



**HAL**  
open science

# Apprentissage profond sans supervision directe pour le traitement automatique des langues

Paul Caillon

► **To cite this version:**

Paul Caillon. Apprentissage profond sans supervision directe pour le traitement automatique des langues. Informatique [cs]. Université de Lorraine, 2023. Français. NNT : 2023LORR0406 . tel-04814022

**HAL Id: tel-04814022**

**<https://hal.univ-lorraine.fr/tel-04814022v1>**

Submitted on 2 Dec 2024

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



**UNIVERSITÉ  
DE LORRAINE**

**BIBLIOTHÈQUES  
UNIVERSITAIRES**

## AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

Contact bibliothèque : [ddoc-theses-contact@univ-lorraine.fr](mailto:ddoc-theses-contact@univ-lorraine.fr)  
*(Cette adresse ne permet pas de contacter les auteurs)*

## LIENS

Code de la Propriété Intellectuelle. articles L 122. 4

Code de la Propriété Intellectuelle. articles L 335.2- L 335.10

[http://www.cfcopies.com/V2/leg/leg\\_droi.php](http://www.cfcopies.com/V2/leg/leg_droi.php)

<http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm>

---

# Apprentissage profond sans supervision directe pour le traitement automatique des langues

## THÈSE

présentée et soutenue publiquement le 31 mai 2023

Pour l'obtention du titre de

Docteur de l'Université de Lorraine  
(mention informatique)

par  
Paul Caillon

sous la direction de  
Christophe Cerisara

### Composition du Jury

Marianne Clausel	PR, Université de Lorraine	<b>Présidente et Examinatrice</b>
Alexandre Allauzen	PR, Université Paris-Dauphine et EPSCI, PSL	<b>Rapporteur</b>
François Portet	PR, Université Grenoble Alpes	<b>Rapporteur</b>
Madalina Olteanu	PR, Université Paris Dauphine PSL	<b>Examinatrice</b>
Christophe Cerisara	CR, Université de Lorraine	<b>Directeur de Thèse</b>



# REMERCIEMENTS

Cette section est dédiée à toutes les personnes se sentant concernées et il me sera difficile d'être exhaustif.

Je veux tout d'abord remercier mon directeur de thèse, Christophe Cerisara, pour son aide et son encadrement. J'ai été ravi de travailler avec lui et outre sa précieuse aide scientifique et son support, nos discussions (*parfois philosophiques*) ont toujours été passionnantes. Malgré les difficultés liées à un environnement sanitaire parfois contrariant, il a toujours été disponible pour m'aiguiller et m'encourager à poursuivre des pistes de recherche originales.

Je tiens également à remercier Marianne Clausel, Madalina Olteanu, Alexandre Allauzen et François Portet de m'avoir fait l'honneur d'être membres de mon jury. Merci aux rapporteurs, Alexandre Allauzen et François Portet, dont les rapports m'ont permis d'envisager mon travail sous un angle différent, ouvrant ainsi mes perspectives de recherche et aux examinatrices, Marianne Clausel et Madalina Olteanu pour leurs remarques et participation scientifiques. Je vous remercie pour le temps que vous avez consacré à mon travail et pour les pistes que vous m'avez ouvertes lors de nos échanges.

Je remercie également toutes les personnes avec qui j'ai partagé mes années de thèse, dans l'équipe SYNALP et plus largement au LORIA et aux Mines de Nancy. À ce titre, j'adresse un remerciement particulier à l'équipe enseignante du département informatique de l'École des Mines de Nancy de m'avoir accueilli comme un membre à part entière pendant mon année d'ATER (et mes années de chargé de cours/TD). À ce titre, un grand merci à ceux avec qui j'ai pu travailler directement, Xavier Goaoc, Pierre-Etienne Moreau, Cédric Zanni, Laurent Ciarletta et Guillaume Bonfante.

J'adresse évidemment de profonds remerciements à mes amis et ma famille qui ont su me soutenir et m'encourager avec patience et compréhension malgré l'austérité occasionnelle de ma présence. Celles et ceux m'ayant accompagné durant ces 4 années que ce soit à l'occasion d'évènements festifs, culturels, ou encore durant les confinements, se reconnaîtront sans peine. Je rends ici un hommage tout particulier à ceux qui ont voulu et pu se libérer un 31 mai pour assister à ma soutenance à Nancy, cette présence a été d'un grand réconfort.

À l'éventuel lecteur enfin, merci pour l'intérêt accordé à ce manuscrit, j'espère qu'il méritera le temps accordé à sa lecture.



# RÉSUMÉ DE THÈSE

La profondeur des réseaux de neurones n'est plus l'aspect le plus important des systèmes d'apprentissage en profondeur de nos jours. Il s'agit plutôt de la possibilité de construire des fonctions de plus en plus abstraites et complexes implémentées sous forme de programmes informatiques paramétriques différentiables. La principale question concerne l'accès à des données annotées suffisantes pour apprendre cette fonction, ce qui devient critique. Par conséquent, de nos jours, la distinction standard entre l'apprentissage supervisé et non supervisé devient de plus en plus floue, car tout corpus annoté existant est inévitablement trop petit et statique pour représenter avec précision les informations les plus récentes.

Un autre problème émergent est l'échelle des tailles de modèle utilisées pour atteindre des performances de pointe. Par exemple, en traitement du langage naturel, des super-modèles sont pré-entraînés, puis affinés pour des tâches spécifiques (Bert, XLM-R ou plus récemment PaLM et GPT-4). Les représentations génériques apprises par ces modèles sont ensuite adaptées aux tâches spécifiques à l'aide de méthodes d'apprentissage à quelques exemples adéquates.

Certaines méthodes récentes réduisent les tailles des modèles appris après la phase d'entraînement tout en obtenant les mêmes performances, afin d'obtenir des modèles d'apprentissage automatique plus petits et éco-énergétiques utilisables sur des dispositifs à performances de calcul limitées (voir les méthodes d'élagage de réseau et de distillation). D'autre part, faire croître un petit réseau à la fois en largeur et en profondeur permet également d'apprendre des réseaux précis et relativement petits, atteignant des performances de pointe.

Dans cette thèse, nous nous concentrons plus particulièrement sur le deuxième type d'approches. Nous travaillons sur des modèles capables d'augmenter progressivement la taille de la mémoire du réseau neuronal pour traiter une quantité croissante de données observées et observer la différence avec des modèles complets entraînés à partir de zéro, en mettant l'accent sur la compréhension des raisons pour lesquelles la croissance progressive conduit à des performances comparables avec moins de paramètres. Nous travaillons également sur de nouvelles fonctions de perte non supervisées ou faiblement supervisées qui permettent de former des modèles génératifs qui résolvent le manque de généralité de la plupart des méthodes d'incorporation actuelles.

Nous proposons ainsi un modèle simple de grossissement permettant d'ajouter progressivement des nouveaux paramètres au cours de l'apprentissage afin d'étudier les comportements propres à ces réseaux dynamiques. Plus précisément, nous comparons les aspects des surfaces des fonctions de coût des réseaux standards et grossissants afin d'expliquer les performances de ces derniers. Pour les étudier plus en détails, nous développons également une approximation du risque théorique qui peut être utilisée à la fois comme une mesure de généralisation et comme une fonction de coût non supervisée.

**Mots-clés**— Apprentissage Automatique, Apprentissage Supervisé, Apprentissage non Supervisé, Réseaux de Neurones Grossissants, Généralisation, Flatness, TAL



# TABLE DES MATIÈRES

<b>Remerciements</b>	<b>i</b>
<b>Résumé de thèse</b>	<b>iii</b>
<b>Table des matières</b>	<b>vi</b>
<b>Liste des figures</b>	<b>viii</b>
<b>Liste des tableaux</b>	<b>x</b>
<b>Introduction</b>	<b>xi</b>
<b>I Étude de la taille des réseaux de neurones</b>	<b>1</b>
<b>1 Chapitre 1 : Étude de l'état de l'art : trouver une architecture optimale</b>	<b>3</b>
1.1 Revue historique des architectures de réseaux de neurones . . . . .	4
1.2 Trouver automatiquement une architecture optimisée : le cas du Neural Architecture Search (NAS) . . . . .	19
1.3 Les réseaux de neurones à architecture dynamique . . . . .	24
<b>2 Chapitre 2 : Réseaux de neurones expansifs : modèle et implémentation</b>	<b>31</b>
2.1 Vers un cadre général pour les architectures expansives . . . . .	32
2.2 Expansion contrôlée : atteindre une architecture prédéfinie . . . . .	39
<b>3 Chapitre 3 : Réseaux de neurones expansifs : résultats et analyse</b>	<b>41</b>
3.1 Résultats et analyse de l'initialisation des nouveaux neurones . . . . .	42
3.2 Comparaison des réseaux expansifs avec des réseaux standards à architecture donnée . . . . .	45
<b>II Étude de la surface de la fonction de coût des réseaux de neurones</b>	<b>53</b>
<b>4 Chapitre 4 : Étude de l'État de l'Art : les Surfaces des fonctions de coût</b>	<b>55</b>
4.1 Les Surfaces des fonctions de coût des ANNs . . . . .	56
4.2 Planéité et Performances . . . . .	57
<b>5 Chapitre 5 : Une manière particulière d'explorer l'espace des paramètres : les réseaux de neurones expansifs</b>	<b>59</b>
5.1 Notations et Hypothèses . . . . .	61
5.2 Expression Théorique : Comparaison des volumes des minima . . . . .	62
<b>6 Chapitre 6 : Les réseaux de neurones expansifs convergent vers des minima plats</b>	<b>67</b>
6.1 Détails des conditions expérimentales . . . . .	68

---

6.2	Résultats et Analyse . . . . .	69
<b>III</b>	<b>Étude des performances de généralisation</b>	<b>73</b>
<b>7</b>	<b>Chapitre 7 : Mesures de Généralisation et Évaluation de Performances : État de l'Art</b>	<b>75</b>
7.1	Comment mesurer la généralisation d'un ANN . . . . .	76
7.2	Au delà du surapprentissage pour les LLM . . . . .	77
<b>8</b>	<b>Chapitre 8 : Risque Non Supervisé : Mesure de Généralisation et Méthode de Régularisation</b>	<b>79</b>
8.1	Risque Non Supervisé : Présentation Théorique . . . . .	80
8.2	Le Risque Non Supervisé comme Méthode de Régularisation : le post-tuning	85
<b>9</b>	<b>Chapitre 9 : Les réseaux de neurones expansifs généralisent-ils mieux?</b>	<b>93</b>
9.1	Le Risque Non Supervisé comme Mesure de Généralisation . . . . .	94
9.2	Résultats expérimentaux . . . . .	104
9.3	Discussion . . . . .	107
	<b>Conclusion</b>	<b>109</b>
	<b>Bibliographie</b>	<b>128</b>
	<b>Table des annexes</b>	<b>129</b>
A	Évolution des métriques au cours de l'entraînement . . . . .	131

# TABLE DES FIGURES

1.1	Illustration du Perceptron, d'après ROSENBLATT [162] . . . . .	5
1.2	Calculs logiques et limites du perceptron (adapté du cours "Introduction à l'Apprentissage Automatique", F. Sur – Mines Nancy) . . . . .	5
1.3	Illustration d'un MLP à une couche cachée. L'information en entrée est ici un vecteur de $\mathbb{R}^n$ et en sortie un vecteur de $\mathbb{R}^s$ . Avec $n$ neurones en entrée, $h$ dans la couche cachée et $s$ neurones en sortie, le nombre de paramètres (poids et biais) du modèle ainsi présenté est $(n + 1) \cdot h + (h + 1) \cdot s$ (chaque neurone introduit un paramètre supplémentaire, appelé biais, d'où le "+1").	6
1.4	Quelques fonctions d'activation. . . . .	7
1.5	Architecture d'AlexNet, d'après KRIZHEVSKY, SUTSKEVER et HINTON [112] : premier réseau de neurones profond à battre les méthodes de vision par ordinateur classiques sur ImageNet (classification d'images à 1000 classes).	9
1.6	Architecture typique d'un AutoEncodeur. Dans cette illustration, une entrée est composée de 8 données que l'encodeur réduit à 3 dans sa représentation latente. Il décode ensuite l'information encodée dans la partie décodeur. . .	11
1.7	Exemple de max-pooling de facteur 2 : chaque bloc de dimension 2x2 est remplacé par une unique valeur égale au maximum des valeurs du bloc considéré. On met donc en relation des neurones présents dans un "patch" de données deux fois plus étendu (adapté du cours "Introduction à l'Apprentissage Automatique", F. Sur – Mines Nancy) . . . . .	12
1.8	Architecture de VGG16, d'après SIMONYAN et ZISSERMAN [174]. L'entrée est constituée de $224 \times 224 \times 3 = 150.528$ neurones, ce qui correspond à une image couleur (trois canaux : (R,G,B)) de $224 \times 224$ pixels. Les deux premiers blocs de convolutions sont constitués de 64 filtres indépendants de support de taille $3 \times 3 \times 3$ , chacun étant suivi d'une activation ReLU. Un pixel sur deux est ensuite gardé dans le max pooling. Les blocs de convolution et de pooling se succèdent jusqu'à atteindre FFNN classique constitué de deux couches de 4096 neurones, une couche de 1000 neurones, puis une couche SoftMax de 1000 neurones pour adapter la représentation des données apprises $\Phi(x)$ à la tâche de classification à 100 classes. Ce réseau possède 138 millions de paramètres. . . . .	13
1.9	Architecture d'une couche de LSTM, adapté de : <a href="#">d2l</a> . . . . .	14
1.10	Architecture résumée de la plupart des combinaisons des briques élémentaires présentées. Les CNN sont utilisés pour extraire les features des données d'entrée. Les RNN permettent d'extraire des liens temporels. Les MLP permettent une projection vers les sorties. . . . .	15
1.11	Différentes variantes de LRCN modèle proposé par DONAHUE et al. [45] . .	15
1.12	Illustration de la self-attention et de la multi-head attention, figures tirées du travail de VASWANI et al. [191] . . . . .	17
1.13	Architecture originale d'un Transformer. Sur la moitié gauche est représenté l'encodeur et sur la moitié droite le décodeur. Dans les travaux originaux de VASWANI et al. [191], le nombre de couches identiques $N=6$ . . . . .	18

