorsque l'on a $E[\hat{\theta}] = \theta$.

□ **Moyenne empirique et variance empirique** – La moyenne empirique et la variance empirique d'un échantillon aléatoire sont utilisées pour estimer la valeur vraie μ et la variance vraie σ^2 d'une distribution, notés \bar{X} et s^2 et sont définies de la manière suivante :

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i \quad \text{and} \quad s^2 = \hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2$$

□ **Théorème de la limite centrale** – Soit un échantillon aléatoire X_1, \dots, X_n suivant une distribution donnée de moyenne μ et de variance σ^2 , alors on a :

$$\bar{X} \underset{n \rightarrow +\infty}{\sim} \mathcal{N}\left(\mu, \frac{\sigma}{\sqrt{n}}\right)$$

5.2 Algèbre linéaire et Analyse

5.2.1 Notations générales

□ **Vecteur** – On note $x \in \mathbb{R}^n$ un vecteur à n entrées, où $x_i \in \mathbb{R}$ est la $i^{\text{ème}}$ entrée :

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n$$

□ **Matrice** – On note $A \in \mathbb{R}^{m \times n}$ une matrice à m lignes et n colonnes, où $A_{i,j} \in \mathbb{R}$ est l'entrée située à la $i^{\text{ème}}$ ligne et $j^{\text{ème}}$ colonne :

$$A = \begin{pmatrix} A_{1,1} & \cdots & A_{1,n} \\ \vdots & & \vdots \\ A_{m,1} & \cdots & A_{m,n} \end{pmatrix} \in \mathbb{R}^{m \times n}$$

Remarque : le vecteur x défini ci-dessus peut être vu comme une matrice $n \times 1$ et est aussi appelé vecteur colonne.

□ **Matrice identité** – La matrice identité $I \in \mathbb{R}^{n \times n}$ est une matrice carrée avec des 1 sur sa diagonale et des 0 partout ailleurs :

$$I = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix}$$

Remarque : pour toute matrice $A \in \mathbb{R}^{n \times n}$, on a $A \times I = I \times A = A$.

□ **Matrice diagonale** – Une matrice diagonale $D \in \mathbb{R}^{n \times n}$ est une matrice carrée avec des valeurs non nulles sur sa diagonale et des zéros partout ailleurs.

$$D = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & d_n \end{pmatrix}$$

Remarque : on note aussi $D = \text{diag}(d_1, \dots, d_n)$.

5.2.2 Opérations matricielles

□ **Vecteur-vecteur** – Il y a deux types de multiplication vecteur-vecteur :

- Produit scalaire : pour $x, y \in \mathbb{R}^n$, on a :

$$x^T y = \sum_{i=1}^n x_i y_i \in \mathbb{R}$$

- Produit dyadique : pour $x \in \mathbb{R}^m, y \in \mathbb{R}^n$, on a :

$$xy^T = \begin{pmatrix} x_1 y_1 & \cdots & x_1 y_n \\ \vdots & & \vdots \\ x_m y_1 & \cdots & x_m y_n \end{pmatrix} \in \mathbb{R}^{m \times n}$$

□ **Matrice-vecteur** – Le produit de la matrice $A \in \mathbb{R}^{m \times n}$ et du vecteur $x \in \mathbb{R}^n$ est un vecteur de taille \mathbb{R}^m , tel que :

$$Ax = \begin{pmatrix} a_{r,1}^T x \\ \vdots \\ a_{r,m}^T x \end{pmatrix} = \sum_{i=1}^n a_{c,i} x_i \in \mathbb{R}^m$$

où $a_{r,i}^T$ sont les vecteurs-ligne et $a_{c,j}$ sont les vecteurs-colonne de A et x_i sont les entrées de x .

□ **Matrice-matrice** – Le produit des matrices $A \in \mathbb{R}^{m \times n}$ et $B \in \mathbb{R}^{n \times p}$ est une matrice de taille $\mathbb{R}^{n \times p}$, tel que :

$$AB = \begin{pmatrix} a_{r,1}^T b_{c,1} & \cdots & a_{r,1}^T b_{c,p} \\ \vdots & & \vdots \\ a_{r,m}^T b_{c,1} & \cdots & a_{r,m}^T b_{c,p} \end{pmatrix} = \sum_{i=1}^n a_{c,i} b_{r,i}^T \in \mathbb{R}^{n \times p}$$

où $a_{r,i}^T, b_{r,i}^T$ sont des vecteurs-ligne et $a_{c,j}, b_{c,j}$ sont des vecteurs-colonne de A et B respectivement.

□ **Transposée** – La transposée est une matrice $A \in \mathbb{R}^{m \times n}$, notée A^T , qui est telle que ses entrées sont renversées.

$$\forall i, j, \quad A_{i,j}^T = A_{j,i}$$

Remarque : pour des matrices A, B , on a $(AB)^T = B^T A^T$.

□ Inverse – L'inverse d'une matrice carrée inversible A est notée A^{-1} et est l'unique matrice telle que :

$$\boxed{AA^{-1} = A^{-1}A = I}$$

Remarque : toutes les matricées carrés ne sont pas inversibles. Aussi, pour des matrices A, B , on a $(AB)^{-1} = B^{-1}A^{-1}$.

□ Trace – La trace d'une matrice carrée A , notée $\text{tr}(A)$, est la somme de ses entrées diagonales :

$$\boxed{\text{tr}(A) = \sum_{i=1}^n A_{i,i}}$$

Remarque : pour toutes matrices A, B , on a $\text{tr}(A^T) = \text{tr}(A)$ et $\text{tr}(AB) = \text{tr}(BA)$.

□ Déterminant – Le déterminant d'une matrice carrée $A \in \mathbb{R}^{n \times n}$ notée $|A|$ ou $\det(A)$ est exprimée récursivement en termes de $A_{\setminus i, \setminus j}$, qui est la matrice A sans sa $i^{\text{ème}}$ ligne et $j^{\text{ème}}$ colonne, de la manière suivante :

$$\boxed{\det(A) = |A| = \sum_{j=1}^n (-1)^{i+j} A_{i,j} |A_{\setminus i, \setminus j}|}$$

Remarque : A est inversible si et seulement si $|A| \neq 0$. Aussi, $|AB| = |A||B|$ et $|A^T| = |A|$.

5.2.3 Propriétés matricielles

□ Décomposition symétrique – Une matrice donnée A peut être exprimée en termes de ses parties symétrique et antisymétrique de la manière suivante :

$$\boxed{A = \underbrace{\frac{A + A^T}{2}}_{\text{Symétrique}} + \underbrace{\frac{A - A^T}{2}}_{\text{Antisymétrique}}}$$

□ Norme – Une norme est une fonction $N : V \rightarrow [0, +\infty[$ où V est un espace vectoriel, et tel que pour tous $x, y \in V$, on a :

- $N(x + y) \leq N(x) + N(y)$
- $N(ax) = |a|N(x)$ pour a scalaire
- si $N(x) = 0$, alors $x = 0$

Pour $x \in V$, les normes les plus utilisées sont récapitulées dans le tableau ci-dessous :

Norme	Notation	Définition	Cas
Manhattan, L^1	$\ x\ _1$	$\sum_{i=1}^n x_i $	LASSO
Euclidien, L^2	$\ x\ _2$	$\sqrt{\sum_{i=1}^n x_i^2}$	Ridge
p -norme, L^p	$\ x\ _p$	$\left(\sum_{i=1}^n x_i^p\right)^{\frac{1}{p}}$	Inégalité de Hölder
Infini, L^∞	$\ x\ _\infty$	$\max_i x_i $	Convergence uniforme

□ **Dépendance linéaire** – Un ensemble de vecteurs est considéré comme étant linéairement dépendant si un des vecteurs de cet ensemble peut être défini comme une combinaison des autres.

Remarque : si aucun vecteur ne peut être noté de cette manière, alors les vecteurs sont dits linéairement indépendants.

□ **Rang d'une matrice** – Le rang d'une matrice donnée A est notée $\text{rang}(A)$ et est la dimension de l'espace vectoriel généré par ses colonnes. Ceci est équivalent au nombre maximum de colonnes indépendantes de A .

□ **Matrice semi-définie positive** – Une matrice $A \in \mathbb{R}^{n \times n}$ est semi-définie positive et est notée $A \succeq 0$ si l'on a :

$$A = A^T \quad \text{et} \quad \forall x \in \mathbb{R}^n, \quad x^T A x \geq 0$$

Remarque : de manière similaire, une matrice A est dite définie positive et est notée $A > 0$ si elle est semi-définie positive et que pour tout vecteur x non-nul, on a $x^T A x > 0$.