---

1.14	Illustration des méthodes de NAS, d'après ELSKEN, METZEN et HUTTER [50]. Une stratégie de recherche sélectionne une architecture $A$ issue d'un espace de recherche prédéfini $\mathcal{A}$ . Cette architecture est passée à une stratégie d'évaluation de la performance qui retourne la performance estimée de $A$ à la stratégie de recherche. . . . .	19
1.15	Illustration des espaces de recherche à structures chaînées (à gauche), à branches multiples (centre gauche) et cellulaires (centre droite et droite). Chaque noeud dans les graphes correspond à une couche quelconque d'ANN (convolution, FFNN, ...). Différentes couleurs sont associées aux divers types de couches. Un lien d'une couche $L_i$ à $L_j$ exprime que $L_j$ reçoit en entrée la sortie de $L_i$ . Pour les espaces cellulaires, deux différents types de cellules sont présentés. Une cellule normale (en haut) et une cellule de réduction (en bas), type de cellules introduit par ZOPH et al. [218] dont la particularité est de réduire la dimension de l'entrée. En empilant les cellules en séquences, une architecture comme présentée sur le graphe de droite peut être obtenue. Il s'agit de noter que les cellules sont combinées de manière chaînée mais qu'elles peuvent être également combinées à la manière des combinaisons de l'espace à branches multiples en remplaçant simplement les couches d'ANNs par des cellules. . . . .	21
1.16	Illustration de l'élagage de poids(synapses) vs élagage de neurones. . . . .	26
1.17	Élagage itératif proposé par MOLCHANOV et al. [138]. . . . .	26
1.18	Framework générique de distillation, par GOU et al. [66]. . . . .	28
2.1	Exemple d'héritage parent/enfant avant et après insertion : les nouveaux poids sont : $\{ w'_{1,7} \sim \mathcal{U}(-0.5, 0.5), w'_{2,7} \sim \mathcal{U}(-0.5, 0.5), w'_{3,7} \sim \mathcal{U}(-0.5, 0.5)w'_{7,5} = w_{3,5}$ . . . . .	33
2.2	Exemple d'une insertion : $h_2^{(1)}$ est choisi aléatoirement et le nouveau neurone $h_1^{(2)}$ , hérite des connexions de sortie de $h_2^{(1)}$ (arcs rouges) et obtient des nouvelles connexions en entrées (arcs verts) initialisées aléatoirement. . .	36
3.1	Exemple d'héritage parent/enfant avant et après insertion avec l'initialisation assurant la continuité : $\{ w'_{1,7} = 0, w'_{2,7} = 0w'_{3,7} = 1w'_{7,5} = w_{3,5}$ . . . . .	42
3.2	Illustration de l'effet de l'expansion avec "initialisation continue" sur la fonction de coût d'un FFNN. . . . .	43
3.3	Illustration de l'effet d'une insertion de neurones répétée toutes les 40 époques dans un FFNN initialement à une couche cachée à fonction d'activation ReLU sur la fonction de coût. . . . .	44
8.1	Risque non supervisé comme fonction de $(\mu_0, \mu_1)$ (gauche), et seulement $\mu_0$ (droite) pour $\mu_1 = 2, \sigma_1 = 1$ et $\sigma_0 \in \{0.1, 1, 3\}$ . . . . .	83
8.2	Modèle de détection d'anomalies textuelles; adapté de RUFF et al. [164]. Les $r$ couches linéaires normalisées implémentent l'Équation 8.4. . . . .	87
9.1	Schéma de grossissement utilisé pour atteindre les modèles décrits. . . . .	105

# LISTE DES TABLEAUX

3.1	Exactitude de Test pour différents neurones post entraînement. . . . .	46
3.2	Performances comparées d'un RoBERTa standard et expansif sur COLA . . .	48
3.3	Performances comparées d'un RoBERTa sans et avec expansion sur les tâches de MetaEval. La colonne <i>Exactitude de test</i> reporte la valeur d'exactitude de test du modèle sur la tâche spécifique juste après l'entraînement sur la tâche concernée. Ces valeurs sont des moyennes sur 10 runs où l'ordre de traitement des tâches a été tiré aléatoirement à chaque fois. La variance moyenne est de 0.03. . . . .	49
6.1	Exactitude de test et norme spectrale aux minima pour différents réseaux de neurones. . . . .	69
6.2	Exactitude de test et Norme Spectrale aux Minima sur COLA . . . . .	70
8.1	Résultats d'aire sous la courbe (AUC) sur de la détection d'anomalies textuelles : les nombres en police normale dans les colonnes CVDD sont obtenus en ré-appliquant le code de RUFF et al. [164]; les nombres correspondant issus de l'article original sont reportés en italique et entre parenthèses : il peut y avoir quelques différences, issues probablement d'environnements expérimentaux différents (GPUs, versions de bibliothèques...). Les colonnes "Eq 8.2" correspondent à notre approche. Il s'agit de noter que seule la fonction de risque change, tous les hyper-paramètres (modèle, nombre de paramètres, ...) restent identiques par ailleurs. . . . .	88
9.1	Corpus : Offensive, $p_0 = 0.6693$ (relativement équilibré). Le modèle commence à surapprendre après l'époque 8 ; le risque proposé détecte l'époque 9 comme la meilleure (uniquement à partir du corpus d'entraînement), faisant une "erreur" d'une seule époque. . . . .	97
9.2	Corpus : Political Media Audience, $p_0 = 0.2099$ (assez déséquilibré). Le modèle commence à surapprendre après l'époque 8 qui est également prédite comme la meilleure par le risque proposé. . . . .	98
9.3	Résultats sur MetaEval : <b>maxACC</b> est la borne supérieure d'exactitude (l'oracle), qui a lieu à l'époque <b>maxE</b> ; <b>empACC</b> est l'exactitude lorsque le coût empirique est minimal, principalement à l'époque 15 ; <b>riskACC</b> est l'exactitude à l'époque <b>riskE</b> , qui est le critère proposé d'early stopping calculé sur le corpus d'entraînement. Les nombres en gras sont les plus proches de la borne supérieure. . . . .	99
9.4	Statistiques de jeux de données de PMLB . . . . .	100
9.6	Évaluation des réseaux de tailles diverses proposés sur CoLA . Le risque empirique et non supervisé ainsi que la norme spectrale sont calculés sur le corpus de train. Le corps de l'architecture est RoBERTa large. . . . .	106
9.8	Évaluation des réseaux proposés sur certains corpora de Metaeval. Le risque empirique et non supervisé ainsi que la norme spectrale sont calculés sur le corpus de train. Le corps de l'architecture est RoBERTa large. . . . .	106
9	Commonsense, $p_0 = 0.5432$ . . . . .	131
10	Justice, $p_0 = 0.4571$ . . . . .	132

---

11	Virtue, $p_0 = 0.9145$	132
12	Binary, $p_0 = 0.5735$	133
13	Emobank-Arousal, $p_0 = 0.5152$	133
14	Persuasiveness-Eloquence, $p_0 = 0.7338$	134
15	Persuasiveness-Relevance, $p_0 = 0.4014$	134
16	Persuasiveness-Specificity, $p_0 = 0.5496$	135
17	Persuasiveness-Strength, $p_0 = 0.6173$	135
18	Emobank-Dominance, $p_0 = 0.6123$	136
19	Squinky-Implicature, $p_0 = 0.5328$	136
20	Sarcasm, $p_0 = 0.5024$	137
21	Squinky-Formality, $p_0 = 0.5191$	137
22	Squinky-Informativeness, $p_0 = 0.5410$	138
23	Emobank-valence, $p_0 = 0.5697$	138
24	Paws, $p_0 = 0.5581$	139
25	Tweet Global Warning, $p_0 = 0.7357$	139
26	Political Media Audience, $p_0 = 0.2099$	140
27	Political Media Bias, $p_0 = 0.2611$	140
28	CoLA, $p_0 = 0.2956$	141
29	SST-2, $p_0 = 0.4422$	141
30	MRPC, $p_0 = 0.3255$	142
31	QNLI, $p_0 = 0.5001$	142
32	RTE, $p_0 = 0.5016$	143
33	Syntax Semantics, $p_0 = 0.4991$	143
34	Syntax+Semantics, $p_0 = 0.4907$	144
35	Morphology, $p_0 = 0.5015$	144
36	Syntax, $p_0 = 0.4988$	145
37	Recast Puns, $p_0 = 0.5$	145
38	Recast Factuality, $p_0 = 0.5$	146
39	Recast Verbnet, $p_0 = 0.4034$	146
40	Recast Sentiment, $p_0 = 0.5$	147
41	Recast megaveridicality, $p_0 = 0.6667$	147
42	Boolq, $p_0 = 0.3769$	148
43	Wic, $p_0 = 0.5$	148
44	ADE Corpus V2 classification, $p_0 = 0.7113$	149
45	Hate, $p_0 = 0.5797$	149
46	Irony, $p_0 = 0.4951$	150
47	Offensive, $p_0 = 0.6693$	150
48	Rotten Tomatoes, $p_0 = 0.5$	151
49	Hover, $p_0 = 0.3934$	151
50	Movie Rationales, $p_0 = 0.5$	152
51	Eraser Multi RC, $p_0 = 0.5600$	152

# INTRODUCTION

Les réseaux de neurones artificiels ont connu un essor considérable ces dernières années, notamment grâce à l'avènement de nouvelles architectures, de nouvelles techniques d'optimisation, ainsi qu'à la disponibilité de grands volumes de données. Ces réseaux ont permis des avancées significatives dans de nombreux domaines, tels que la reconnaissance d'images, la compréhension du langage naturel, la prédiction de séries temporelles, et bien d'autres. Cependant, la conception de réseaux de neurones performants reste un défi important. Le choix d'une architecture appropriée dépend de nombreux facteurs, tels que la complexité du problème, la quantité et la qualité des données, la présence de bruit ou d'incertitudes, etc. De plus, la performance d'un réseau de neurones sur les données d'apprentissage ne garantit pas sa capacité à généraliser à de nouveaux exemples.

Pour répondre à ces défis, les réseaux de neurones grossissants ont été proposés comme une alternative intéressante aux réseaux classiques, dont l'architecture est fixe. En effet, lorsqu'on apprend un réseau de neurones artificiels, il est courant de fixer à l'avance l'architecture du réseau, c'est-à-dire le nombre de couches et le nombre de neurones dans chaque couche. Une fois que cette architecture est fixée, il est souvent nécessaire d'apprendre l'ensemble du réseau à partir de zéro si on souhaite ajouter des paramètres supplémentaires. Cependant, avec les réseaux de neurones grossissants, de nouveaux paramètres peuvent être progressivement ajoutés pendant l'apprentissage, sans avoir à réinitialiser complètement le réseau. En d'autres termes, la taille du réseau peut être augmentée au fil du temps en ajoutant de nouvelles couches ou en augmentant le nombre de neurones dans les couches existantes. En fixant les paramètres précédemment appris, les informations apprises précédemment sont conservées et les performances sur les tâches précédentes peuvent être conservées grâce à certaines heuristiques. Le résultat est un réseau qui peut continuer à apprendre de nouvelles tâches tout en conservant les connaissances acquises précédemment, ce qui peut conduire à des performances supérieures à celles des réseaux standard.

Dans cette thèse, nous proposons un modèle de réseau de neurones grossissant original, qui permet d'étudier le comportement de ces réseaux. Nous définissons ce modèle simple afin d'isoler au mieux l'impact du grossissement sur les résultats des réseaux obtenus pour l'étudier. Nous montrons théoriquement et expérimentalement que les réseaux grossissants convergent vers des minima plats. La planéité, étant une métrique correspondant à la régularité de la fonction de coût du réseau dans le voisinage de son minimum, est souvent liée aux performances de généralisation des réseaux de neurones et peut être assimilée au volume du minimum dans l'espace. Nous étudions également si les réseaux grossissants convergent effectivement, à architecture finale fixée, vers des meilleurs minima que leurs contreparties standards. Pour ce faire, nous développons une approximation non-supervisée du risque théorique pour la classification binaire, que nous utilisons pour optimiser des réseaux après leur apprentissage (*post-tuning*) et pour apporter une nouvelle mesure de généralisation à l'ensemble des estimateurs de performances.

En résumé, cette thèse aborde deux grandes questions de recherche, pourquoi le comportement des réseaux de neurones grossissants permet de concevoir des architectures performantes et comment évaluer efficacement la généralisation. En reliant ces deux as-

---

pects importants de l'apprentissage des réseaux de neurones artificiels, notre étude apporte un éclairage nouveau sur les mécanismes sous-jacents à la généralisation des réseaux de neurones, en les reliant à d'autres perspectives assez peu communes. Ces résultats peuvent ainsi avoir des implications pour la conception et l'optimisation des réseaux de neurones artificiels, en offrant des clés de compréhension nouvelles.

# **Première partie**

## **Étude de la taille des réseaux de neurones**



---

# CHAPITRE 1 : ÉTUDE DE L'ÉTAT DE L'ART : TROUVER UNE ARCHITECTURE OPTIMALE

*La volonté trouve, la liberté choisit.  
Trouver et choisir, c'est penser*

---

– Victor Hugo

---

1.1	Revue historique des architectures de réseaux de neurones . . . .	4
1.2	Trouver automatiquement une architecture optimisée : le cas du Neural Architecture Search (NAS) . . . . .	19
1.3	Les réseaux de neurones à architecture dynamique . . . . .	24

---

Dans ce premier chapitre, une étude approfondie de l'état de l'art sur les diverses architectures des réseaux de neurones artificiels (*Artificial Neural Networks* ou *ANNs*) sera menée. Bien qu'historiquement ces architectures ont été créées conçues "à la main", la recherche de l'architecture optimale pour un problème donné est de plus en plus automatisée, donnant naissance à un sous-domaine particulier : la recherche d'architecture neuronale ou *Neural Architecture Search (NAS)*.

Dans la suite, le NAS sera d'abord présenté comme un sous-domaine vaste, composé de 3 champs de recherche principaux. Cette présentation générale permettra de définir les éléments clés utilisés dans la suite de notre travail. Cela permettra aussi de définir précisément le contexte de cette thèse, à savoir les architectures d'ANNs évoluant dynamiquement.

La dernière section de ce chapitre se concentrera donc sur ces réseaux particuliers et présentera les travaux clés de la littérature concernant l'expansion et le rétrécissement des ANN.

## 1.1 Revue historique des architectures de réseaux de neurones

Dans cette première partie, un état des lieux historique des diverses architectures de réseaux de neurones artificiels est dressé. Les briques élémentaires pour construire un ANN sont présentées et des architectures marquantes sont rappelées. À travers le panorama qui suit, il est clair que définir l'architecture d'un ANN est une tâche complexe et chronophage, basée sur l'expérience et les nombreux essais des différents chercheurs. Cette première section s'adresse principalement au lecteur novice puisqu'elle constitue un rappel des modèles emblématiques de réseaux de neurones artificiels, notamment pour le Traitement Automatique du Langage Naturel (TALN ou *NLP* en anglais), qui est le domaine principal d'application des travaux de cette thèse.

### 1.1.1 Des premiers réseaux de neurones artificiels à l'apprentissage profond

#### Le perceptron (neurone artificiel)

**Historique et rappels** Les premiers réseaux de neurones artificiels émergent dans les années 1950 suite à la première modélisation du neurone biologique proposé par McCULLOCH et PITTS [135]. Ainsi, un modèle de neurone artificiel permettant de résoudre des problèmes de classification supervisée à partir d'images ou de sons est proposé dès 1957 par ROSENBLATT [162].

Il s'agit dès lors de rappeler précisément les définitions des types originaux d'apprentissage, ainsi que les principales familles de problèmes d'apprentissage communément rencontrés.

- L'apprentissage supervisé (*supervised learning*) s'intéresse aux données étiquetées et son objectif est de prédire l'étiquette (inconnue)  $y$  associée à une nouvelle observation  $\mathbf{x}$  à partir de la connaissance issue des  $N$  observations étiquetées du jeu de données  $(x_n, y_n)_{1 \leq n \leq N}$ .

Deux grandes familles de problèmes d'apprentissage supervisé peuvent être distinguées : la classification supervisée où  $y$  désigne une classe et la régression où  $y$  est une valeur scalaire ou vectorielle.

- À l'inverse, l'apprentissage non-supervisé (*unsupervised learning*) traite des données non-étiquetées. L'objectif étant d'identifier automatiquement des caractéristiques communes aux observations, pour par exemple identifier des groupes (ou clusters) d'observations partageant des similarités (on parle alors de classification non-supervisée ou de partitionnement). Il s'agit alors d'identifier automatiquement des structures sous-jacentes aux données. Il faut donc a priori disposer d'une distance  $D$  entre observations, mais également déterminer un seuil sur celle-ci afin de décider de la limite pour appartenir à un groupe ou alors déterminer le nombre de groupes à identifier dans l'ensemble des observations. D'autres tâches d'apprentissage non-supervisé existent mais ne sont pas traitées dans la suite du travail (estimation de densités de probabilités, réduction de dimension).

Afin de résoudre les tâches de classification supervisée, le premier modèle de neurone artificiel, le perceptron de ROSENBLATT [162] peut être présenté comme un classifieur binaire. Il admet  $d$  valeurs scalaires en entrée ( $x^1, x^2, \dots, x^d$ ) et retourne en sortie 1 ou  $-1$  selon le signe de  $w_0 + w_1x^1 + \dots + w_dx^d$ , où  $w_0, \dots, w_d$  sont les paramètres du modèle, comme illustré Figure 1.1.

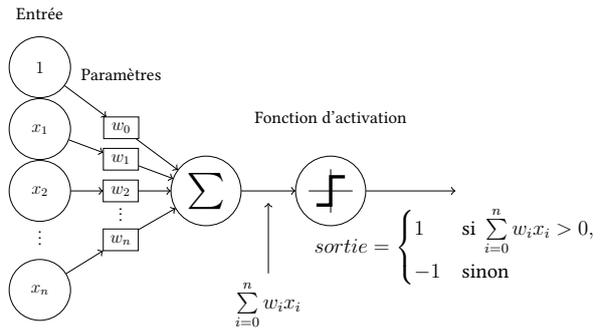


FIGURE 1.1 – Illustration du Perceptron, d'après ROSENBLATT [162]

Il calcule donc de quel côté d'un hyperplan affine se situe une observation  $\mathbf{x} = (x^1, \dots, x^d)$  et agit donc linéairement. Les  $d + 1$  paramètres  $w_i$  (appelés plutôt *poids*) sont adaptés selon les exemples de la base d'apprentissage. Ce perceptron évoque un modèle assez simpliste de neurone biologique et se contente simplement de mettre en oeuvre un classifieur linéaire, car la surface de séparation qu'il détermine est un hyperplan de l'espace des observations. L'algorithme d'apprentissage du perceptron, permettant d'adapter les poids en fonction de la base d'exemples d'apprentissage consiste à chaque étape à comparer la "vraie étiquette"  $y_n$  et l'étiquette prédite  $\hat{y}_n$  par le perceptron sur une observation  $x_n$ , cette étiquette étant calculée à l'aide de l'estimation courante des poids  $w$ . Si l'étiquette prédite ne correspond pas à la réalité, les poids du perceptron sont mis à jour et l'algorithme est donc *error-driven* (chaque mise à jour vise à corriger les erreurs) et *online* (les exemples sont présentés successivement).

D'après NOVIKOFF [145], si les observations étiquetées sont linéairement séparables, alors l'algorithme du perceptron aboutit en un nombre fini d'étapes aux paramètres d'un hyperplan séparateur et que la convergence est plus rapide si la marge entre les deux classes à discriminer est grande. Si les données ne sont pas séparables linéairement, l'algorithme ne converge pas et il faut donc arrêter après un certain nombre de parcours de données (appelé *epoch*). Il faut également noter que l'hyperplan séparateur fourni est sélectionné selon l'ordre de parcours des observations et peut être donc différent d'une exécution à l'autre. Cette indétermination peut poser problème pour la prédiction de la classe de nouvelles observations. Le perceptron peut également être étendu au cas de  $K > 2$  classes avec des étiquettes  $y \in \{1, 2, \dots, K\}$ .

Les classes sont associées à  $K$  vecteurs-poids  $w_1, w_2, \dots, w_K$  (un par classe) et on associe chaque observation  $x$  à l'étiquette  $\hat{y} = \arg \max_k w_k \cdot x$ . L'apprentissage est effectué similairement au cas bi-classe.

**Limites naturelles du perceptron** Dès lors qu'il s'agit de résoudre un problème de classification non linéaire, le perceptron est confronté à ses limites naturelles. Ainsi, la Figure 1.2 illustre un cas pourtant simple face auquel le perceptron échoue : la fonction XOR étant non séparable linéairement,

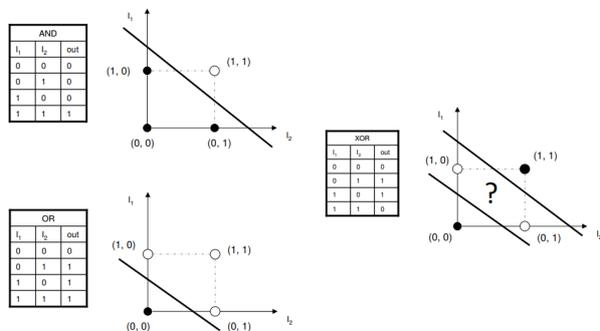


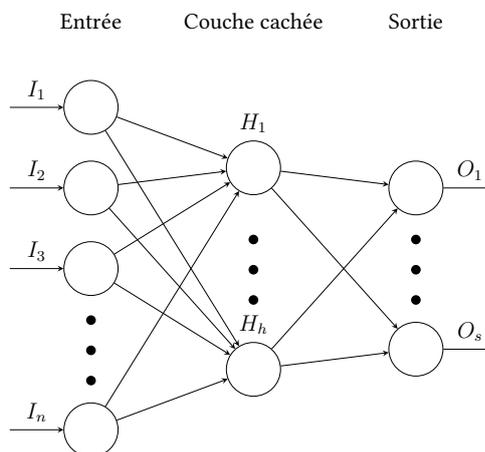
FIGURE 1.2 – Calculs logiques et limites du perceptron (adapté du cours "Introduction à l'Apprentissage Automatique", F. Sur – Mines Nancy)

il est impossible pour un hyperplan seul de résoudre le problème. Une solution naturelle pour résoudre ce problème serait d'empiler les perceptrons, seulement la règle originale d'apprentissage du perceptron ne se généralisait que mal à un réseau à plusieurs couches, rendant l'apprentissage de tels systèmes très coûteux et complexe. Ce problème fût résolu par RUMELHART, HINTON et WILLIAMS [165] à l'aide de l'algorithme de rétro-propagation des erreurs. Les auteurs proposèrent l'usage d'une fonction différentiable plutôt que la fonction échelon comme fonction d'activation du perceptron. Grâce à cette modification, un réseau multicouche devient différentiable et une descente de gradient peut donc être appliquée pour minimiser l'erreur du réseau et grâce à la chain-rule, l'erreur pourrait être rétro-propagée et dérivée et les poids de chaque couche du réseau mis à jour, les couches intermédiaires étant appelées "couches cachées".

Un perceptron multicouche (ou MLP, pour *multilayer perceptron*) est un réseau possédant plusieurs couches de neurones dont les fonctions d'activation sont des fonctions différentiables. Un tel réseau ne présente pas de boucle ou cycle, l'information se propage de l'entrée à la sortie : ce sont des réseaux à propagation avant (ou FFNN pour *feedforward neural network*). Toute structure quelconque d'un tel réseau (peu importe sa taille ou ses détails structurels tels que des *skip-connexions*<sup>1</sup>) utilisé pour une classification à  $K$  classes peut être considéré de manière topologique comme un graphe direct acyclique (ou DAG pour *directed acyclic graph*). HEALY et NIKOLOV [78] et NEYSHABUR, TOMIOKA et SREBRO [141] ont montré que tout DAG possède au moins un ordonnancement topologique, qui peut être utilisé pour créer une structure en couches, ce qui assure la correspondance entre FFNN et DAG qui sera développé dans la Section 2.1.1, chaque couche correspondant à un "niveau" dans le vocabulaire de la théorie des graphes.

Un perceptron multicouche traite des vecteurs de  $\mathbb{R}^d$ , la couche d'entrée étant composée de  $d$  neurones lisant les composantes des vecteurs et envoyant l'information aux neurones de la première couche cachée. Chaque neurone d'une couche cachée fait une moyenne pondérée des informations reçues de la couche précédente et réémet aux neurones de la couche suivante cette information moyenne modifiée par une fonction d'activation différentiable. Cette structure est illustrée dans la Figure 1.3

On peut ainsi développer un modèle contenant plusieurs couches cachées se succédant,



**FIGURE 1.3** – Illustration d'un MLP à une couche cachée. L'information en entrée est ici un vecteur de  $\mathbb{R}^n$  et en sortie un vecteur de  $\mathbb{R}^s$ . Avec  $n$  neurones en entrée,  $h$  dans la couche cachée et  $s$  neurones en sortie, le nombre de paramètres (poids et biais) du modèle ainsi présenté est  $(n + 1) \cdot h + (h + 1) \cdot s$  (chaque neurone introduit un paramètre supplémentaire, appelé biais, d'où le "+1").

1. skip-connexion : type de connexion au sein d'un réseau neuronal qui permet de relier directement la sortie d'une couche à une couche ultérieure, en "sautant" une ou plusieurs couches intermédiaires.

chacune étant constituée d'un certain nombre de neurones connectés aux neurones de la couche précédente. L'information arrivant à un neurone  $j$  de la couche  $k$  (ou activation d'entrée) est :

$$a_j = \sum_i w_{ji} z_i + b_j, \text{ où}$$

- la somme porte sur l'ensemble des neurones  $i$  de la couche précédente  $k - 1$  connectés avec le neurone  $j$ ,
- $w_{ji}$  est le poids de la connexion entre le  $i$ -ème neurone de la couche  $k - 1$  et le neurone  $j$ ,
- $b_j$  (appelé "biais") est un scalaire,
- $z_i$  est le signal émis par ce  $i$ -ème neurone.

Le neurone  $j$  émet ensuite l'information  $\sigma(a_j)$  aux neurones de la couche suivante  $k + 1$  auxquels il est connecté, où  $\sigma$  est une fonction d'activation différentiable. La dernière couche émet enfin le vecteur ou le scalaire de sortie du réseau. La nature de la sortie dépend du problème traité (classification ou régression). L'apprentissage du modèle consiste à optimiser les paramètres du réseau (poids et biais) afin de prédire correctement les étiquettes d'une base d'exemple en minimisant les erreurs produites sur celle-ci. Nous généralisons ces notations dans la Section 2.1.1, pour décrire le plus généralement possible un FFNN quelconque. Cependant, il est bon de rappeler ici les définitions essentielles suivantes :

- **Fonction de coût** : Une fonction de coût  $\mathcal{L}(\theta)$  est une mesure de l'erreur d'un modèle par rapport à ses prédictions  $\hat{y}$  par rapport aux valeurs réelles  $y$ . Elle est généralement définie comme une fonction des paramètres du modèle  $\theta$  et des données d'apprentissage.
- **Gradient** : Le gradient d'une fonction  $f(x)$  par rapport à un vecteur  $x$  est un vecteur contenant les dérivées partielles de  $f$  par rapport à chaque composante de  $x$ . Il est noté  $\nabla f(x)$ .
- **Hessienne** : La hessienne d'une fonction  $f(x)$  est une matrice des dérivées partielles secondes de  $f$  par rapport aux composantes de  $x$ . Elle est notée  $\nabla^2 f(x)$ .

**Fonctions d'activation** Comme mentionné précédemment, le signal en sortie d'un neurone  $j$  est :

$$z_j = \sigma(a_j) = \sigma(\sum_i w_{ij} z_i + b_j).$$

Si les diverses fonctions d'activation  $\sigma_k$  des différentes couches d'un réseau étaient toutes linéaires, le modèle ne ferait que des compositions de fonctions linéaires des entrées et émettrait donc en sortie une simple combinaison linéaire des entrées, ce qui limiterait son intérêt. C'est pourquoi il est fondamental de considérer des fonctions d'activation non linéaire. Une activation en échelon comme pour le perceptron de ROSENBLATT [162] restreint les  $z_j$  aux seules valeurs  $-1$  ou  $1$ . De plus une telle fonction pose, comme vu précédemment, le problème de non dérivabilité pour le cas

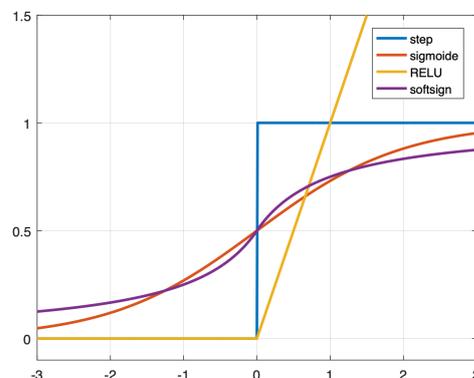


FIGURE 1.4 – Quelques fonctions d'activation.

multi-couche. D'autres activations ont donc été proposées, toutes continues, croissantes et (presque partout) dérivables. Le champ de recherche des fonctions d'activation est toujours actif, mais nous pouvons citer à titre d'exemples majeurs les fonctions suivantes :

- sigmoïde :  $\forall x \in \mathbb{R}, f(x) = \frac{1}{1+e^{-x}}$
- softsign :  $\forall x \in \mathbb{R}, f(x) = \frac{x}{1+|x|}$
- ReLU (pour *Rectified Linear Unit*) :  $\forall x \in \mathbb{R}, f(x) = \max\{0, x\}$

On illustre Figure 1.4 le graphe des quelques fonctions d'activation mentionnées ici.

**Expressivité des FFNN** Il a été démontré dès 1989 par CYBENKO [36] que si  $\sigma$  était continue et sigmoïdale (ie strictement croissante et  $\lim_{-\infty}\sigma = 0, \lim_{\infty}\sigma = 1$ ), toute fonction réelle continue sur un compact<sup>2</sup> peut être approchée d'aussi près que l'on veut par la sortie d'un perceptron à une couche cachée possédant suffisamment de neurones.

Ce résultat a ensuite été étendu, notamment par HORNIK [86], avec  $\sigma$  continue, bornée et non constante.

Le résultat prouvé par BARRON [10] majore ensuite le nombre  $N$  de neurones d'un réseau à une couche cachée pour approcher une fonction  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  admettant une représentation de Fourier de la forme

$$f(x) = \int_{\mathbb{R}^d} \widehat{f}(w) e^{i\langle w, x \rangle} dw$$

avec  $\widehat{f} : \mathbb{R}^d \rightarrow \mathbb{C}$ , telle que  $C_f := \int_{\mathbb{R}^d} \|w\|_2 |\widehat{f}(w)| dw$  est finie.

LESHNO et al. [121] améliorent les hypothèses faites sur  $\sigma$ , en effet leur travail montre que l'ensemble des FFNN à une couche cachée permet d'approcher toute fonction continue sur tout compact (en norme  $L^\infty$ ) si et seulement si  $\sigma$  est non-polynomiale.

Plus récemment enfin, ce problème a été étudié par KIDGER et LYONS [108] avec une profondeur quelconque mais en limitant la largeur. Ainsi, ils montrent que si  $\sigma$  est continue, non-affine, et de classe  $C^1$  au voisinage d'un point  $t_0$  avec  $\sigma'(t_0) \neq 0$ , alors, pour tout compact  $K \subset \mathbb{R}^d$ , l'ensemble FFNN de largeur  $d + 3$  et de profondeur quelconque est dense dans  $(C(K; \mathbb{R}), \|\cdot\|_\infty)$ .