□ **Valeur propre, vecteur propre** – Étant donné une matrice $A \in \mathbb{R}^{n \times n}$, λ est une valeur propre de A s'il existe un vecteur $z \in \mathbb{R}^n \setminus \{0\}$, appelé vecteur propre, tel que :

$$Az = \lambda z$$

□ **Théorème spectral** – Soit $A \in \mathbb{R}^{n \times n}$. Si A est symétrique, alors A est diagonalisable par une matrice orthogonale réelle $U \in \mathbb{R}^{n \times n}$. En notant $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$, on a :

$$\exists \Lambda \text{ diagonal}, \quad A = U \Lambda U^T$$

□ **Décomposition en valeurs singulières** – Pour une matrice A de dimensions $m \times n$, la décomposition en valeurs singulières est une technique de factorisation qui garantit l'existence d'une matrice unitaire U $m \times m$, d'une matrice diagonale Σ $m \times n$ et d'une matrice unitaire V $n \times n$, tel que :

$$A = U \Sigma V^T$$

5.2.4 Analyse matricielle

□ Gradient – Soit $f : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}$ une fonction et $A \in \mathbb{R}^{m \times n}$ une matrice. Le gradient de f par rapport à A est une matrice de taille $m \times n$, notée $\nabla_A f(A)$, telle que :

$$\boxed{\left(\nabla_A f(A) \right)_{i,j} = \frac{\partial f(A)}{\partial A_{i,j}}}$$

Remarque : le gradient de f est seulement défini lorsque f est une fonction donnant un scalaire.

□ Hessienne – Soit $f : \mathbb{R}^n \rightarrow \mathbb{R}$ une fonction et $x \in \mathbb{R}^n$ un vecteur. La hessienne de f par rapport à x est une matrice symétrique $n \times n$, notée $\nabla_x^2 f(x)$, telle que :

$$\boxed{\left(\nabla_x^2 f(x) \right)_{i,j} = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}}$$

Remarque : la hessienne de f est seulement définie lorsque f est une fonction qui donne un scalaire.

□ Opérations de gradient – Pour des matrices A, B, C , les propriétés de gradient suivants sont bons à savoir :

$$\boxed{\nabla_A \text{tr}(AB) = B^T}$$

$$\boxed{\nabla_A^T f(A) = (\nabla_A f(A))^T}$$

$$\boxed{\nabla_A \text{tr}(ABA^T C) = CAB + C^T AB^T}$$

$$\boxed{\nabla_A |A| = |A|(A^{-1})^T}$$

Super Pense-bête VIP : Apprentissage profond

Afshine AMIDI et Shervine AMIDI

6 janvier 2019

Table des matières

1 Réseaux de neurones convolutionnels	1
1.1 Vue d'ensemble	1
1.2 Types de couche	1
1.3 Paramètres du filtre	2
1.4 Réglage des paramètres	2
1.5 Fonctions d'activation communément utilisées	3
1.6 Détection d'objet	3
1.6.1 Vérification et reconnaissance de visage	5
1.6.2 Transfert de style neuronal	5
1.6.3 Architectures utilisant des astuces de calcul	6
2 Réseaux de neurones récurrents	7
2.1 Vue d'ensemble	7
2.2 Dépendances à long terme	8
2.3 Apprentissage de la représentation de mots	9
2.3.1 Motivation et notations	9
2.3.2 Représentation de mots	9
2.4 Comparaison de mots	10
2.5 Modèle de langage	10
2.6 Traduction machine	10
2.7 Attention	11
3 Petites astuces	12
3.1 Traitement des données	12
3.2 Entrainer un réseau de neurones	12
3.2.1 Définitions	12
3.2.2 Recherche de coefficients optimaux	12
3.3 Réglage des paramètres	13
3.3.1 Initialisation des coefficients	13
3.3.2 Optimisation de la convergence	13
3.4 Régularisation	13
3.5 Bonnes pratiques	14

1 Réseaux de neurones convolutionnels

1.1 Vue d'ensemble

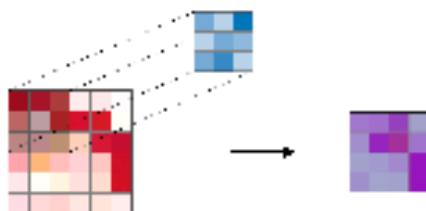
□ **Architecture d'un CNN traditionnel** – Les réseaux de neurones convolutionnels (en anglais *Convolutional neural networks*), aussi connus sous le nom de CNNs, sont un type spécifique de réseaux de neurones qui sont généralement composés des couches suivantes :



La couche convolutionnelle et la couche de pooling peuvent être ajustées en utilisant des paramètres qui sont décrites dans les sections suivantes.

1.2 Types de couche

□ **Couche convolutionnelle (CONV)** – La couche convolutionnelle (en anglais *convolution layer*) (CONV) utilise des filtres qui scannent l'entrée I suivant ses dimensions en effectuant des opérations de convolution. Elle peut être réglée en ajustant la taille du filtre F et le stride S . La sortie O de cette opération est appelée *feature map* ou aussi *activation map*.

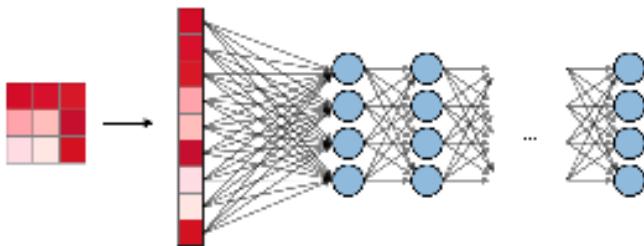


Remarque : l'étape de convolution peut aussi être généralisée dans les cas 1D et 3D.

□ **Pooling (POOL)** – La couche de pooling (en anglais *pooling layer*) (POOL) est une opération de sous-échantillonnage typiquement appliquée après une couche convolutionnelle. En particulier, les types de pooling les plus populaires sont le max et l'average pooling, où les valeurs maximales et moyennes sont prises, respectivement.

	Max pooling	Average pooling
But	Chaque opération de pooling sélectionne la valeur maximale de la surface	Chaque opération de pooling sélectionne la valeur moyenne de la surface
Illustration		
Commentaires	<ul style="list-style-type: none"> - Garde les caractéristiques détectées - Plus communément utilisé 	<ul style="list-style-type: none"> - Sous-échantillonne la <i>feature map</i> - Utilisé dans LeNet

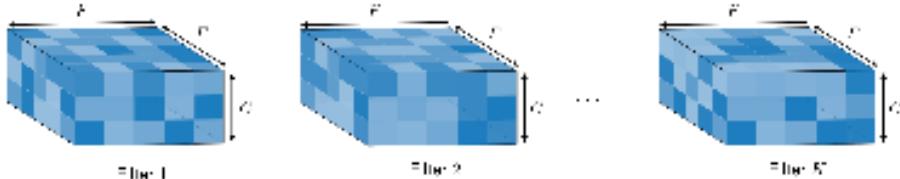
□ **Fully Connected (FC)** – La couche de fully connected (en anglais *fully connected layer*) (FC) s'applique sur une entrée préalablement aplatie où chaque entrée est connectée à tous les neurones. Les couches de fully connected sont typiquement présentes à la fin des architectures de CNN et peuvent être utilisées pour optimiser des objectifs tels que les scores de classe.



1.3 Paramètres du filtre

La couche convolutionnelle contient des filtres pour lesquels il est important de savoir comment ajuster ses paramètres.

□ **Dimensions d'un filtre** – Un filtre de taille $F \times F$ appliqué à une entrée contenant C canaux est un volume de taille $F \times F \times C$ qui effectue des convolutions sur une entrée de taille $I \times I \times C$ et qui produit un *feature map* de sortie (aussi appelé *activation map*) de taille $O \times O \times 1$.



Remarque : appliquer K filtres de taille $F \times F$ engendre un *feature map* de sortie de taille $O \times O \times K$.

□ **Stride** – Dans le contexte d'une opération de convolution ou de pooling, la stride S est un paramètre qui dénote le nombre de pixels par lesquels la fenêtre se déplace après chaque opération.