Les FFNN à une couche cachée ont donc une expressivité suffisante pour représenter correctement toutes les frontières de classification, seulement d'après les résultats de ELKAN et SHAMIR [49] et DANIELY [39], à moins de considérer une couche cachée de taille exponentielle en la dimension d'entrée  $d$ , il existe des fonctions non approchables par un réseau à 1 couche cachée, mais aisément représentables par un réseau à 2 couches ou plus. En pratique, les réseaux utilisés aujourd'hui ont plusieurs couches (parfois une dizaine) et font partie des réseaux profonds (*Deep Neural Networks* ou DNN).

---

2. **Rappel** : Un ensemble  $K$  d'un espace topologique  $X$  est dit *compact* si pour tout recouvrement ouvert de  $K$  ( $\{U_\alpha\}_{\alpha \in A}$  telle que  $K \subseteq \bigcup_{\alpha \in A} U_\alpha$ ), il existe un sous-recouvrement fini de  $K$  ( $\{U_1, U_2, \dots, U_n\}$  telle que  $K \subseteq \bigcup_{i=1}^n U_i$ ).



latente est conditionnée par la largeur de la couche finale de l'encodeur qui constitue un goulot d'étranglement pour les informations (*information bottleneck, IB*), c'est-à-dire que seulement un nombre fini d'informations peut être transmis par l'encodeur (nombre limité par la largeur du goulot d'étranglement) Une fois encodées, les données sont décodées, c'est-à-dire que le réseau va essayer de construire une sortie aussi proche que possible de l'entrée à l'aide de la représentation latente de l'entrée.

Cette architecture peut être considérée comme une brique élémentaire car sa structure est utilisée comme une structure de base pour représenter des données (peu importe ses composants internes). On peut ainsi extraire uniquement la partie encodeur de l'auto-encodeur après apprentissage pour construire un réseau à sa suite, réseau prenant donc en entrée la représentation latente des données apprises par l'encodeur.

Initialement indirectement introduits par COTTRELL et al. [35] pour compresser et reconstruire des données d'image, les Auto-Encodeurs étaient initialement une architecture particulière de FFNN illustrée Figure 1.6. BALLARD [8] propose d'utiliser une telle architecture comme une méthode pour pré-apprendre de manière non supervisée des ANNs. L'architecture des auto-encodeurs fait donc partie de l'histoire des réseaux de neurones artificiels (voir par exemple les travaux de LE CUN et FOGELMAN-SOULIÉ [115], BOURLARD et KAMP [16], ZEMEL et HINTON [212]) et étaient utilisés traditionnellement pour réduire la dimension d'entrée ou effectuer du *feature learning*.

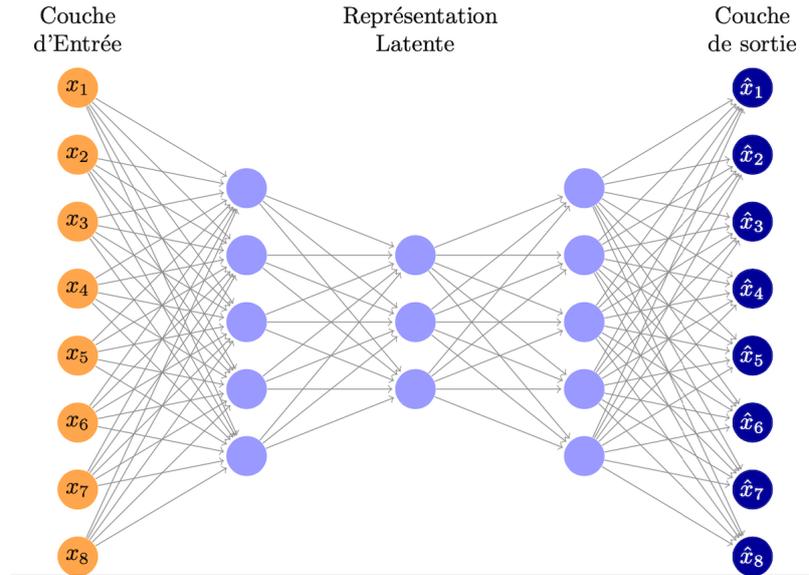
De nombreuses versions perfectionnées d'auto-encodeurs ont depuis vu le jour comme nous le verrons dans le paragraphe dédié aux Transformers. Parmi celles-ci nous pouvons néanmoins mentionner :

- les auto-encodeurs variationnels (VAEs) introduits par KINGMA et WELLING [111], les VAEs sont un type probabiliste d'auto-encodeurs. En plus de mapper les données à une représentation compressée, le VAE tente de modéliser la distribution des données. Ils peuvent de plus générer de nouvelles données en décodant une gaussienne aléatoire.
- les auto-encodeurs de débruitage (DAEs), introduits comme tels par VINCENT et al. [193], les DAEs ajoutent du bruit aux données d'entrée et apprennent à le supprimer. Avec le bruit introduit sur les données d'entrée, l'encodeur extrait naturellement les caractéristiques majeures des données et apprend ainsi une représentation robuste des données.

## Réseaux convolutifs

La convolution est l'outil mathématique de base du traitement du signal (comme le traitement des images ou du son). Elle se base sur les propriétés naturelles de transformation du signal : la linéarité, la continuité et l'invariance par translation. Ces propriétés naturelles ont les définitions suivantes :

- linéarité : la réponse à deux signaux superposés est la superposition des réponses à chacun des signaux indépendamment
- continuité : une perturbation en entrée du signal induit une perturbation du signal en sortie



**FIGURE 1.6** – Architecture typique d'un AutoEncodeur. Dans cette illustration, une entrée est composée de 8 données que l'encodeur réduit à 3 dans sa représentation latente. Il décode ensuite l'information encodée dans la partie décodeur.

- invariance par translation : le signal est transformé de la même manière peu importe sa position (temporelle dans le cas de sons) ou physique (dans le cas d'images).

Pour tout filtre de signal  $H$  vérifiant ces trois propriétés il existe une distribution  $h$  telle que pour tout signal en entrée  $e$  :  $H(e) = h * s$ , où  $*$  désigne le produit de convolution tel que : si  $h$  et  $s$  sont deux fonctions réelles intégrables,

$$\forall t \in \mathbb{R}, h * s(t) = \int_{\mathbb{R}} h(t - u)s(u)du = \int_{\mathbb{R}} h(u)s(t - u)du.$$

Ainsi tout filtre peut s'exprimer comme un produit de convolution et est défini par son noyau de convolution  $h$ . On peut définir une convolution discrète en remplaçant naturellement l'intégrale sur  $\mathbb{R}$  par une somme sur  $\mathbb{Z}$  et ainsi s'intéresser aux signaux discrets. Dans le cas d'une image par exemple, la convolution peut s'écrire sous forme d'une somme double :

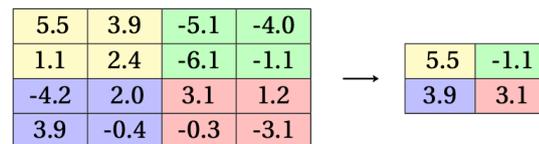
$$\forall (m, n) \in \mathbb{Z}^2, h * s(m, n) = \sum_{(k, l) \in \mathbb{Z}^2} h(k, l)s(m - k, n - l).$$

Cette opération étant linéaire, elle est réalisable par un réseau de neurones, dont les neurones d'entrée lisent le signal et le transmettent à la couche cachée, faite du même nombre de neurones que la couche d'entrée et qui calcule les  $h * s$ . Pour limiter le nombre de coefficients à estimer et donc le temps de calcul, on considère un support de noyau de convolution de taille réduite. Ainsi en traitement des images, chaque neurone de la première couche n'est connecté qu'à une sous-partie de l'image initiale typiquement de taille 3x3 ou 5x5 par exemple (ou 3x3x3 ou 5x5x3 pour des images à 3 canaux de couleurs). Chaque neurone de ce type fait donc une combinaison linéaire locale des valeurs de pixels auxquels

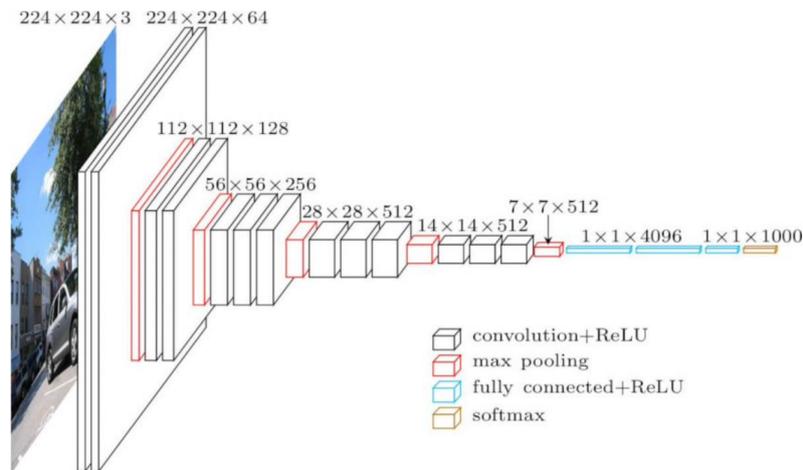
il est connecté. Puisqu'on a de plus aucune information a priori sur l'image, les poids de tous ces neurones doivent être les mêmes, ce qui traduit l'invariance par translation. Ainsi, on a une réduction considérable de la complexité : chaque neurone s'occupant parfois de plusieurs dizaines d'entrées (selon la taille de la sous-partie de l'image attribuée) et la matrice des poids étant la même pour tous les neurones. Les extrémités du signal (les bords dans le cas d'une image par exemple) sont par ailleurs traitées de manière spécifique, par exemple en fixant l'extérieur du signal à une valeur nulle.

En sortie de cette couche de convolution, une fonction d'activation rend le signal non linéaire. Plusieurs couches de convolution peuvent être empilées afin de complexifier le réseau. Ainsi, si une première couche est composée de  $k$  filtres, les noyaux de la seconde couche agiront sur ces  $k$  sorties de la première couche. En pratique, les fonctions d'activation les plus utilisées sont les fonctions non bornées comme ReLU afin d'éviter un "vanishing gradient" (valeur du gradient qui tend vers 0) lors de la composition par plusieurs sigmoïdes. Afin d'intégrer à l'entrée traitée

par un neurone des informations plus distantes que la taille du noyau, il est possible de faire opérer un noyau sur couche sous-échantillonnée (par exemple par un facteur 2) pour avoir un effet semblable à une augmentation de la taille du noyau de support (de taille double dans cet exemple). Il s'agit de mise en commun (ou pooling) et plusieurs formes de pooling existent (max-pooling : on prend la valeur maximale, mean pooling : on fait la moyenne des valeurs, etc.). Cette technique est illustrée Figure 1.7. À la fin d'un tel réseau enfin, on a en général un FFNN qui donne l'estimation  $\hat{y}$  adaptée à la tâche considérée à partir de la représentation des données qu'on notera  $\Phi(x)$  apprise par le réseau. Une couche de convolution peut donc être vue comme une couche de FFNN classique à laquelle on fixe des contraintes supplémentaires sur les poids. L'apprentissage permet ici d'adapter les filtres obtenus en optimisant leurs paramètres. Le premier usage de ConvNets pour traiter des images après apprentissage par rétro-propagation du gradient (comme un MLP) date historiquement du travail de LECUN et al. [117]. LECUN et al. [118] expliquent le principe d'apprendre des DNN en utilisant des techniques d'optimisation basées sur le gradient et montrent que ces réseaux peuvent être effectivement combinés avec des mécanismes inférentiels pour modéliser des sorties interdépendantes. Il faut cependant attendre les années 2010, apportant une disponibilité de grandes bases d'apprentissage et un matériel de calcul performant pour que ces réseaux émergent et dominent leurs concurrents en traitement du signal. Deux réseaux historiques utilisant des convolutions sont AlexNet (2012) que nous avons déjà mentionné Figure 1.5 et VGG16 introduit par SIMONYAN et ZISSERMAN [174] représenté Figure 1.8.



**FIGURE 1.7** – Exemple de max-pooling de facteur 2 : chaque bloc de dimension 2x2 est remplacé par une unique valeur égale au maximum des valeurs du bloc considéré. On met donc en relation des neurones présents dans un "patch" de données deux fois plus étendu (adapté du cours "Introduction à l'Apprentissage Automatique", F. Sur – Mines Nancy)



**FIGURE 1.8** – Architecture de VGG16, d’après SIMONYAN et ZISSERMAN [174]. L’entrée est constituée de  $224 \times 224 \times 3 = 150.528$  neurones, ce qui correspond à une image couleur (trois canaux : (R,G,B)) de  $224 \times 224$  pixels. Les deux premiers blocs de convolutions sont constitués de 64 filtres indépendants de support de taille  $3 \times 3 \times 3$ , chacun étant suivi d’une activation ReLU. Un pixel sur deux est ensuite gardé dans le max pooling. Les blocs de convolution et de pooling se succèdent jusqu’à atteindre FFNN classique constitué de deux couches de 4096 neurones, une couche de 1000 neurones, puis une couche SoftMax de 1000 neurones pour adapter la représentation des données apprises  $\Phi(x)$  à la tâche de classification à 100 classes. Ce réseau possède 138 millions de paramètres.

## Réseaux récurrents

Pour des données d’entrée de taille variable, en particulier pour l’analyse de séries temporelles des réseaux avec une notion de mémoire ont également été développés. Ainsi, les réseaux de neurones récurrents (ou *recurrent neural networks*, RNN) présentant des connexions récurrentes ont vu le jour suite aux travaux de RUMELHART, HINTON et WILLIAMS [165] en 1986. Cette classe de réseaux de neurones permet aux connexions entre neurones de créer des cycles, permettant ainsi à la sortie de neurones d’avoir un impact en entrée de ces mêmes neurones, permettant ainsi une forme de comportement temporel dynamique, ou de mémoire.

Ces réseaux ont donc été utilisés dans des domaines où la temporalité ou continuité des données joue un rôle, comme l’analyse de séries temporelles, par exemple par HEWAMALAGE, BERGMEIR et BANDARA [80], ou encore la reconnaissance de la parole (*speech recognition*), notamment dans les travaux de GRAVES, MOHAMED et HINTON [68]. Les RNN reposent sur deux principes : des connexions récurrentes permettant d’avoir des informations du passé du signal d’entrée (en ré-injectant la sortie d’une couche en entrée de celle-ci) et le traitement de signaux d’entrée de taille variable en le traitant à l’aide d’une "fenêtre glissante".

Les réseaux les plus connus de ce type sont les LSTM (*Long short-term memory*), modélisés par HOCHREITER et SCHMIDHUBER [83] qui ont été appliqués dès 2007 pour révolutionner de nombreux domaines, de la reconnaissance de la parole, à la reconnaissance de texte manuscrit ou encore la traduction automatique (voir les travaux de FERNÁNDEZ, GRAVES et

SCHMIDHUBER [55], GRAVES et SCHMIDHUBER [71], ou encore SUTSKEVER, VINYALS et LE [181] pour plus de détails sur ces applications).

Le principe des LSTM se décompose en 3 portes, comme l'illustre la Figure 1.9.

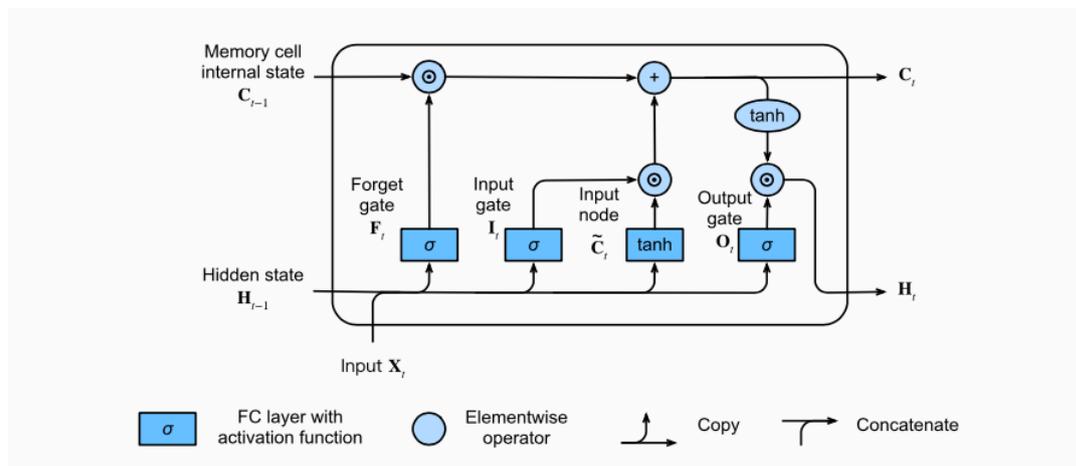


FIGURE 1.9 – Architecture d'une couche de LSTM, adapté de : [d2l](#).

Le but pour un LSTM est de retenir les informations utiles le plus longtemps possible pour pallier au problème des réseaux récurrents initiaux qui remplaçaient l'information la plus ancienne par des nouvelles données dès lors que la mémoire du réseau était pleine et ainsi d'avoir une mémoire à long terme en plus d'une mémoire à court terme. Pour ceci, le LSTM possède une mémoire interne (*cell*) qui permet de maintenir un état aussi longtemps que nécessaire. Cette cellule mémoire est pilotée par trois portes qui sont en fait des sélecteurs d'information dont la valeur en sortie est entre 0 et 1 (grâce à une sigmoïde) :

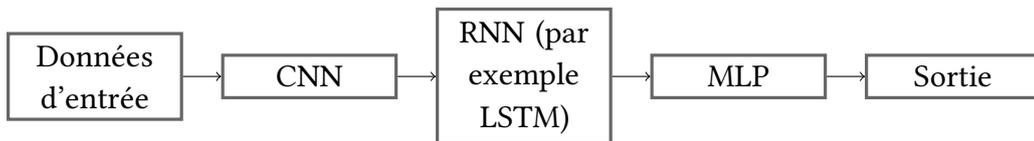
- *input gate* ou porte d'entrée qui indique en sortie si l'entrée doit modifier le contenu de la mémoire, c'est-à-dire si l'on doit insérer de nouvelles informations en mémoire.
- *forget gate* ou porte d'oubli qui indique en sortie s'il faut effacer la mémoire (et donc la remettre à 0), c'est-à-dire si l'on doit supprimer de l'information en mémoire
- *output gate* ou porte de sortie qui indique si le contenu de la cellule influe sur la sortie du LSTM, c'est-à-dire si l'on doit utiliser de l'information présente en mémoire

Cette architecture a été déclinée en de nombreux modèles, comme les LSTM bi-directionnels permettant de traiter de l'information future (lorsqu'elle est présente) dans les travaux de GRAVES et SCHMIDHUBER [70] qui a permis d'atteindre des performances exceptionnelles, notamment en reconnaissance de la parole et Traitement Automatique du Langage (TAL), notamment dans les publications de GRAVES, JAITLY et MOHAMED [69], CHIU et NICHOLS [30] et HUANG, XU et YU [89].

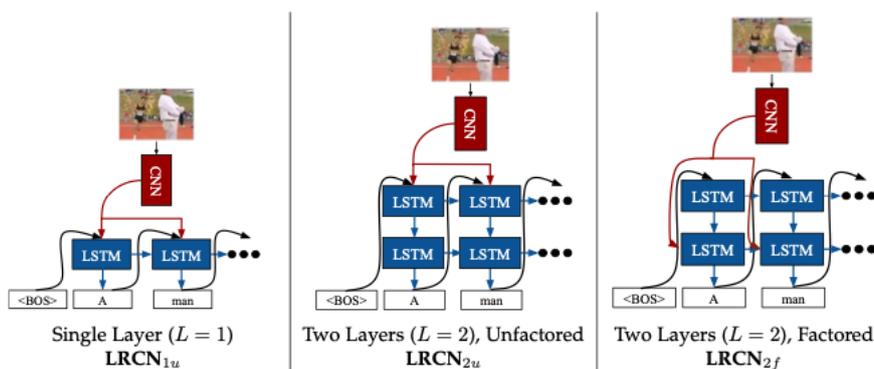
De nombreux autres types de réseaux récurrents ont également été modélisés comme par exemple les *Gated Recurrent Unit* (GRU) qui ont des architectures plus simples que les LSTM et peuvent être utilisés également pour modéliser des problématiques court et long terme, introduits par DEY et SALEM [42].

Ces RNN ont souvent été combinés avec des CNN et des MLP pour créer des architectures profondes complexes, à l'image du modèle LRCN proposé par DONAHUE et al. [45] combinant

les avantages des LSTM et des CNN pour simultanément apprendre des dynamiques temporelles et des représentations convolutives. Différentes variantes de ce modèle sont illustrées Figure 1.11. De nombreuses autres architectures combinant ces briques élémentaires ont vu le jour, se basant la plupart du temps sur un CNN comme extracteur de features qui est ensuite lié à un LSTM qui permet au modèle d'avoir une mémoire qui à son tour est lié à un MLP qui projette le modèle en sortie. On peut donc résumer la plupart de l'architecture de ce type de modèle par la Figure 1.10.



**FIGURE 1.10** – Architecture résumée de la plupart des combinaisons des briques élémentaires présentées. Les CNN sont utilisés pour extraire les features des données d'entrée. Les RNN permettent d'extraire des liens temporels. Les MLP permettent une projection vers les sorties.



**FIGURE 1.11** – Différentes variantes de LRCN modèle proposé par DONAHUE et al. [45]

Ce type de réseaux s'est naturellement imposé comme l'approche ayant les meilleurs résultats dans la modélisation de séquences, en particulier pour modéliser le langage naturel et pour la traduction automatique depuis le début de leur utilisation avec une structure d'auto-encodeur (par exemple dans les travaux de BAHDANAU, CHO et BENGIO [6], CHO et al. [31], ou encore SUTSKEVER, VINYALS et LE [181]); jusqu'à des modèles extrêmement perfectionnés, proposés par exemple par JOZEFOWICZ et al. [97], LUONG, PHAM et MANNING [132], ou encore WU et al. [205].

En alignant les positions sur des pas de temps de calcul, les réseaux récurrents génèrent une séquence d' "états cachés" (*hidden states*)  $h_t$  comme une fonction de l'état caché précédent  $h_{t-1}$  et de l'entrée à la position  $t$ . Cependant, cette nature séquentielle empêche la parallélisation dans les exemples d'apprentissage ce qui est un problème majeur pour les séquences de longueur élevée puisque la mémoire limite l'usage de lots (ou *batches*) d'exemples. Même si des travaux ont permis des améliorations significatives en termes computationnels via des *factorization tricks* introduits par KUCHAIEV et GINSBURG [113] ou des calculs conditionnels proposés par SHAZEER et al. [169], la contrainte relative au calcul séquentiel elle reste problématique.

Afin de modéliser des dépendances sans se soucier de la distance à la séquence d'entrée ou de sortie, certains modèles ont également combiné les réseaux récurrents à des mécanismes d'attention, notamment proposés par KIM et al. [110] et BAHDANAU, CHO et BENGIO [6]. Peu de temps après ces travaux, un article fondateur du traitement du Deep Learning contemporain par VASWANI et al. [191] utilise ces mécanismes d'attention pour modéliser le premier Transformer, modèle rendant pratiquement obsolètes les modèles vus précédemment.

## L'hégémonie des Transformers : les mécanismes d'attention

Comme vu précédemment, un FFNN standard constitue une série de couches de transformation non linéaires, chacune produisant une représentation cachée des données d'entrée de dimension fixe. Pour des tâches où les entrées sont de grande taille (typiquement des séquences de texte), ce paradigme rend l'interaction entre composantes difficile à contrôler. Si nous prenons l'exemple de la traduction automatique, l'entrée constitue une phrase entière et la sortie une prédiction pour chaque mot dans la phrase traduite. En utilisant un FFNN, un Information Bottleneck se forme puisqu'une couche cachée doit encoder une phrase entière. Les mécanismes d'attention permettent une approche alternative, à savoir qu'un réseau d'attention permet à l'ensemble des représentations cachées d'avoir une échelle comparable à la taille de l'entrée. Ce type de modèles utilise une étape d'inférence interne pour effectuer une "sélection douce" (*soft selection*) de ces représentations cachées. Cette méthode permet ainsi au modèle de garder une mémoire de la taille des entrées.

Décrivons formellement ce mécanisme en reprenant les notations de KIM et al. [110] : Soit  $X = [x_1, \dots, x_n]$  une séquence d'entrée, soit  $q$  une requête et  $z$  une variable catégorique latente ayant pour espace d'échantillonnage  $\{1, \dots, n\}$  qui encode la sélection désirée parmi les entrées. Le but ici est de produire un contexte  $c$  basé sur la séquence et la requête. Pour ce faire, l'accès à une *distribution d'attention*  $z \sim p(z|X, q)$ , où  $p$  est conditionnée sur les entrées et la requête, est admis. Le *contexte* sur une séquence est donc définie comme l'espérance :  $c = \mathbb{E}_{z \sim p(z|X, q)}[f(X, z)]$ , où  $f(X, z)$  est une *fonction d'annotation*, c'est-à-dire une fonction qui associe des métadonnées ou des informations auxiliaires à un objet ou à une donnée brute. Ce genre d'attention peut être appliqué à tout type d'entrées, mais dans le cas des DNN, la fonction d'annotation et la distribution d'attention sont toutes deux paramétrées par des réseaux de neurones et le contexte produit est ainsi un vecteur, qui sert d'entrée à la suite du réseau. En résumé, le mécanisme d'attention peut être interprété comme l'espérance d'une fonction d'annotation  $f(X, z)$  par rapport à une variable latente  $z \sim p$ , où  $p$  est définie comme une fonction de  $X$  et  $q$ . Ainsi, pour se rapprocher du formalisme contemporain, un mécanisme d'attention peut être décrit comme mappant une requête et un ensemble de paires (clé, valeur) à une sortie, où toutes les composantes de la fonction sont des vecteurs. La sortie est calculée comme la somme pondérée des valeurs où le poids associé à chaque valeur est calculé via une fonction calculant la compatibilité de la requête avec la clé correspondante.

L'auto attention (*self-attention*) est un mécanisme d'attention particulier, reliant différentes positions d'une séquence unique pour calculer une représentation de ladite séquence. Ce mécanisme a été utilisé avec succès sur une variété de tâches de TALN, allant de la compréhension de lecture à l'apprentissage de représentations de phrases (voir notamment les travaux de CHENG, DONG et LAPATA [28], LIN et al. [125], PARIKH et al. [147] et PAULUS,

XIONG et SOCHER [149]).

Cependant, le Transformer est le premier modèle basé entièrement sur la *self-attention* pour calculer des représentations des entrées et des sorties, n'utilisant ainsi ni RNN ni convolution. Le calcul de l'attention se fait ainsi à l'aide de trois matrices : K, V et Q (pour *Key*, *Value*, *Query*, resp. clé, valeur, requête), calculées en multipliant l'entrée X par des matrices de poids apprises pendant l'apprentissage, chaque colonne faisant référence à un mot de la séquence. Ainsi, en notant  $Q = XW_q$  la matrice requête (mot courant),  $K = XW_k$  la matrice clé (index unique du vecteur valeur) et  $V = XW_v$  le vecteur valeur (information contenue dans le mot d'entrée), la self attention présentée par VASWANI et al. [191] (appelée *Scaled Dot-Product Attention*) est calculée ainsi :

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

où l'entrée consiste en des requêtes et clés Q et K de dimension  $d_k$  et de valeurs V de dimensions  $d_v$ .

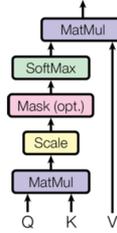
Le produit scalaire entre Q et K permet de calculer la similarité de la requête avec toutes les clés, puis normaliser chacune en divisant par  $\sqrt{d_k}$  avant d'appliquer un softmax pour obtenir le poids à accorder à chacune des valeurs de V.

Une des particularités de ce modèle réside dans son attention à multiples têtes (*multi-head attention*) qui permet au modèle d'avoir accès simultanément à de l'information provenant de différentes représentations en ayant plusieurs couches d'attention opérant en parallèle. La self-attention et la multi-head attention sont représentées Figure 1.12. Le principe est de séparer (horizontalement) les matrices Requête, Clé et Valeur en plusieurs matrices avant d'appliquer le Self-Attention aux sous-matrices obtenues. Il suffit ensuite de concaténer les sous-matrices résultantes pour obtenir le résultat. Ainsi, la multi-head attention se formalise :

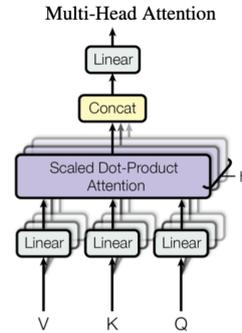
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O,$$

où  $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$  et les matrices de paramètres sont telles que :  $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$ ,  $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ ,  $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ , et  $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ . VASWANI et al. [191] utilisent  $h = 8$  têtes d'attention en parallèle et pour chacune d'entre elle,  $d_k = d_v = \frac{d_{\text{model}}}{h} = 64$ . Les auteurs précisant également que grâce à la dimension réduite de

Scaled Dot-Product Attention



(a) Auto-Attention simple



(b) Attention à Multiples têtes

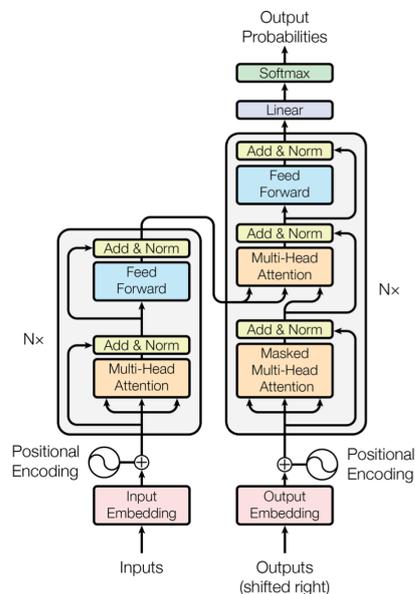
FIGURE 1.12 – Illustration de la self-attention et de la multi-head attention, figures tirées du travail de VASWANI et al. [191]

chaque tête, le coût de calcul total est comparable au cas où des têtes d'attention simples traitent l'entrée telle quelle.

Dans ce cas, l'encodeur mappe une séquence d'entrée de représentations symboliques  $(x_1, \dots, x_n)$  à une séquence de représentations continues  $z = (z_1, \dots, z_n)$ . Étant donné  $z$ , le décodeur génère en sortie une séquence de symboles  $(y_1, \dots, y_m)$  élément par élément. À chaque étape, le modèle, à la manière d'un RNN, est auto-régressif, c'est-à-dire qu'il utilise les symboles générés précédemment comme entrées supplémentaires à mesure qu'il génère la sortie. L'architecture résumée est illustrée Figure 1.13, avec respectivement à gauche l'encodeur et à droite le décodeur.