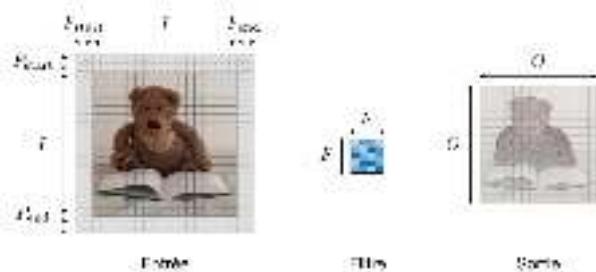
□ **Zero-padding** – Le zero-padding est une technique consistant à ajouter P zeros à chaque côté des frontières de l'entrée. Cette valeur peut être spécifiée soit manuellement, soit automatiquement par le bias d'une des configurations détaillées ci-dessous :

	Valide	Pareil	Total
Valeur	$P = 0$	$P_{\text{start}} = \left\lceil \frac{S \lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rceil$ $P_{\text{end}} = \left\lceil \frac{S \lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rceil$	$P_{\text{start}} \in [0, F - 1]$ $P_{\text{end}} = F - 1$
Illustration			
But	<ul style="list-style-type: none"> - Pas de padding - Enlève la dernière opération de convolution si les dimensions ne collent pas 	<ul style="list-style-type: none"> - Le padding tel que la feature map est de taille $\left\lceil \frac{I}{S} \right\rceil$ - La taille de sortie est mathématiquement satisfaisante - Aussi appelé 'demi' padding 	<ul style="list-style-type: none"> - Padding maximum tel que les dernières convolutions sont appliquées sur les bords de l'entrée - Le filtre 'voit' l'entrée du début à la fin

1.4 Réglage des paramètres

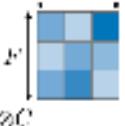
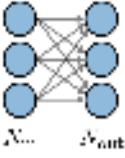
□ **Compatibilité des paramètres dans la couche convolutionnelle** – En notant I le côté du volume d'entrée, F la taille du filtre, P la quantité de zero-padding, S la stride, la taille O de la feature map de sortie suivant cette dimension est telle que :

$$O = \frac{I - F + P_{\text{start}} + P_{\text{end}}}{S} + 1$$



Remarque : on a souvent $P_{start} = P_{end} \triangleq P$, auquel cas on remplace $P_{start} + P_{end}$ par $2P$ dans la formule au-dessus.

□ **Comprendre la complexité du modèle** – Pour évaluer la complexité d'un modèle, il est souvent utile de déterminer le nombre de paramètres que l'architecture va avoir. Dans une couche donnée d'un réseau de neurones convolutionnels, on a :

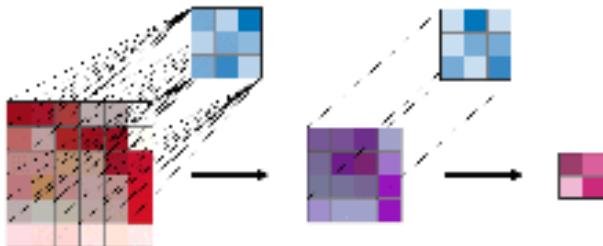
	CONV	POOL	FC
Illustration	 $F \times K$		 $N_{in} \times N_{out}$
Taille d'entrée	$I \times I \times C$	$I \times I \times C$	N_{in}
Taille de sortie	$O \times O \times K$	$O \times O \times C$	N_{out}
Nombre de paramètres	$(F \times F \times C + 1) \cdot K$	0	$(N_{in} + 1) \times N_{out}$
Remarques	<ul style="list-style-type: none"> - Un paramètre de biais par filtre - Dans la plupart des cas, $S < F$ - $2C$ est un choix commun pour K 	<ul style="list-style-type: none"> - L'opération de pooling est effectuée pour chaque canal - Dans la plupart des cas, $S = F$ 	<ul style="list-style-type: none"> - L'entrée est aplatie - Un paramètre de biais par neurone - Le choix du nombre de neurones de FC est libre

□ **Champ récepteur** – Le champ récepteur à la couche k est la surface notée $R_k \times R_k$ de l'entrée que chaque pixel de la k -ième *activation map* peut 'voir'.

En notant F_j la taille du filtre de la couche j et S_i la valeur de stride de la couche i et avec la convention $S_0 = 1$, le champ récepteur à la couche k peut être calculé de la manière suivante :

$$R_k = 1 + \sum_{j=1}^k (F_j - 1) \prod_{i=0}^{j-1} S_i$$

Dans l'exemple ci-dessous, on a $F_1 = F_2 = 3$ et $S_1 = S_2 = 1$, ce qui donne $R_2 = 1+2 \cdot 1+2 \cdot 1 = 5$.



1.5 Fonctions d'activation communément utilisées

□ **Unité linéaire rectifiée** – La couche d'unité linéaire rectifiée (en anglais *rectified linear unit layer*) (ReLU) est une fonction d'activation g qui est utilisée sur tous les éléments du volume. Elle a pour but d'introduire des complexités non-linéaires au réseau. Ses variantes sont récapitulées dans le tableau suivant :

ReLU	Leaky ReLU	ELU
$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$	$g(z) = \max(\alpha(e^z - 1), z)$ with $\alpha \ll 1$
Complexités non-linéaires interprétables d'un point de vue biologique	Répond au problème de <i>dying ReLU</i>	Dérivable partout

□ **Softmax** – L'étape softmax peut être vue comme une généralisation de la fonction logistique qui prend comme argument un vecteur de scores $x \in \mathbb{R}^n$ et qui renvoie un vecteur de probabilités $p \in \mathbb{R}^n$ à travers une fonction softmax à la fin de l'architecture. Elle est définie de la manière suivante :

$$p = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \quad \text{where} \quad p_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

1.6 Détection d'objet

□ **Types de modèles** – Il y a 3 principaux types d'algorithme de reconnaissance d'objet, pour lesquels la nature de ce qui est prédit est différent. Ils sont décrits dans la table ci-dessous :

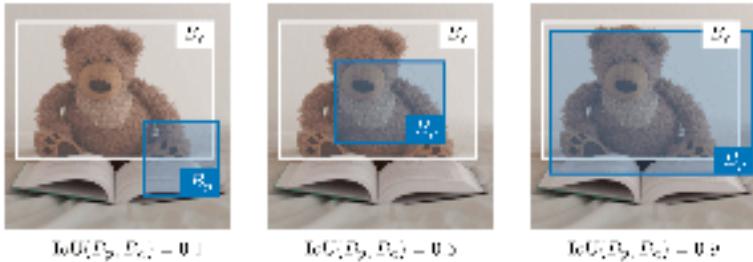
Classification d'image	Classification avec localisation	Détection
		
<ul style="list-style-type: none"> - Classifie une image - Prédit la probabilité d'un objet 	<ul style="list-style-type: none"> - Détecte un objet dans une image - Prédit la probabilité de présence d'un objet et où il est situé 	<ul style="list-style-type: none"> - Peut détecter plusieurs objets dans une image - Prédit les probabilités de présence des objets et où ils sont situés
CNN traditionnel	YOLO simplifié, R-CNN	YOLO, R-CNN

□ **Détection** – Dans le contexte de la détection d'objet, des méthodes différentes sont utilisées selon si l'on veut juste localiser l'objet ou alors détecter une forme plus complexe dans l'image. Les deux méthodes principales sont résumées dans le tableau ci-dessous :

Détection de zone délimitante	Détection de forme complexe
Détecte la partie de l'image où l'objet est situé	<ul style="list-style-type: none"> - Détecte la forme ou les caractéristiques d'un objet (e.g. yeux) - Plus granulaire
	
Zone de centre (b_x, b_y) , hauteur b_h et largeur b_w	Points de référence $(l_{1x}, l_{1y}), \dots, (l_{nx}, l_{ny})$

□ **Intersection sur Union** – Intersection sur Union (en anglais *Intersection over Union*), aussi appelé IoU, est une fonction qui quantifie à quel point la zone délimitante prédite B_p est correctement positionnée par rapport à la zone délimitante vraie B_a . Elle est définie de la manière suivante :

$$\text{IoU}(B_p, B_a) = \frac{B_p \cap B_a}{B_p \cup B_a}$$



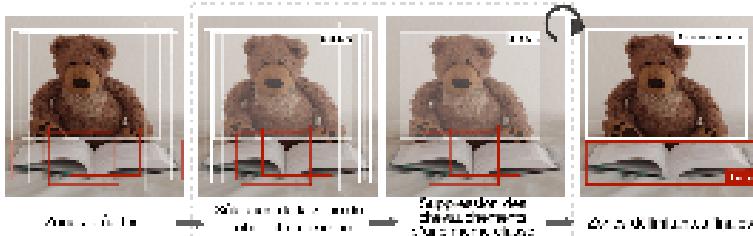
Remarque : on a toujours $\text{IoU} \in [0,1]$. Par convention, la prédiction B_p d'une zone délimitante est considérée comme étant satisfaisante si l'on a $\text{IoU}(B_p, B_a) \geq 0.5$.

□ **Zone d'accroche** – La technique des zones d'accroche (en anglais *anchor boxing*) sert à prédrer des zones délimitantes qui se chevauchent. En pratique, on permet au réseau de prédrer plus d'une zone délimitante simultanément, où chaque zone prédict doit respecter une forme géométrique particulière. Par exemple, la première prédiction peut potentiellement être une zone rectangulaire d'une forme donnée, tandis qu'une seconde prédiction doit être une zone rectangulaire d'une autre forme.

□ **Suppression non-max** – La technique de suppression non-max (en anglais *non-max suppression*) a pour but d'enlever des zones délimitantes qui se chevauchent et qui prédissent un seul et même objet, en sélectionnant les zones les plus représentatives. Après avoir enlevé toutes les zones ayant une probabilité prédict de moins de 0.6, les étapes suivantes sont répétées pour éliminer les zones redondantes :

Pour une classe donnée,

- Étape 1 : Choisir la zone ayant la plus grande probabilité de prédiction.
- Étape 2 : Enlever toute zone ayant $\text{IoU} \geq 0.5$ avec la zone choisie précédemment.



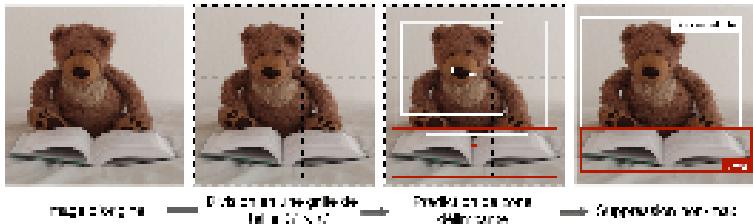
□ **YOLO** – L'algorithme You Only Look Once (YOLO) est un algorithme de détection d'objet qui fonctionne de la manière suivante :

- Étape 1 : Diviser l'image d'entrée en une grille de taille $G \times G$.
- Étape 2 : Pour chaque cellule, faire tourner un CNN qui prédit y de la forme suivante :

$$y = \left[\underbrace{p_c, b_x, b_y, b_h, b_w, c_1, c_2, \dots, c_p}_{\text{répété } k \text{ fois}}, \dots \right]^T \in \mathbb{R}^{G \times G \times k \times (5+p)}$$

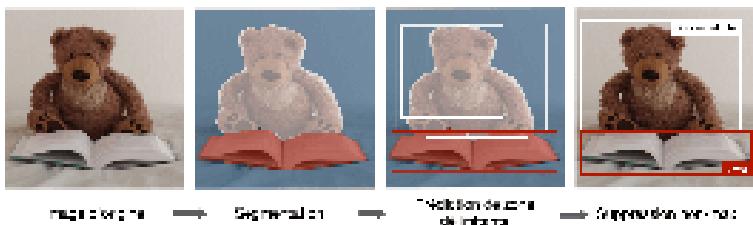
où p_c est la probabilité de détecter un objet, b_x, b_y, b_h, b_w sont les propriétés de la zone délimitante détectée, c_1, \dots, c_p est une représentation binaire (en anglais *one-hot representation*) de l'une des p classes détectée, et k est le nombre de zones d'accroche.

- Étape 3 : Faire tourner l'algorithme de suppression non-max pour enlever des doublons potentiels qui chevauchent des zones délimitantes.



Remarque : lorsque $p_c = 0$, le réseau ne détecte plus d'objet. Dans ce cas, les prédictions correspondantes b_x, \dots, c_p doivent être ignorées.

□ **R-CNN** – L'algorithme de région avec des réseaux de neurones convolutionnels (en anglais *Region with Convolutional Neural Networks*) (R-CNN) est un algorithme de détection d'objet qui segmente l'image d'entrée pour trouver des zones délimitantes pertinentes, puis fait tourner un algorithme de détection pour trouver les objets les plus probables d'apparaître dans ces zones délimitantes.



Remarque : bien que l'algorithme original soit lent et coûteux en temps de calcul, de nouvelles architectures ont permis de faire tourner l'algorithme plus rapidement, tels que le Fast R-CNN et le Faster R-CNN.

1.6.1 Vérification et reconnaissance de visage

□ **Types de modèles** – Deux principaux types de modèle sont récapitulés dans le tableau ci-dessous :

Vérification de visage	Reconnaissance de visage
<ul style="list-style-type: none"> - Est-ce la bonne personne ? - Un à un 	<ul style="list-style-type: none"> - Est-ce une des K personnes dans la base de données ? - Un à plusieurs

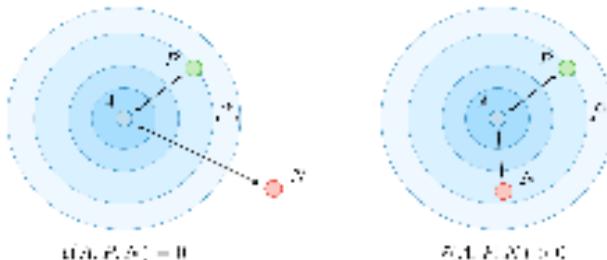
□ **Apprentissage par coup** – L'apprentissage par coup (en anglais *One Shot Learning*) est un algorithme de vérification de visage qui utilise un training set de petite taille pour apprendre

une fonction de similarité qui quantifie à quel point deux images données sont différentes. La fonction de similarité appliquée à deux images est souvent notée $d(\text{image 1}, \text{image 2})$.

□ **Réseaux siamois** – Les réseaux siamois (en anglais *Siamese Networks*) ont pour but d'apprendre comment encoder des images pour quantifier le degré de différence de deux images données. Pour une image d'entrée donnée $x^{(i)}$, l'encodage de sortie est souvent notée $f(x^{(i)})$.

□ **Loss triple** – Le loss triple (en anglais *triplet loss*) ℓ est une fonction de loss calculée sur une représentation encodée d'un triplet d'images A (accroche), P (positif), et N (négatif). L'exemple d'accroche et l'exemple positif appartiennent à la même classe, tandis que l'exemple négatif appartient à une autre. En notant $\alpha \in \mathbb{R}^+$ le paramètre de marge, le loss est défini de la manière suivante :

$$\ell(A, P, N) = \max(d(A, P) - d(A, N) + \alpha, 0)$$



1.6.2 Transfert de style neuronal

□ **Motivation** – Le but du transfert de style neuronal (en anglais *neural style transfer*) est de générer une image G à partir d'un contenu C et d'un style S .



□ **Activation** – Dans une couche l donnée, l'activation est notée $a^{[l]}$ et est de dimensions $n_H \times n_W \times n_C$.

□ **Fonction de coût de contenu** – La fonction de coût de contenu (en anglais *content cost function*), notée $J_{\text{content}}(C, G)$, est utilisée pour quantifier à quel point l'image générée G diffère de l'image de contenu original C . Elle est définie de la manière suivante :

$$J_{\text{content}}(C, G) = \frac{1}{2} \|a^{[l]}(C) - a^{[l]}(G)\|^2$$

□ **Matrice de style** – La matrice de style (en anglais *style matrix*) $G^{[l]}$ d'une couche l donnée est une matrice de Gram dans laquelle chacun des éléments $G_{kk'}^{[l]}$ quantifie le degré de corrélation des canaux k and k' . Elle est définie en fonction des activations $a^{[l]}$ de la manière suivante :

$$G_{kk'}^{[l]} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_w^{[l]}} a_{ijk}^{[l]} a_{ijk'}^{[l]}$$

Remarque : les matrices de style de l'image de style et de l'image générée sont notées $G^{[l](S)}$ and $G^{[l](G)}$ respectivement.

□ **Fonction de coût de style** – La fonction de coût de style (en anglais *style cost function*), notée $J_{\text{style}}(S, G)$, est utilisée pour quantifier à quel point l'image générée G diffère de l'image de style S . Elle est définie de la manière suivante :

$$J_{\text{style}}^{[l]}(S, G) = \frac{1}{(2n_H n_w n_c)^2} \|G^{[l](S)} - G^{[l](G)}\|_F^2 = \frac{1}{(2n_H n_w n_c)^2} \sum_{k,k'=1}^{n_c} \left(G_{kk'}^{[l](S)} - G_{kk'}^{[l](G)} \right)^2$$

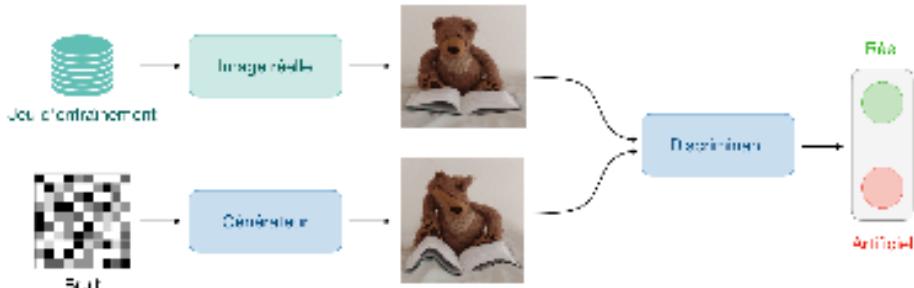
□ **Fonction de coût total** – La fonction de coût total (en anglais *overall cost function*) est définie comme étant une combinaison linéaire des fonctions de coût de contenu et de style, pondérées par les paramètres α, β , de la manière suivante :

$$J(G) = \alpha J_{\text{content}}(C, G) + \beta J_{\text{style}}(S, G)$$

Remarque : plus α est grand, plus le modèle privilégiera le contenu et plus β est grand, plus le modèle sera fidèle au style.