De nombreux travaux ont ensuite adapté et affiné cette architecture en TALN (par exemple, DEVLIN et al. [41], LIU et al. [129], HE et al. [77], SCAO et al. [167], etc.) que nous détaillerons en Section 3.2.2 du en effectuant un pré-apprentissage de l'auto-encodeur sur des masses de données de plus en plus massives, créant des modèles toujours plus volumineux ; mais cette architecture est également en train de révolutionner le domaine de la vision par ordinateur, voir notamment le travail de KHAN et al. [106] pour une review récente.

Ces variations d'architectures pour obtenir des modèles plus performants se font via un long et coûteux processus de conception de modèles, souvent effectué "à la main" au prix de nombreuses heures de calcul, sans assurance préalable sur la qualité du modèle proposé. C'est pour pallier cette problématique d'optimisation d'architecture que le domaine de la recherche d'architecture neuronale (*Neural Architecture Search*, NAS) tente de proposer des solutions pour trouver automatiquement des architectures optimisées.



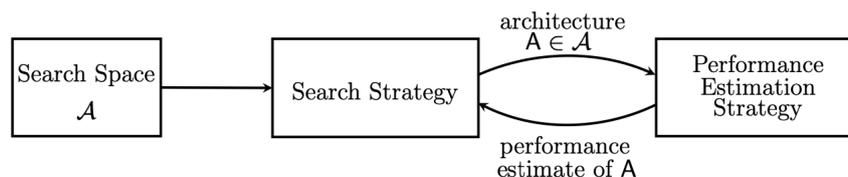
**FIGURE 1.13** – Architecture originale d'un Transformer. Sur la moitié gauche est représenté l'encodeur et sur la moitié droite le décodeur. Dans les travaux originaux de VASWANI et al. [191], le nombre de couches identiques  $N=6$ .

## 1.2 Trouver automatiquement une architecture optimisée : le cas du Neural Architecture Search (NAS)

Un aspect crucial de la réussite des ANN sont les nouvelles architectures. Les architectures actuelles sont pour la plupart manufacturées par des experts humains via un processus long, coûteux et sujet aux erreurs et l'ingénierie d'architectures de plus en plus complexes devient elle-même extrêmement complexe. Le NAS, dont des reviews approfondies sur lesquelles nous baserons cette section ont été effectuées par ELSKEN, METZEN et HUTTER [50], WISTUBA, RAWAT et PEDAPATI [203] et REN et al. [160] permet de répondre à cette problématique, en proposant dès 2018 des architectures plus performantes que celles proposées par des humains sur certaines tâches comme la classification d'images dans des travaux de ZOPH et al. [218] et REAL et al. [159] ou encore la segmentation d'images, par CHEN et al. [25].

Le NAS est un sous domaine du Machine Learning Automatisé (AutoML) (domaine étudié par HUTTER, KOTTHOFF et VANSCHOREN [91] notamment), qui est le processus d'automatisation des tâches fastidieuses et itératives de développement de modèles de Machine Learning. Il recouvre également l'optimisation des hyper-paramètres des modèles (voir l'article de FEURER et HUTTER [56]) et peut être vu dans une certaine mesure comme du meta-learning (comme étudié par VANSCHOREN [189]) puisque pour améliorer le processus d'apprentissage en soi, optimiser l'architecture du modèle qui apprend est un des paramètres à prendre en compte (tout comme la manière dont il apprend par exemple).

Automatiser cette recherche est donc un domaine très actif qui peut être délimité en trois catégories d'après ELSKEN, METZEN et HUTTER [50] : l'étude de l'espace de recherche, l'étude de la stratégie de recherche et l'étude de la stratégie d'estimation de la performance. Ils illustrent ceci dans la Figure 1.14.



**FIGURE 1.14** – Illustration des méthodes de NAS, d'après ELSKEN, METZEN et HUTTER [50]. Une stratégie de recherche sélectionne une architecture  $A$  issue d'un espace de recherche prédéfini  $\mathcal{A}$ . Cette architecture est passée à une stratégie d'évaluation de la performance qui retourne la performance estimée de  $A$  à la stratégie de recherche.

La principale limite du NAS et de manière générale de l'AutoML est qu'il demande de comparer et donc d'apprendre et d'évaluer plusieurs modèles, ce qui impose des contraintes calculatoires importantes dans le cas répandu des Transformers. Ces contraintes calculatoires limitent donc le nombre d'architectures explorables et de nombreux travaux limitent donc l'espace de recherche afin de trouver une architecture localement optimale dans un espace de recherche souvent discret.

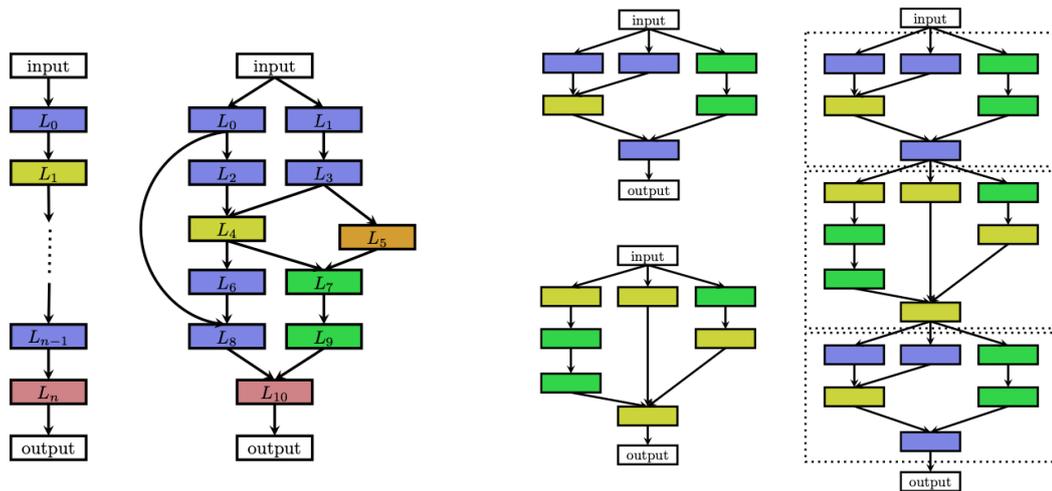
### 1.2.1 Étude de l'espace de recherche

D'après la définition donnée par ELSKEN, METZEN et HUTTER [50], l'espace de recherche définit quelles sont les limites imposées aux architectures recherchées. Pour simplifier la recherche, il est possible d'introduire des connaissances humaines à propos des architectures adaptées à telle ou telle tâche et ainsi réduire la taille de cet espace de recherche. Seulement ce genre de biais humains limite l'espace et peut empêcher de découvrir de nouvelles briques élémentaires ou des assemblages originaux qui dépassent la connaissance humaine actuelle. Les espaces de recherche peuvent donc être contraints à des architectures simples ou rester le plus libres possibles, au risque d'un temps de calcul extrêmement long. Plusieurs types d'espaces de recherche sont ainsi couramment utilisés.

- Dans l'espace de recherche des ANN à structures chaînées (*chain-structured neural networks search space*), l'architecture  $A$  est définie comme une séquence de  $n$  couches  $A = L_n \circ \dots \circ L_1 \circ L_0$ . L'espace de recherche est ainsi paramétré par le nombre  $n$  maximal de couches, le type d'opérations admises à chaque couche (quelles "briques élémentaires" sont accessibles : par exemple les convolutions, les FFNN, etc.) ainsi que les hyper-paramètres associés à chacune des opérations réalisées (le nombre de neurones pour un FFNN par exemple).
- Dans l'espace de recherche des réseaux à branches multiples (*multi-branch networks search space*), des opérations plus complexes sont admises, telles que les skip-connexions par exemple. En effet, une skip-connexion étant une connexion dans un réseau neuronal qui relie directement une couche à une couche ultérieure en "sautant" une ou plusieurs couches intermédiaires, elle permet d'ajouter des branches supplémentaires par rapport à un réseau à structure chaînée. Elles ont été notoirement utilisées dans les architectures de réseaux résiduels (ResNet) introduites par HE et al. [76] et sont désormais une brique élémentaire des Transformers. De même l'entrée d'une couche peut être le résultat d'une combinaison de la sortie de plusieurs autres couches.
- Dans l'espace de recherche cellulaire (*cell search space*) des blocs entiers de couches de réseaux (ou cellules) sont optimisés avant d'être empilés de manière prédéfinie au lieu de définir toute l'architecture. Ce genre d'architectures est plus facilement transférable ou adaptable à d'autres données. De plus, la taille de l'espace de recherche est considérablement réduite par rapport aux deux espaces mentionnés précédemment puisque moins de degrés de liberté sont accessibles. Cette approche nécessite une optimisation à la fois au niveau de l'architecture globale (optimisation inter-cellules ou macro-architecturale), mais également à l'intérieur des cellules (optimisation intra-cellules ou micro-architecturale). En effet, d'un point de vue macro, il s'agit de définir le nombre de cellules ainsi que les connexions entre celles-ci, tandis que d'un point de vue micro il s'agit d'optimiser la structure interne de chaque cellule. Idéalement, ces deux niveaux sont co-optimisés, pour éviter de devoir manuellement construire des macro-architectures après avoir trouvé une cellule optimale.
- Pour ce faire, l'espace de recherche hiérarchique (*hierarchical search-space*) introduit par LIU et al. [128], représente une étape dans l'optimisation macro-architecturale. Dans ce cas, des niveaux de transformations hiérarchiques sont étudiés. Le premier consiste en les opérations primitives, le second en les combinaisons d'opérations primitives via un DAG, le troisième en la connexion des transformations de second

niveau, etc. En ce sens, il s'agit d'une généralisation de l'espace de recherche cellulaire puisqu'en fixant le nombre de niveaux de transformations à 3, le second niveau correspond aux cellules et le troisième aux macro-architectures définies à la main.

La Figure 1.15 représente les trois premiers espaces de recherche mentionnés et est tirée du travail de ELSKEN, METZEN et HUTTER [50].



**FIGURE 1.15** – Illustration des espaces de recherche à structures chaînées (à gauche), à branches multiples (centre gauche) et cellulaires (centre droite et droite). Chaque noeud dans les graphes correspond à une couche quelconque d'ANN (convolution, FFNN, ...). Différentes couleurs sont associées aux divers types de couches. Un lien d'une couche  $L_i$  à  $L_j$  exprime que  $L_j$  reçoit en entrée la sortie de  $L_i$ . Pour les espaces cellulaires, deux différents types de cellules sont présentés. Une cellule normale (en haut) et une cellule de réduction (en bas), type de cellules introduit par ZOPH et al. [218] dont la particularité est de réduire la dimension de l'entrée. En empilant les cellules en séquences, une architecture comme présentée sur le graphe de droite peut être obtenue. Il s'agit de noter que les cellules sont combinées de manière chaînée mais qu'elles peuvent être également combinées à la manière des combinaisons de l'espace à branches multiples en remplaçant simplement les couches d'ANNs par des cellules.

Même pour un cas relativement simple d'un espace de recherche basé sur une cellule unique à la macro-architecture fixée, la micro-optimisation reste un problème non continu en grande dimension puisque des modèles plus complexes ont tendance à avoir de meilleures performances, ce qui ouvre le champ libre à de nombreux choix de conception.

## 1.2.2 Stratégie de recherche

La stratégie de recherche est l'optimisation d'une architecture au sein de l'espace de recherche défini. Cela englobe les compromis à effectuer entre exploration totale de l'espace et l'exploitabilité rapide des architectures obtenues. En effet, il est théoriquement désirable de trouver rapidement des architectures performantes, mais cette rapidité ne doit pas se

faire aux dépens d’architectures optimales plus complexes et coûteuses à découvrir. À l’inverse, il n’est pas non plus souhaitable d’attendre indéfiniment la convergence vers une architecture théoriquement optimale qui n’apporte qu’une amélioration faible des performances comparée à des architectures moins complexes à trouver. D’un point de vue formel, l’optimisation d’architecture est exprimée comme suit par WISTUBA, RAWAT et PEDAPATI [203].

En notant l’espace de tous les jeux de données  $D$ , l’espace de tous les modèles de DL  $M$  et l’espace de recherche  $A$ , un algorithme général de Deep Learning  $\Lambda$  est défini comme le mapping

$$\Lambda : D \times A \rightarrow M.$$

Dans ce contexte, les auteurs mentionnent qu’une architecture  $\alpha \in A$  définit alors non seulement une topologie, mais également les propriétés requises pour apprendre un modèle sur un jeu de données (ce qui englobe également l’algorithme d’optimisation des paramètres, les stratégies de régularisation éventuelles et les autres hyper paramètres du modèle). Ayant un jeu de données  $d$  séparé en une partie dédiée à l’apprentissage  $d_{\text{train}}$  et une partie à la validation  $d_{\text{valid}}$ , l’algorithme de DL  $\Lambda$  estime le modèle  $m_{\alpha, \theta} \in M_{\alpha}$ . Pour ce faire, une fonction de coût (*loss*)  $\mathcal{L}$  est minimisée, souvent accompagnée d’un terme de régularisation des données d’apprentissage  $\mathcal{R}$ .

$$\Lambda(\alpha, d) = \arg \min_{m_{\alpha, \theta} \in M_{\alpha}} \mathcal{L}(m_{\alpha, \theta}, d_{\text{train}}) + \mathcal{R}(\theta).$$

Le but du NAS est alors de trouver l’architecture  $\alpha^*$  qui maximise une fonction objectif  $\mathcal{O}$  sur le jeu de données de validation. Formellement, WISTUBA, RAWAT et PEDAPATI [203] définissent ainsi :

$$\alpha^* = \arg \max_{\alpha \in A} \mathcal{O}(\Lambda(\alpha, d_{\text{train}}), d_{\text{valid}}) = \arg \max_{\alpha \in A} f(\alpha).$$

La fonction  $\mathcal{O}$  peut être la même que la fonction négative de coût  $\mathcal{L}$ . Pour un problème de classification comme nous traiterons dans cette thèse, la fonction de coût est la cross-entropy négative et la fonction objectif est l’exactitude (ou *accuracy*) de classification. Cette définition tombe donc d’après les auteurs dans le spectre de l’optimisation d’hyper-paramètres définie par BERGSTRÄ et BENGIO [14]. Pour ce faire, plusieurs méthodes sont généralement employées.

Depuis les années 1990, les algorithmes évolutionnistes (*Evolutionary Algorithms*, EA) ont été utilisés pour optimiser à la fois l’architecture d’un modèle et ses poids (voir notamment les travaux menés par DING et al. [43], ELSKEN, METZEN et HUTTER [50] ou encore STANLEY et al. [179]). Avec la complexité des réseaux contemporains, les approches neuro-évolutionnistes utilisent les EAs pour optimiser les architectures des ANNs (par exemple par ELSKEN, HENDRIK METZEN et HUTTER [51] ou LIU et al. [128]). Les paramètres de ces modèles sont optimisés classiquement avec des méthodes basées sur la descente de gradient. Dans ce contexte, l’ensemble des modèles est construit grâce à un processus évolutionniste qui à chaque itération utilise au moins un des modèles de la population courante comme parent pour générer de nouveaux modèles. Des modifications locales des réseaux parents comme la modification d’hyper-paramètres ou de structure (ajout ou suppression d’une couche par

exemple) donnent ainsi lieu aux nouveaux modèles. Les modèles ainsi créés sont ensuite appris et évalués avant d'être ajoutés à la population de modèle.