1.6.3 Architectures utilisant des astuces de calcul

□ **Réseau antagoniste génératif** – Les réseaux antagonistes génératifs (en anglais *generative adversarial networks*), aussi connus sous le nom de GANs, sont composés d'un modèle génératif et d'un modèle discriminatif, où le modèle génératif a pour but de générer des prédictions aussi réaliste que possible, qui seront ensuite envoyées dans un modèle discriminatif qui aura pour but de différencier une image générée d'une image réelle.



Remarque : les GANs sont utilisées dans des applications pouvant aller de la génération de musique au traitement de texte vers image.

□ **ResNet** – L'architecture du réseau résiduel (en anglais *Residual Network*), aussi appelé ResNet, utilise des blocs résiduels avec un nombre élevé de couches et a pour but de réduire l'erreur de training. Le bloc résiduel est caractérisé par l'équation suivante :

$$a^{[l+2]} = g(a^{[l]} + z^{[l+2]})$$

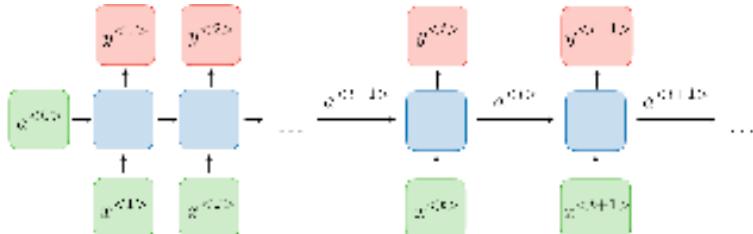
□ **Inception Network** – Cette architecture utilise des modules d'*inception* et a pour but de tester toute sorte de configuration de convolution pour améliorer sa performance en diversifiant ses attributs. En particulier, elle utilise l'astuce de la convolution 1×1 pour limiter sa complexité de calcul.

* * *

2 Réseaux de neurones récurrents

2.1 Vue d'ensemble

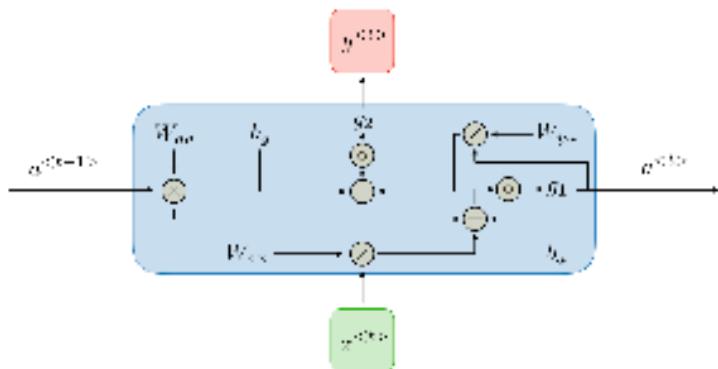
□ Architecture d'un RNN traditionnel – Les réseaux de neurones récurrents (en anglais *recurrent neural networks*), aussi appelés RNNs, sont une classe de réseaux de neurones qui permettent aux prédictions antérieures d'être utilisées comme entrées, par le biais d'états cachés (en anglais *hidden states*). Ils sont de la forme suivante :



À l'instant t , l'activation $a^{<t>}$ et la sortie $y^{<t>}$ sont de la forme suivante :

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \quad \text{et} \quad y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

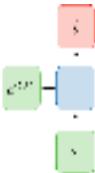
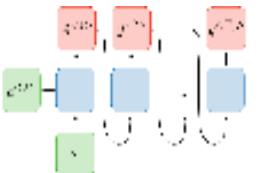
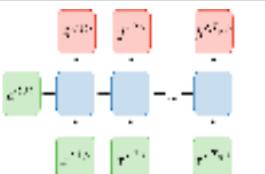
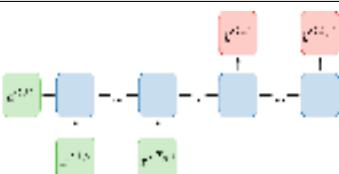
où $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ sont des coefficients indépendants du temps et où g_1, g_2 sont des fonctions d'activation.



Les avantages et inconvénients des architectures de RNN traditionnelles sont résumés dans le tableau ci-dessous :

Avantages	Inconvénients
<ul style="list-style-type: none"> - Prend en compte des entrées de toute taille - La taille du modèle n'augmente pas avec la taille de l'entrée - Les calculs prennent en compte les infos antérieures - Les coefficients sont indépendants du temps 	<ul style="list-style-type: none"> - Le temps de calcul est long - Difficulté d'accéder à des informations d'un passé lointain - Impossibilité de prendre en compte des informations futures un état donné

□ **Applications des RNNs** – Les modèles RNN sont surtout utilisés dans les domaines du traitement automatique du langage naturel et de la reconnaissance vocale. Le tableau suivant détaille les applications principales à retenir :

Type de RNN	Illustration	Exemple
Un à un $T_x = T_y = 1$		Réseau de neurones traditionnel
Un à plusieurs $T_x = 1, T_y > 1$		Génération de musique
Plusieurs à un $T_x > 1, T_y = 1$		Classification de sentiment
Plusieurs à plusieurs $T_x = T_y$		Reconnaissance d'entité
Plusieurs à plusieurs $T_x \neq T_y$		Traduction machine

□ **Fonction de loss** – Dans le contexte des réseaux de neurones récurrents, la fonction de loss \mathcal{L} prend en compte le loss à chaque temps T de la manière suivante :

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$$

□ **Backpropagation temporelle** – L'étape de backpropagation est appliquée dans la dimension temporelle. À l'instant T , la dérivée du loss \mathcal{L} par rapport à la matrice de coefficients W est donnée par :

$$\frac{\partial \mathcal{L}^{(T)}}{\partial W} = \sum_{t=1}^T \left. \frac{\partial \mathcal{L}^{(T)}}{\partial W} \right|_{(t)}$$

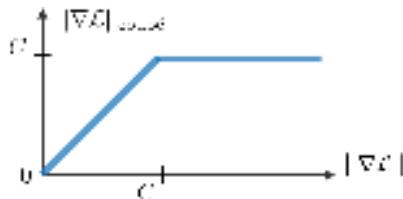
2.2 Dépendances à long terme

□ **Fonctions d'activation communément utilisées** – Les fonctions d'activation les plus utilisées dans les RNNs sont décrites ci-dessous :

Sigmoïde	Tanh	RELU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$

□ **Gradient qui disparaît/explose** – Les phénomènes de gradient qui disparaît et qui explose (en anglais *vanishing gradient* et *exploding gradient*) sont souvent rencontrés dans le contexte des RNNs. Ceci est dû au fait qu'il est difficile de capturer des dépendances à long terme à cause du gradient multiplicatif qui peut décroître/croître de manière exponentielle en fonction du nombre de couches.

□ **Coupeur de gradient** – Cette technique est utilisée pour atténuer le phénomène de gradient qui explose qui peut être rencontré lors de l'étape de backpropagation. En plafonnant la valeur qui peut être prise par le gradient, ce phénomène est maîtrisé en pratique.



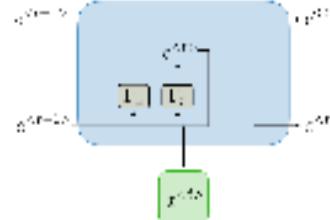
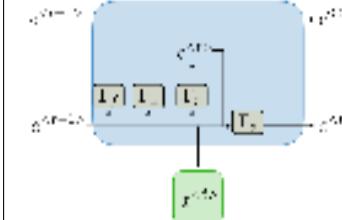
□ **Types de porte** – Pour remédier au problème du gradient qui disparaît, certains types de porte sont spécifiquement utilisés dans des variantes de RNNs et ont un but bien défini. Les portes sont souvent notées Γ et sont telles que :

$$\Gamma = \sigma(Wx^{<t>} + Ua^{<t-1>} + b)$$

où W, U, b sont des coefficients spécifiques à la porte et σ est une sigmoïde. Les portes à retenir sont récapitulées dans le tableau ci-dessous :

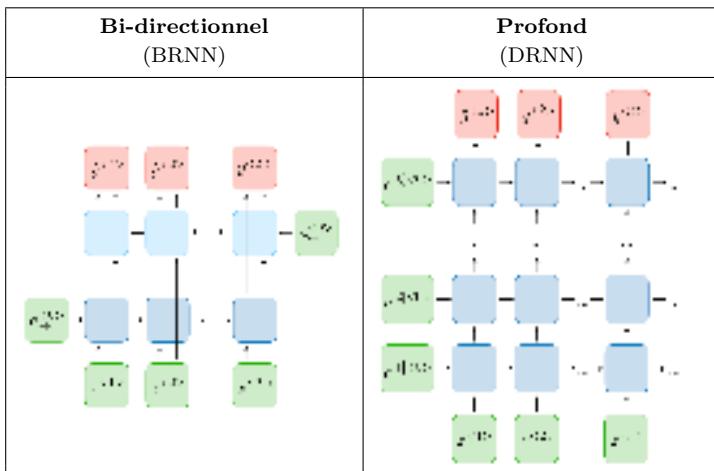
Type de porte	Rôle	Utilisée dans
Porte d'actualisation Γ_u	Dans quelle mesure le passé devrait être important ?	GRU, LSTM
Porte de pertinence Γ_r	Enlever les informations précédentes ?	GRU, LSTM
Porte d'oubli Γ_f	Enlever une cellule ?	LSTM
Porte de sortie Γ_o	Combien devrait-on révéler d'une cellule ?	LSTM

□ **GRU/LSTM** – Les unités de porte récurrente (en anglais *Gated Recurrent Unit*) (GRU) et les unités de mémoire à long/court terme (en anglais *Long Short-Term Memory units*) (LSTM) appaissent le problème du gradient qui disparaît rencontré par les RNNs traditionnels, où le LSTM peut être vu comme étant une généralisation du GRU. Le tableau ci-dessous résume les équations caractéristiques de chacune de ces architectures :

	Gated Recurrent Unit (GRU)	Long Short-Term Memory (LSTM)
$\tilde{c}^{<t>}$	$\tanh(W_c[\Gamma_r \star a^{<t-1>} \cdot x^{<t>}] + b_c)$	$\tanh(W_c[\Gamma_r \star a^{<t-1>} \cdot x^{<t>}] + b_c)$
$c^{<t>}$	$\Gamma_u \star \tilde{c}^{<t>} + (1 - \Gamma_u) \star c^{<t-1>}$	$\Gamma_u \star \tilde{c}^{<t>} + \Gamma_f \star c^{<t-1>}$
$a^{<t>}$	$c^{<t>}$	$\Gamma_o \star c^{<t>}$
Dépendances		

Remarque : le signe \star dénote le produit de Hadamard entre deux vecteurs.

□ **Variantes des RNNs** – Le tableau ci-dessous récapitule les autres architectures RNN communément utilisées :



2.3 Apprentissage de la représentation de mots

Dans cette section, on note V le vocabulaire et $|V|$ sa taille.

2.3.1 Motivation et notations

□ **Techniques de représentation** – Les deux manières principales de représenter des mots sont décrites dans le tableau suivant :

Représentation binaire	Représentation du mot
<ul style="list-style-type: none"> - Noté o_w - Approche naïve, pas d'information de similarité 	<ul style="list-style-type: none"> - Noté e_w - Prend en compte la similarité des mots

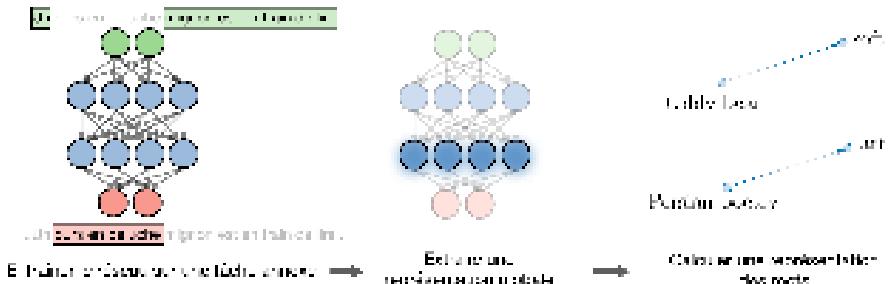
□ **Matrice de représentation** – Pour un mot donné w , la matrice de représentation (en anglais *embedding matrix*) E est une matrice qui relie une représentation binaire o_w à sa représentation correspondante e_w de la manière suivante :

$$e_w = E o_w$$

Remarque : l'apprentissage d'une matrice de représentation peut être effectué en utilisant des modèles probabilistiques de cible/contexte.

2.3.2 Représentation de mots

□ **Word2vec** – Word2vec est un ensemble de techniques visant à apprendre comment représenter les mots en estimant la probabilité qu'un mot donné a d'être entouré par d'autres mots. Le skip-gram, l'échantillonnage négatif et le CBOW font partie des modèles les plus populaires.



□ **Skip-gram** – Le skip-gram est un modèle de type supervisé qui apprend comment représenter les mots en évaluant la probabilité de chaque mot cible t donné dans un mot contexte c . En notant θ_t le paramètre associé à t , la probabilité $P(t|c)$ est donnée par :

$$P(t|c) = \frac{\exp(\theta_t^T e_c)}{\sum_{j=1}^{|V|} \exp(\theta_j^T e_c)}$$

Remarque : le fait d'additionner tout le vocabulaire dans le dénominateur du softmax rend le modèle coûteux en temps de calcul. CBOW est un autre modèle utilisant les mots avoisinants pour prédire un mot donné.

□ **Échantillonnage négatif** – Cette méthode utilise un ensemble de classificateurs binaires utilisant des régressions logistiques qui visent à évaluer dans quelle mesure des mots contexte et cible sont susceptible d'apparaître simultanément, avec des modèles étant entraînés sur des ensembles de k exemples négatifs et 1 exemple positif. Étant donné un mot contexte c et un mot cible t , la prédiction est donnée par :

$$P(y = 1|c, t) = \sigma(\theta_t^T e_c)$$

Remarque : cette méthode est moins coûteuse en calcul par rapport au modèle skip-gram.

□ **GloVe** – Le modèle GloVe (en anglais *global vectors for word representation*) est une technique de représentation des mots qui utilise une matrice de co-occurrence X où chaque $X_{i,j}$ correspond au nombre de fois qu'une cible i se produit avec un contexte j . Sa fonction de coût J est telle que :

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{|V|} f(X_{ij})(\theta_i^T e_j + b_i + b'_j - \log(X_{ij}))^2$$

où f est une fonction à coefficients telle que $X_{i,j} = 0 \implies f(X_{i,j}) = 0$.

Étant donné la symétrie que e et θ ont dans un modèle, la représentation du mot final $e_w^{(\text{final})}$ est donnée par :

$$e_w^{(\text{final})} = \frac{e_w + \theta_w}{2}$$

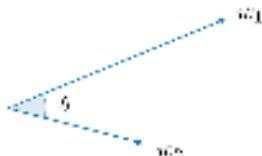
Remarque : les composantes individuelles de la représentation d'un mot n'est pas nécessairement facilement interprétable.

2.4 Comparaison de mots

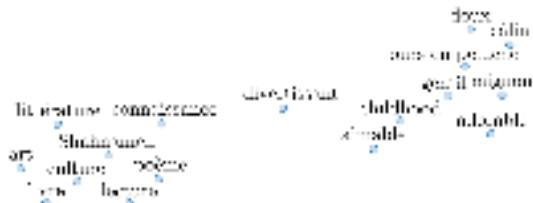
□ **Similarité cosinus** – La similarité cosinus (en anglais *cosine similarity*) entre les mots w_1 et w_2 est donnée par :

$$\text{similarity} = \frac{w_1 \cdot w_2}{\|w_1\| \|w_2\|} = \cos(\theta)$$

Remarque : θ est l'angle entre les mots w_1 et w_2 .



□ **t-SNE** – La méthode *t-SNE* (en anglais *t-distributed Stochastic Neighbor Embedding*) est une technique visant à réduire une représentation dans un espace de haute dimension en un espace de plus faible dimension. En pratique, on visualise les vecteur-mots dans un espace 2D.



2.5 Modèle de langage

□ **Vue d'ensemble** – Un modèle de langage vise à estimer la probabilité d'une phrase $P(y)$.

□ **Modèle n-gram** – Ce modèle consiste en une approche naïve qui vise à quantifier la probabilité qu'une expression apparaisse dans un corpus en comptabilisant le nombre de son apparition dans le training data.

□ **Perplexité** – Les modèles de langage sont communément évalués en utilisant la perplexité, aussi noté PP, qui peut être interprété comme étant la probabilité inverse des données normalisée par le nombre de mots T . La perplexité est telle que plus elle est faible, mieux c'est. Elle est définie de la manière suivante :

$$\text{PP} = \prod_{t=1}^T \left(\frac{1}{\sum_{j=1}^{|V|} y_j^{(t)} \cdot \hat{y}_j^{(t)}} \right)^{\frac{1}{T}}$$

Remarque : PP est souvent utilisée dans le cadre du t-SNE.