Le NAS peut être également vu comme un problème d'Apprentissage par Renforcement (*Reinforcement Learning*, RL) (voir le travail pionnier de SUTTON et BARTO [182]) où l'architecture est générée de manière séquentielle à partir des actions d'un agent. Ainsi, ZOPH et LE [217] utilisent un RNN pour générer des architectures convolutives. De même, CAI et al. [19] proposent que l'agent de RL explore l'espace de recherche à l'aide d'opérations de transformation du réseau comme l'élargissement de couches, l'insertion de nouvelles couches ou l'ajout de skip-connexions, etc. Afin de dépasser les opérations au niveau des couches, CAI et al. [20] introduisent une transformation à un niveau dit "*path-level*" qui permet à l'agent de RL d'explorer un espace de recherche semblable à un arbre, représentant la généralisation des architectures multi-branches. Il s'agit de noter que ces techniques de NAS sont appliquées pour trouver notamment des architectures de Transformers, dans CHITTY-VENKATA et al. [29].

Après avoir fait émerger de nouvelles architectures, il faut évaluer leurs performances afin de les comparer entre elles mais également aux architectures standards pour déterminer quelles architectures performant le mieux sur des données non vues (capacité de généralisation). C'est ici que la stratégie d'estimation de la performance entre en jeu.

### 1.2.3 Stratégie d'estimation de la performance

L'objectif intrinsèque du NAS est de trouver automatiquement des architectures originales qui généralisent mieux que les architectures classiques. Pour estimer cette performance, la manière de procéder typique est d'apprendre le modèle sur une partie de *train* des données et d'estimer les performances du modèle sur une partie de validation des données. Cette manière de procéder est simple et standard mais elle est coûteuse en termes de calcul et limite donc l'ensemble des architectures pouvant être explorées. De récents travaux, s'inspirant de l'algorithme DARTS proposé par LIU, SIMONYAN et YANG [127] étendent cependant leur recherche à un espace de recherche *continu* afin de rechercher efficacement des architectures en utilisant des algorithmes d'optimisation dérivés de la descente de gradient, ce qui réduit significativement les ressources computationnelles nécessaires. Cependant, les architectures ainsi découvertes sont moins bonnes que celles obtenues via les méthodes évolutionnistes ou de RL qui atteignent les meilleures performances, comme en atteste notamment le travail de TAN et LE [183]. De plus, évaluer et représenter de manière adéquate différents réseaux proches dans un espace de recherche continu est toujours aujourd'hui une question ouverte. Nous verrons dans la suite d'autres mesures de généralisation, certaines moins coûteuses et nous présenterons une mesure que nous avons développée qui se montre utile pour mesurer les performances de généralisation sur des tâches de classification binaire sans avoir à recourir à un jeu de données de validation.

## 1.3 Les réseaux de neurones à architecture dynamique

La section précédente a permis de présenter de manière générale le domaine du NAS qui consiste à automatiquement développer des architectures originales et optimisées faisant partie d'un espace de recherche défini préalablement, optimisant conjointement l'architecture et les poids du modèle. Mais il existe également des réseaux de neurones dont l'architecture pourrait être qualifiée de dynamique : certains réseaux sont sur-paramétrés et appris comme tels avant de voir leur taille être réduite et à l'inverse d'autres réseaux grossissent progressivement à mesure que de nouvelles données leur sont présentées. Même si notre travail porte sur ces derniers, nous allons rapidement présenter dans cette section les divers travaux qui font office d'état de l'art des réseaux de neurones à architecture dynamique.

### 1.3.1 Réduire la taille d'un ANN

Depuis une dizaine d'années les modèles de DL se complexifient et leur nombre de paramètres augmente considérablement. Ainsi nous sommes passés de 62,3 millions de paramètres pour AlexNet proposé par KRIZHEVSKY, SUTSKEVER et HINTON [112] à 355M pour RoBERTa large introduit dans LIU et al. [129] ou encore  $\sim 770M$  pour le récent T-5 large de RAFFEL et al. [157]. Les modèles les plus impressionnants en termes de taille sont à ce jour PaLM issu du travail de CHOWDHERY et al. [34] de Google avec 540 milliards de paramètres ou encore Wu Dao 2.0 développé par la Beijing Academy of Artificial Intelligence (BAAI) et ses 1750 milliards de paramètres; et les contraintes de taille semblent ne plus importer. À l'heure de finir la rédaction de ce manuscrit cependant, Meta a annoncé LLaMa par TOUVRON et al. [185] qui semble par exemple, dans sa version avec 65 milliards de paramètres, être compétitif avec les 540 milliards de paramètre de PaLM. Les améliorations du hardware et leur spécialisation permettent ces avancées, mais ce genre de modèles n'est pas adapté à des appareils à faible ressource comme les smartphones par exemple et il est donc nécessaire que les coûts d'apprentissage et d'inférence (c'est-à-dire d'utilisation "réelle" du modèle pour prédire une nouvelle valeur) soient optimisés d'un point de vue énergétique puisque ce genre d'appareil est limité par sa batterie. Les GPU (*Graphic Processing Units*) sont capables d'effectuer un très grand nombre de calculs en parallèle, mais ne sont pas très efficaces énergétiquement.

Plutôt que d'espérer une amélioration technologique du hardware, une amélioration des modèles est donc nécessaire. Il est donc nécessaire d'introduire des réseaux moins denses (ou *sparse* pour parcimonieux dans le sens où les connexions dans le modèle sont réalisées avec plus de parcimonie, résultant en des réseaux plus petits). Cette idée remonte au travail de SRIVASTAVA et al. [178], où l'abandon partiel (en anglais *dropout*), une technique de régularisation désormais classique qui "éteint" aléatoirement certains neurones durant l'apprentissage a été introduit. Un abandon partiel adaptatif a été proposé par SPRING et SHRIVASTAVA [177] en utilisant un hachage aléatoire, résultant en un calcul réduit à 5% de l'état initial tout en gardant des résultats à 1% près des résultats initiaux. Pour ce faire, les noeuds du réseau à faible activation ne sont pas utilisés. Ceci permet d'après les auteurs de contrôler la complexité du réseau, mais également de réduire largement le nombre de calculs nécessaires puisque moins d'opérations sont requises. Contrairement à l'abandon

partiel standard où les activations des neurones étaient tout de même calculées, les auteurs montrent que sélectionner l'ensemble de neurones à activation élevés peut être approximé à la recherche du produit scalaire maximal en utilisant un hachage sensible aux asymétries locales (*asymmetric locality sensitive hashing* (LSH)). L'idée est de mapper des éléments similaires ensemble pour que la probabilité de collision de ces éléments soit élevée si ils sont proches d'un point de vue d'une mesure de similarité. Ces résultats permettent donc d'obtenir de manière très efficace un ensemble de neurones à haute activation par étape d'apprentissage en mappant les neurones dans des tables de hachage en fonction de leurs produits scalaires. Ceci montre notamment que les neurones d'un ANN se spécialisent et ne contribuent de manière active que pour des exemples spécifiques, mais également que de nombreuses connexions sont redondantes.

De même, le framework SLIDE (pour *Sub-Linear Deep learning Engine*), proposé par CHEN et al. [24] réduit drastiquement les calculs à la fois d'apprentissage et d'inférence, permettant d'utiliser des modèles sur des CPU et sur-performant par rapport à des calculs classiques sur GPU. Dans cet article, les auteurs mettent en place plusieurs améliorations d'implémentation en utilisant le LSH, notamment au niveau de l'initialisation et de la passe avant (*forward pass*). Cependant, ces idées ne sont pas applicables à des GPU puisque les performances démontrées empirent avec un accès limité à la mémoire.

Enfin SHAZEER et al. [170] proposent un modèle basé sur des calculs conditionnels permet d'augmenter la taille d'un modèle (lui permettant d'être efficace sur un grand nombre de données) jusqu'à 137 milliards de paramètres tout en réduisant les coûts de calcul. L'objectif ici est d'augmenter la capacité d'un modèle sans impacter le temps de calcul nécessaire. Faire grossir simplement la taille d'un réseau (en augmentant son nombre de connexions, de neurones ou de couches) augmente également le nombre d'opérations nécessaires et donc le temps de calcul. Les auteurs proposent un modèle appelé *sparsely-gated mixture of experts layer* (MoE) dans lequel des réseaux experts sont appris en utilisant des réseaux "portes" (*gating*) qui sélectionnent une combinaison des experts pour chaque exemple d'entrée. Le modèle final est donc beaucoup plus massif qu'un modèle appris classiquement mais avec des connexions plus parcimonieuses puisque selon l'exemple d'entrée, les réseaux de gating ne sélectionnent qu'une partie des experts. Le réseau de gating choisi initialement aléatoirement les experts puisqu'il n'est pas appris. Il est également nécessaire d'équilibrer le choix des experts car dans le cas où un expert serait choisi à chaque fois, cela produirait un réseau expert unique "spécialisé" sur tout le corpus et donc un réseau dense. Ceci est évité en ajoutant un terme additionnel à la fonction de coût du modèle. Cette proposition permet donc potentiellement d'atteindre des modèles énormes pouvant résoudre des problèmes d'apprentissage complexes sans toutefois compromettre le temps de calcul de tels réseaux. Il est à noter que Wu Dao 2.0 a également été appris en utilisant une variante du MoE (plus précisément le FastMoE développé par HE et al. [75]).

Le sujet de rendre les DNN plus parcimonieux (moins denses et donc avec du "vide") rejoint également le fait que la plupart des entrées réelles sont elles mêmes "parcimonieuses" dans le sens où certaines parties de l'entrée n'ont pas d'utilité dans le monde des concepts. Ainsi dans une image, les pixels composant l'objet d'intérêt sont entourés d'autres pixels "inutiles", de même que dans un texte, beaucoup de mots n'apportent que peu de sens au contexte (par exemple les mots outils). Ce concept peut donc être utilisé pour réduire la taille

des réseaux en réduisant le nombre d'opérations nécessaires et ce soit en supprimant des connexions, ce qui est le cas de l'**élagage** (*pruning*) ou en extrayant les informations apprises par un réseau massif pour composer un réseau plus petit avec les mêmes performances comme c'est le cas de la **distillation**. Nous allons rapidement présenter ces deux pratiques dans la suite.

### Le cas de l'élagage : le *pruning*

L'élagage est une des techniques employées pour compresser les modèles (réduire leur taille sans trop réduire leurs performances). Il s'agit de supprimer des poids d'un modèle appris.

En pratique, FRANKLE et CARBIN [59] montrent qu'il est presque toujours possible de trouver un sous-réseau plus petit avec des performances similaires au réseau original. Ces sous-réseaux sont appelés les "tickets gagnants" par les auteurs et leur initialisation joue un rôle prépondérant. Il y a plusieurs manières de procéder, à savoir supprimer des connexions en mettant le paramètre en question à 0, ce qui rend le réseau plus parcimonieux. Cela réduit le nombre de paramètres du modèle mais garde l'architecture inchangée. Il est également possible de supprimer entièrement des neurones, ce qui réduit l'architecture du réseau tout en voulant garder néanmoins les performances du réseau initial. Ces deux méthodes sont illustrées dans la Figure 1.16 tirée du travail de HAN et al. [74].

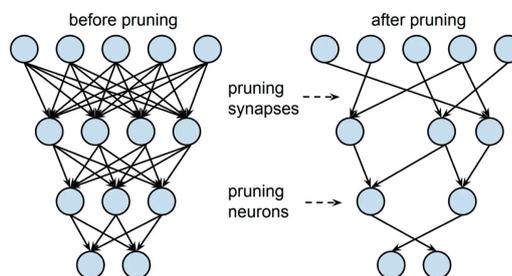


FIGURE 1.16 – Illustration de l'élagage de poids(synapses) vs élagage de neurones.

Ces deux méthodes ont des avantages comme des inconvénients. En supprimant seulement des connexions, les performances du modèle sont peu impactées mais cela rend le réseau plus sparse, ce qui requiert des efforts de calcul différents notamment pour les GPU, mais également un certain taux de parcimonie pour fonctionner. À l'inverse, supprimer des nœuds entiers permet de garder des réseaux denses pour lesquels les calculs sur GPU sont optimisés, potentiellement au détriment des performances du modèle. Dans les deux cas il s'agit de supprimer les paramètres les moins importants et différentes heuristiques et méthodes existent pour ce faire comme le montrent BLALOCK et al. [15], en supprimant les redondances, les para-

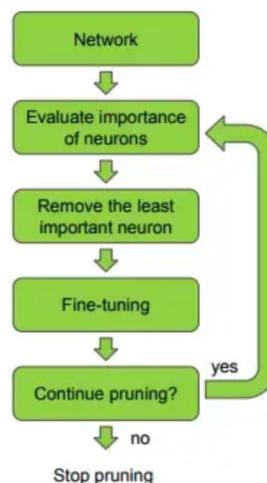


FIGURE 1.17 – Élagage itératif proposé par MOLCHANOV et al. [138].

mètres avec peu d'impact, etc. Il faut également déterminer le moment pour effectuer l'élagage. Pour ce faire, MOLCHANOV et al. [138] proposent notamment un élagage itératif illustré Figure 1.17. Enfin pour évaluer l'élagage, il est nécessaire non seulement d'évaluer les performances du modèle obtenu, mais aussi sa taille (nombre de paramètres ou nombre de bytes nécessaire pour le stockage), son temps de calcul (en utilisant par exemple les FLOPs *Floating point operations*), et il est noté par BLALOCK et al. [15] qu'un framework uni d'évaluation est nécessaire. Ainsi il existe quatre problématiques principales du *pruning*, à savoir *quoi* élaguer, *combien* de neurones (ou connexions) supprimer, *comment* et *quand* élaguer. Notre travail ne couvre pas ce sujet, mais pour une revue détaillée, nous invitons le lecteur intéressé à lire la revue de HOEFLER et al. [84]. Par ailleurs, alors que le *pruning* supprime classiquement des connexions ou des neurones après l'apprentissage et la convergence d'un réseau ; il existe également des méthodes d'élagage à l'initialisation, une étude approfondie de l'état de l'art ainsi que des problèmes encore ouverts sur ce sujet étant disponible dans l'article de WANG et al. [198].

### Transmettre de la connaissance à un ANN plus petit : la distillation

Initialement introduite par BUCILUĂ, CARUANA et NICULESCU-MIZIL [18] sous la forme d'une compression de modèle pour transférer l'information d'un large modèle ou d'un ensemble de modèles pour apprendre un petit modèle sans perte significative de performances, le transfert de connaissance entre un modèle professeur appris de manière supervisée et un modèle élève utilisant des données non labellisées est également introduite pour l'apprentissage semi-supervisé par URNER, BEN-DAVID et SHALEV-SHWARTZ [187]. Formalisée par HINTON, VINYALS et DEAN [81], la distillation (*knowledge distillation*, KD) a pour principe d'utiliser les connaissances acquises d'un modèle massif pour apprendre un modèle plus petit. Il s'agit donc de l'apprentissage supervisé d'un petit modèle à partir d'un gros modèle appris (voir par exemple les travaux de BUCILUĂ, CARUANA et NICULESCU-MIZIL [18], BA et CARUANA [5] et HINTON, VINYALS et DEAN [81] ou encore URBAN et al. [186]), l'idée principale étant que le petit modèle apprend à copier le modèle enseignant pour obtenir des performances similaires. Pour ce faire, un système de distillation est composé de l'algorithme de distillation, d'une architecture couplée enseignant/élève et de la connaissance acquise par le modèle enseignant. Ceci est illustré Figure 1.18, tirée du travail de GOU et al. [66]. La distillation n'entrant pas dans le cadre de notre travail, nous encourageons le lecteur intéressé à se diriger vers la revue de GOU et al. [66] pour une étude complète et détaillée de la distillation.