2.6 Traduction machine

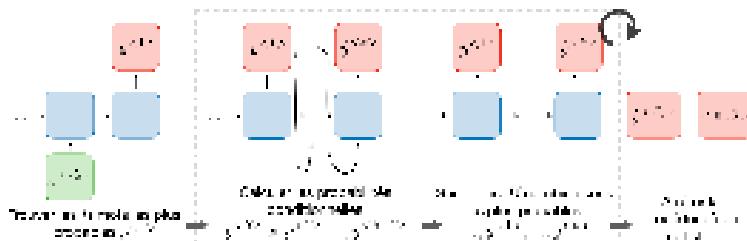
□ Vue d'ensemble – Un modèle de traduction machine est similaire à un modèle de langage ayant un auto-encodeur placé en amont. Pour cette raison, ce modèle est souvent surnommé modèle conditionnel de langage.

Le but est de trouver une phrase y telle que :

$$y = \arg \max_{y^{<1>} , \dots , y^{<T_y>}} P(y^{<1>} , \dots , y^{<T_y>} | x)$$

□ Recherche en faisceau – Cette technique (en anglais *beam search*) est un algorithme de recherche heuristique, utilisé dans le cadre de la traduction machine et de la reconnaissance vocale, qui vise à trouver la phrase la plus probable y sachant l'entrée x .

- Étape 1 : Trouver les B mots les plus probables $y^{<1>}$
- Étape 2 : Calculer les probabilités conditionnelles $y^{<k>} | x, y^{<1>} , \dots , y^{<k-1>}$
- Étape 3 : Garder les B combinaisons les plus probables $x, y^{<1>} , \dots , y^{<k>}$



Remarque : si la largeur du faisceau est prise égale à 1, alors ceci est équivalent à un algorithme glouton.

□ Largeur du faisceau – La largeur du faisceau (en anglais *beam width*) B est un paramètre de la recherche en faisceau. De grandes valeurs de B conduisent à avoir de meilleurs résultats mais avec un coût de mémoire plus lourd et à un temps de calcul plus long. De faibles valeurs de B conduisent à de moins bons résultats mais avec un coût de calcul plus faible. Une valeur de B égale à 10 est standard et est souvent utilisée.

□ Normalisation de longueur – Pour que la stabilité numérique puisse être améliorée, la recherche en faisceau utilise un objectif normalisé, souvent appelé l'objectif de log-probabilité normalisé, défini par :

$$\text{Objectif} = \frac{1}{T_g^\alpha} \sum_{t=1}^{T_y} \log \left[p(y^{<t>} | x, y^{<1>} , \dots , y^{<t-1>}) \right]$$

Remarque : le paramètre α est souvent comprise entre 0.5 et 1.

□ Analyse d'erreur – Lorsque l'on obtient une mauvaise traduction prédite \hat{y} , on peut se demander la raison pour laquelle l'algorithme n'a pas obtenu une bonne traduction y^* en faisant une analyse d'erreur de la manière suivante :

Cas	$P(y^* x) > P(\hat{y} x)$	$P(y^* x) \leq P(\hat{y} x)$
Cause	Recherche en faisceau défectueuse	RNN défectueux
Remèdes	Augmenter la largeur du faisceau	<ul style="list-style-type: none"> - Essayer une différente architecture - Régulariser - Obtenir plus de données

□ **Score bleu** – Le score bleu (en anglais *bilingual evaluation understudy*) a pour but de quantifier à quel point une traduction est bonne en calculant un score de similarité basé sur une précision n -gram. Il est défini de la manière suivante :

$$\text{score bleu} = \exp \left(\frac{1}{n} \sum_{k=1}^n p_k \right)$$

où p_n est le score bleu unique basé sur les n -gram, défini par :

$$p_n = \frac{\sum_{\substack{n\text{-gram} \in \hat{y}}} \text{count}_{\text{clip}}(\text{n-gram})}{\sum_{\substack{n\text{-gram} \in \hat{y}}} \text{count}(\text{n-gram})}$$

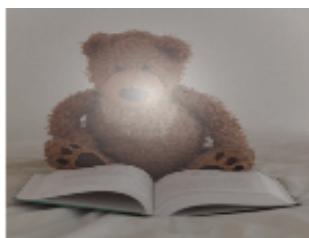
Remarque : une pénalité de brièveté peut être appliquée aux traductions prédictes courtes pour empêcher que le score bleu soit artificiellement haut.

2.7 Attention

□ **Modèle d'attention** – Le modèle d'attention (en anglais *attention model*) permet au RNN de mettre en valeur des parties spécifiques de l'entrée qui peuvent être considérées comme étant importantes, ce qui améliore la performance du modèle final en pratique. En notant $\alpha^{<t,t'>}$ la quantité d'attention que la sortie $y^{<t>}$ devrait porter à l'activation $a^{<t'>}$ et au contexte $c^{<t>}$ à l'instant t , on a :

$$c^{<t>} = \sum_{t'} \alpha^{<t,t'>} a^{<t'>} \quad \text{avec} \quad \sum_{t'} \alpha^{<t,t'>} = 1$$

Remarque : les scores d'attention sont communément utilisés dans la génération de légende d'image ainsi que dans la traduction machine.



Le ours en peluche regardait les mots de la légende pour leur donner plus de sens.



Le ours en peluche regardait les mots de la légende pour leur donner plus de sens.

□ **Coefficient d'attention** – La quantité d'attention que la sortie $y^{<t>}$ devrait porter à l'activation $a^{<t'>}$ est donné $\alpha^{<t,t'>}$, qui est calculé de la manière suivante :

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t''=1}^{T_x} \exp(e^{<t,t''>})}$$

Remarque : la complexité de calcul est quadratique par rapport à T_x .

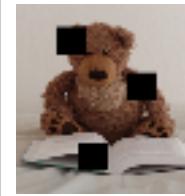
★ ★ ★

3 Petites astuces

3.1 Traitement des données

□ Augmentation des données – Les modèles d'apprentissage profond ont typiquement besoin de beaucoup de données afin d'être entraînés convenablement. Il est souvent utile de générer plus de données à partir de celles déjà existantes à l'aide de techniques d'augmentation de données. Celles les plus souvent utilisées sont résumées dans le tableau ci-dessous. À partir d'une image, voici les techniques que l'on peut utiliser :

Original	Symmétrie axiale	Rotation	Recadrage aléat.
			
- Image sans aucune modif	- Symmétrie par rapport à un axe pour lequel le sens de l'image est conservé	- Rotation avec un petit angle - Reproduit une calibration imparfaite de l'horizon	- Concentration aléatoire sur une partie de l'image - Plusieurs rognements aléatoires peuvent être faits à la suite

Ch. de couleur	Addition de bruit	Perte d'info.	Ch. de contraste
			
- Nuances de RGB sont légèrement changées - Capture le bruit qui peut survenir avec de l'exposition lumineuse	- Addition de bruit - Plus de tolérance envers la variation de la qualité de l'entrée	- Parties de l'image ignorées - Imité des pertes potentielles de parties de l'image	- Changement de luminosité - Contrôle la différence de l'exposition dû à l'heure de la journée

□ Normalisation de lot – La normalisation de lot (en anglais *batch normalization*) est une étape qui normalise le lot $\{x_i\}$ avec un choix de paramètres γ, β . En notant μ_B, σ_B^2 la moyenne et la variance de ce que l'on veut corriger du lot, on a :

$$x_i \leftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

Ceci est couramment fait après un fully connected/couche de convolution et avant une couche non-linéaire. Elle vise à permettre d'avoir de plus grands taux d'apprentissages et de réduire la dépendance à l'initialisation.

3.2 Entraîner un réseau de neurones

3.2.1 Définitions

Epoch – Dans le contexte de l'entraînement d'un modèle, l'*epoch* est un terme utilisé pour référer à une itération où le modèle voit tout le training set pour mettre à jour ses coefficients.

Gradient descent sur mini-lots – Durant la phase d'entraînement, la mise à jour des coefficients n'est souvent basée ni sur tout le training set d'un coup à cause de temps de calculs coûteux, ni sur un seul point à cause de bruits potentiels. À la place de cela, l'étape de mise à jour est faite sur des mini-lots, où le nombre de points dans un lot est un paramètre que l'on peut régler.

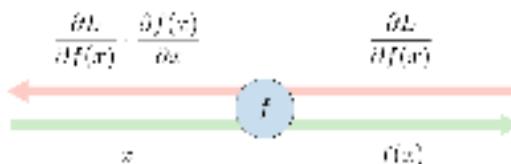
Fonction de loss – Pour pouvoir quantifier la performance d'un modèle donné, la fonction de loss (en anglais *loss function*) L est utilisée pour évaluer la mesure dans laquelle les sorties vraies y sont correctement prédites par les prédictions du modèle z .

Entropie croisée – Dans le contexte de la classification binaire d'un réseau de neurones, l'entropie croisée (en anglais *cross-entropy loss*) $L(z,y)$ est couramment utilisée et est définie par :

$$L(z,y) = - \left[y \log(z) + (1-y) \log(1-z) \right]$$

3.2.2 Recherche de coefficients optimaux

Backpropagation – La backpropagation est une méthode de mise à jour des coefficients d'un réseau de neurones en prenant en compte les sorties vraies et désirées. La dérivée par rapport à chaque coefficient w est calculée en utilisant la règle de la chaîne.

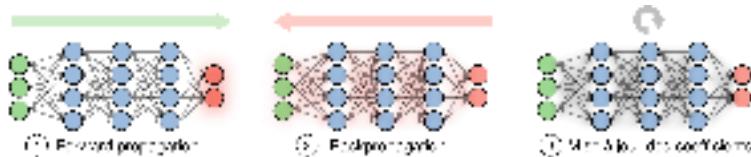


En utilisant cette méthode, chaque coefficient est mis à jour par :

$$w \leftarrow w - \alpha \frac{\partial L(z,y)}{\partial w}$$

Mise à jour les coefficients – Dans un réseau de neurones, les coefficients sont mis à jour par :

- Étape 1 : Prendre un lot de training data et effectuer une forward propagation pour calculer le loss.
- Étape 2 : Backpropaguer le loss pour obtenir le gradient du loss par rapport à chaque coefficient.
- Étape 3 : Utiliser les gradients pour mettre à jour les coefficients du réseau.

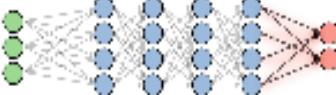
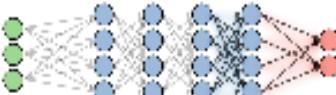


3.3 Réglage des paramètres

3.3.1 Initialisation des coefficients

□ **Initialisation de Xavier** – Au lieu de laisser les coefficients s'initialiser de manière purement aléatoire, l'initialisation de Xavier permet d'avoir des coefficients initiaux qui prennent en compte les caractéristiques uniques de l'architecture.

□ **Apprentissage par transfert** – Entraîner un modèle d'apprentissage profond requiert beaucoup de données et beaucoup de temps. Il est souvent utile de profiter de coefficients pré-entraînés sur des données énormes qui ont pris des jours/semaines pour être entraînés, et profiter de cela pour notre cas. Selon la quantité de données que l'on a sous la main, voici différentes manières d'utiliser cette méthode :

Taille du training	Illustration	Explication
Petite		Gèle toutes les couches, entraîne les coefficients du softmax
Moyenne		Gèle la plupart des couches, entraîne les coefficients des dernières couches et du softmax
Grande		Entraîne les coefficients des couches et du softmax en initialisant les coefficients sur ceux qui ont été pré-entraînés

3.3.2 Optimisation de la convergence

□ **Taux d'apprentissage** – Le taux d'apprentissage (en anglais *learning rate*), souvent noté α ou η , indique la vitesse à laquelle les coefficients sont mis à jour. Il peut être fixe ou variable. La

méthode actuelle la plus populaire est appelée Adam, qui est une méthode faisant varier le taux d'apprentissage.

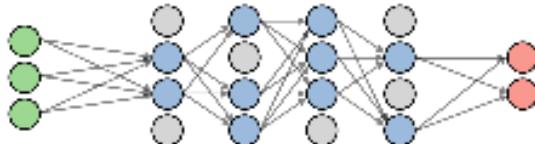
□ **Taux d'apprentissage adaptatifs** – Laisser le taux d'apprentissage varier pendant la phase d'entraînement du modèle peut réduire le temps d'entraînement et améliorer la qualité de la solution numérique optimale. Bien que la méthode d'Adam est la plus utilisée, d'autres peuvent aussi être utiles. Les différentes méthodes sont récapitulées dans le tableau ci-dessous :

Méthode	Explication	Mise à jour de w	Mise à jour de b
Momentum	- Amortit les oscillations - Amélioration par rapport à la méthode SGD - 2 paramètres à régler	$w \leftarrow w - \alpha v_{dw}$	$b \leftarrow b - \alpha v_{db}$
RMSprop	- Root Mean Square propagation - Accélère l'algorithme d'apprentissage en contrôlant les oscillations	$w \leftarrow w - \alpha \frac{dw}{\sqrt{s_{dw}}}$	$b \leftarrow b - \alpha \frac{db}{\sqrt{s_{db}}}$
Adam	- Adaptive Moment estimation - Méthode la plus populaire - 4 paramètres à régler	$w \leftarrow w - \alpha \frac{v_{dw}}{\sqrt{s_{dw}} + \epsilon}$	$b \leftarrow b - \alpha \frac{v_{db}}{\sqrt{s_{db}} + \epsilon}$

Remarque : parmi les autres méthodes existantes, on trouve Adadelta, Adagrad et SGD.

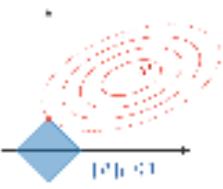
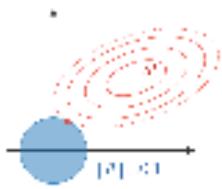
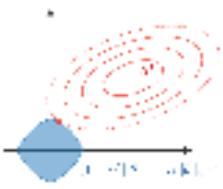
3.4 Régularisation

□ **Dropout** – Le dropout est une technique qui est destinée à empêcher le sur-ajustement sur les données de training en abandonnant des unités dans un réseau de neurones avec une probabilité $p > 0$. Cela force le modèle à éviter de trop s'appuyer sur un ensemble particulier de features.

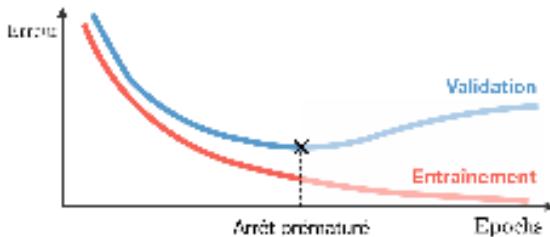


Remarque : la plupart des frameworks d'apprentissage profond paramètrent le dropout à travers le paramètre 'garder' $1 - p$.

□ **Régularisation de coefficient** – Pour s'assurer que les coefficients ne sont pas trop grands et que le modèle ne sur-ajuste pas sur le training set, on utilise des techniques de régularisation sur les coefficients du modèle. Les techniques principales sont résumées dans le tableau suivant :

LASSO	Ridge	Elastic Net
- Réduit les coefficients à 0 - Bon pour la sélection de variables	Rend les coefficients plus petits	Compromis entre la sélection de variables et la réduction de la taille des coefficients
		
$\dots + \lambda \ \theta\ _1$ $\lambda \in \mathbb{R}$	$\dots + \lambda \ \theta\ _2^2$ $\lambda \in \mathbb{R}$	$\dots + \lambda \left[(1 - \alpha) \ \theta\ _1 + \alpha \ \theta\ _2^2 \right]$ $\lambda \in \mathbb{R}, \alpha \in [0,1]$

□ **Arrêt prématûré** – L'arrêt prématûré (en anglais *early stopping*) est une technique de régularisation qui consiste à stopper l'étape d'entraînement dès que le loss de validation atteint un plateau ou commence à augmenter.



3.5 Bonnes pratiques

□ **Surapprentissage d'un mini-lot** – Lorsque l'on débuge un modèle, il est souvent utile de faire de petits tests pour voir s'il y a un gros souci avec l'architecture du modèle lui-même. En particulier, pour s'assurer que le modèle peut être entraîné correctement, un mini-lot est passé dans le réseau pour voir s'il peut sur-ajuster sur lui. Si le modèle ne peut pas le faire, cela signifie que le modèle est soit trop complexe ou pas assez complexe pour être sur-ajusté sur un mini-lot.

□ **Vérification de gradient** – La méthode de la vérification de gradient (en anglais *gradient checking*) est utilisée durant l'implémentation d'un backward pass d'un réseau de neurones. Elle compare la valeur du gradient analytique par rapport au gradient numérique au niveau de certains points et joue un rôle de vérification élémentaire.

	Gradient numérique	Gradient analytique
Formule	$\frac{df}{dx}(x) \approx \frac{f(x+h) - f(x-h)}{2h}$	$\frac{df}{dx}(x) = f'(x)$
Commentaires	<ul style="list-style-type: none"> - Coûteux ; le loss doit être calculé deux fois par dimension - Utilisé pour vérifier l'exactitude d'une implémentation analytique - Compromis dans le choix de h entre pas trop petit (instabilité numérique) et pas trop grand (estimation du gradient approximative) 	<ul style="list-style-type: none"> - Résultat 'exact' - Calcul direct - Utilisé dans l'implémentation finale

* * *