Ces modèles plus petits ont tout de même besoin que l'on apprenne un modèle plus gros au préalable. Afin d'éviter ceci, l'idée dans la section suivante est d'étudier le cas des neurones grossissants, qui en partant d'une architecture petite voire minimale, atteignent des réseaux de petites tailles aux performances comparables à des gros réseaux.

### 1.3.2 Augmenter la taille d'un ANN

Comme vu précédemment, les ANN standards utilisent typiquement une architecture statique définie avec un nombre fixe de couches et de neurones qui sont construites grâce

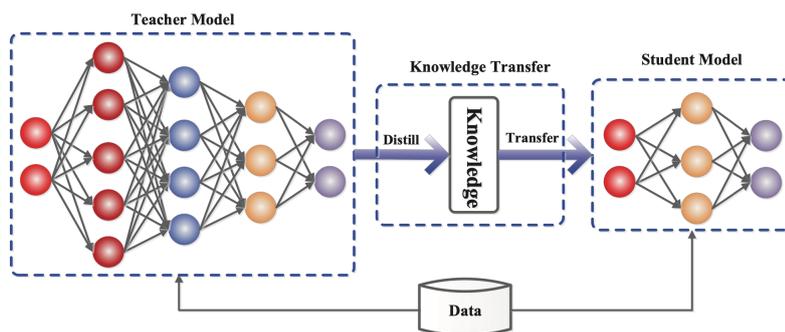


FIGURE 1.18 – Framework générique de distillation, par GOU et al. [66].

à de coûteuses expériences. Nous avons ainsi vu dans la section précédente qu'automatiser cette recherche est un champ de recherche actif, appelé NAS par ELSKEN, METZEN et HUTTER [50] avec très notablement des propositions de DAI, YIN et JHA [38, 37] d'utiliser un paradigme d'expansion et élagage (*grow-and-prune*) pour itérativement ajouter ou enlever des neurones qui obtiennent des ANNs plus petits en taille (et donc moins complexes à optimiser) et compétitifs avec leur contrepartie statique une fois que les questions suivantes sont correctement envisagées : **où, quand, comment**.

Les réseaux de neurones grossissants (*GrowNN*) sont un sujet de recherche actif et relativement ancien. Une étude historique complète est ainsi faite par QIANG, CHENG et WANG [155] et des travaux récents utilisent cette idée de réseaux progressivement grossissants, par exemple dans le contexte des Generative Adversarial Networks (proposés par GOODFELLOW et al. [64]) pour ajouter petit à petit des nouveaux détails pendant l'apprentissage afin de progressivement atteindre des détails fins des données. Ainsi KARRAS et al. [100] augmentent progressivement la résolution d'images de faibles résolutions en ajoutant de nouvelles couches au réseaux, ce qui lui permet d'apprendre des détails de plus en plus fins au lieu de devoir tout apprendre d'un coup.

Les *GrowNN* sont souvent utilisés dans le domaine du Continual Learning (défini par PARISI et al. [148]) où le modèle doit être capable d'apprendre progressivement des nouvelles connaissances, et ce sur des longues périodes de temps, sans toutefois interférer avec les connaissances préalablement apprises. Ce phénomène d'oubli des données précédemment apprises au fur et à mesure de la découverte de nouvelles données et donc la dégradation des performances sur ces anciennes données est appelée le *catastrophic forgetting* (introduit par McCLOSKEY et COHEN [134]) et peut être évité en ajoutant de nouveaux neurones, par exemple dans le travail de LI et al. [123].

La possibilité de transférer la connaissance d'un petit réseau vers un nouveau réseau plus large ou plus profond est explorée depuis le travail de CHEN, GOODFELLOW et SHLENS [27] qui proposait déjà une réponse à la question "où insérer des nouveaux neurones?" en créant à la fois des nouveaux neurones dans les couches existantes mais également en générant entièrement de nouvelles couches. Des travaux récents ont également montré que les *GrowNN*, en commençant de petits réseaux et en les grossissant progressivement à la fois en largeur et en profondeur, répondent au problème du *catastrophic forgetting*. Ces réseaux dynamiques atteignent également des modèles avec une fraction des paramètres

originaux (avec par exemple des tailles de l'ordre de 4% du nombre de paramètres original) et néanmoins meilleurs que les modèles état de l'art à l'époque avec des méthodes telles que *Learn-to-Grow* introduite par LI et al. [123], *Compact-Pick-Grow* proposée par HUNG et al. [90], ou encore récemment *Firefly Neural Architecture Descent* issue du travail de WU et al. [204]. L'augmentation dynamique de la taille des réseaux est également étudiée dans le champ de l'apprentissage fédéré afin de prendre en compte les spécificités des clients sans nuire à la généralisation par exemple dans les travaux de EK et al. [48] ou encore WANG et al. [197].

L'initialisation des nouveaux neurones a également été étudiée par EVCI et al. [53], proposant une réponse à "comment insérer et initialiser les nouveaux neurones?". Enfin la question "quand insérer les nouveaux neurones" possède plusieurs réponses, par exemple par WANG et al. [196] qui attendent que la fonction de coût d'apprentissage atteigne un plateau avant d'ajouter de nouveaux paramètres, tandis que d'autres travaux utilisent une temporalité régulière définie au préalable.

Cependant, la question restant entière est le "pourquoi" ces réseaux de neurones grossissants atteignent des performances relevant de l'état de l'art tout en étant plus petit que les modèles compétitifs (par exemple, une exactitude de test semblable à celle du modèle total avec seulement 4% des paramètres totaux dans le travail de WU et al. [204]). Une idée naturelle pour y répondre est la suivante : à mesure que l'espace des paramètres grossit, la topologie de l'espace des paramètres se complexifie au fur et à mesure de l'apprentissage et ce qui était précédemment un minimum local peut devenir un point selle et ainsi le modèle peut s'en échapper dans un espace de dimension supérieure, ce qui donne une décroissance continue de la fonction de coût dans l'article de WANG et al. [196]. KAWAGUCHI et KAEHLING [102] ajoutent par exemple un neurone spécial par neurone de sortie et suppriment ainsi tous les minima locaux sous optimaux de tout DNN. Bien que relativement inapplicable dans les faits, ce résultat montre que la manière originale d'explorer l'espace des paramètres qu'ont les GrowNN peut être une des raisons de leurs performances. Nous présentons dans les chapitres suivants un algorithme de GrowNN le plus générique afin d'étudier ensuite, dans le contexte de l'étude des paysages de la fonction de coût un nouveau point de vue pour apporter une réponse à ce "pourquoi".



---

## CHAPITRE 2 : RÉSEAUX DE NEURONES EXPANSIFS : MODÈLE ET IMPLÉMENTATION

*La vraie science est une ignorance qui se sait.*

---

– Michel de Montaigne

---

2.1	Vers un cadre général pour les architectures expansives . . . . .	32
2.2	Expansion contrôlée : atteindre une architecture prédéfinie . . . . .	39

---

Dans ce chapitre, nous présentons un modèle de réseau de neurones expansif original, à la fois flexible et générique. Ce modèle a évolué au cours de nos travaux et nous en présenterons dans cette première partie les motivations originales ainsi que les différentes versions. Dans un second temps, nous montrerons comment construire en restreignant algorithmiquement ce cadre générique, des réseaux cibles avec lesquels nous pourrons nous comparer dans le chapitre suivant, toutes choses étant égales par ailleurs.

## 2.1 Vers un cadre général pour les architectures expansives

L'idée de développer un algorithme original d'expansion vient d'une volonté d'exploration et de simplicité. En effet, l'objectif était de développer un algorithme d'expansion le plus libre possible, utilisant un minimum d'heuristiques comparés aux algorithmes d'expansion standards précédemment mentionnés et ainsi introduisant le moins de contraintes possibles. Ceci avait pour but d'isoler au mieux le mécanisme d'expansion afin d'étudier le comportement du réseau résultant et surtout les divergences entre ce comportement et celui d'un réseau standard.

### 2.1.1 Présentation et limites du premier modèle

Les notations suivantes sont inspirées des travaux de KAWAGUCHI, KAEHLING et BENGIO [103]. Nous considérons le cas général d'ANN d'une profondeur quelconque pour une classification à  $k$  classes et comme mentionné dans la Section 1.1.1, ce réseau peut être assimilé à un DAG avec des fonctions d'activation non-linéaire telles que ReLU; ceci incluant notamment toute structure de FFNN avec des couches entièrement connectées ou non et avec potentiellement des skip connections.

Soient  $N = \{1, \dots, n\}$  les neurones d'un réseau. Pour deux neurones  $(i, j) \in N^2$ , on note :

- $w_{j,i}$  le poids de la connexion de  $j$  vers  $i$ . Si elle n'existe pas  $w_{j,i} = 0$ . On peut noter que la matrice sparse  $n \times n W = [w_{j,i}]_{1 \leq j, i \leq n}$  définit un graphe acyclique direct;
- $\sigma_i$  est la fonction d'activation du neurone  $i$ ;
- $\pi(i)$  est l'ensemble des parents de  $i$  :  $\pi(i) = \{j | w_{j,i} \neq 0\}_{1 \leq j \leq n}$
- $d(i)$  est la profondeur de  $i$  : c'est à dire le plus long chemin pour atteindre  $i$  depuis n'importe quelle entrée.

Il y a équivalence entre la représentation d'un ANN par couches ou par profondeurs et la couche  $l$  est composée de tous les neurones à la profondeur  $l$ .

Étant donné un vecteur d'entrée  $x$ , on définit la pré-activation d'un neurone  $i$  de profondeur  $l$  récursivement comme :

$$z_i(x, W) = \sum_{j \in \pi(i)} \sigma_j(z_j(x, W)) w_{j,i}. \quad (2.1)$$

Dans notre modèle, les neurones sont insérés incrémentalement à un pas régulier (i.e., à un intervalle de temps constant) durant la phase d'entraînement, commençant à un stade minimal (seulement les couches d'entrée et de sortie) et nous augmentons le nombre de paramètres à apprendre avec le nombre d'époques.

Ainsi, initialement,

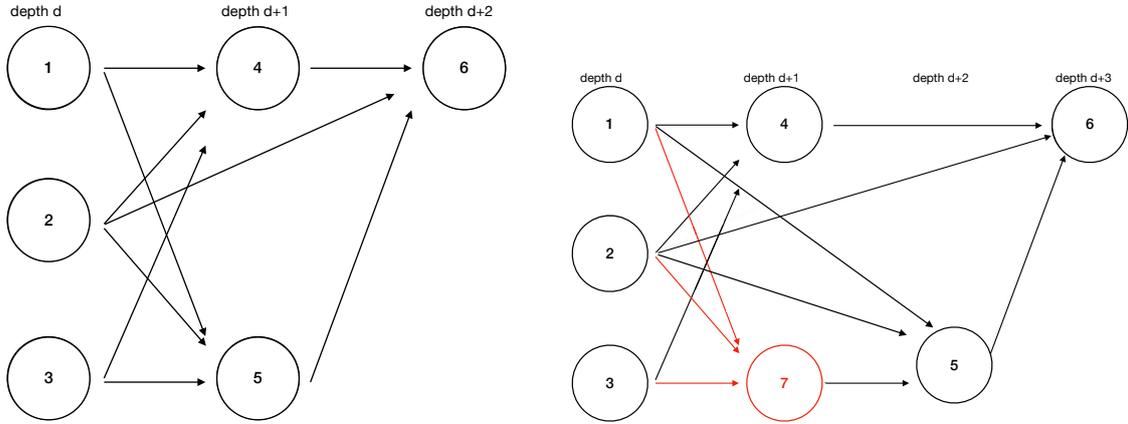
- L'ensemble de neurones est  $N = \{1, \dots, n_I, n_{I+1}, \dots, n\}$  : les  $n_I$  premiers neurones sont les neurones d'entrée, tandis que les  $n - n_I$  sont ceux de sortie ;
- La matrice de poids  $W$  est nulle excepté dans le coin haut-droit, avec les connexions non-nulles initialisées aléatoirement, par exemple en utilisant l'initialisation de GOROT et BENGIO [61] ;
- La profondeur des entrées est de 0 et celle des sorties est de 1.

On choisit aléatoirement un neurone du réseau courant, appelé le *parent primaire*, ainsi que  $I - 1$  autres neurones sur des couches plus profondes qui seront désignés comme *parents*. Ainsi à chaque étape, le nombre de paramètres du modèle est augmenté de  $I$ . Ce processus est ainsi itéré jusqu'à l'obtention du nombre désiré de paramètres.

Intuitivement, ce processus naïf d'insertion peut être considéré comme l'insertion d'un nouveau neurone après son parent primaire en héritant certains de ces enfants, comme détaillé dans l'Algorithme 1.

La connexion entre le parent primaire et le nouveau neurone est stockée dans  $\pi^0$  qui est l'ensemble contenant les paires (parent primaire, neurone inséré). Cet ensemble est initialisé comme contenant toutes les connexions entre entrées et sorties, considérant que les neurones d'entrée sont les parents primaires de tous les neurones de sortie :  $\pi^0 = \{(1, n_{I+1}), \dots, (1, n), \dots, (n_I, n_{I+1}), \dots, (n_I, n)\}$ .

Ce processus d'insertion est illustré Figure 2.2.



(a) Avant insertion : dans cet exemple 5 avait été inséré avec 3 comme parent primaire

(b) Insertion avec 3 comme parent primaire. 5 est décalé d'une profondeur et son enfant 6 également par suite.

**FIGURE 2.1** – Exemple d'héritage parent/enfant avant et après insertion : les nouveaux poids sont :

$$\begin{cases} w'_{1,7} \sim \mathcal{U}(-0.5, 0.5), \\ w'_{2,7} \sim \mathcal{U}(-0.5, 0.5), \\ w'_{3,7} \sim \mathcal{U}(-0.5, 0.5) \\ w'_{7,5} = w_{3,5} \end{cases}$$

On peut faire quelques remarques sur l'Algorithme 1.

- L'étape 6 garantit que  $d(n+1) = d(j) + 1$  car  $\forall i \in \pi(n+1), d(i) \leq d(j)$ .
- Les étapes 1 et 8 assurent que les neurones de sortie sont les seuls neurones de profondeur maximale puisque : Si  $d(n+1) = D, \forall i \in \{n_{I+1}, \dots, n\}, (j, i) \in \pi^0$

**Algorithm 1:** Algorithme d'insertion initial

- 
- Input:** Matrice des poids  $W \in \mathbb{R}^{n \times n}$   
**Output:** Matrice des poids  $W' \in \mathbb{R}^{(n+1) \times (n+1)}$
- 1 Enregistrement l'ensemble initial de parents primaires  $\pi^0$
  - 2 Création d'un nouveau neurone  $n + 1$
  - 3 Création de  $W'$  en copiant  $W$  et en initialisant les nouvelles dimensions avec des poids nuls.
  - 4 Échantillonnage du parent primaire :  $j \sim \mathcal{U}(\{j | d(j) < D\}_{1 \leq j \leq n})$  où  $\mathcal{U}$  est la distribution uniforme, et  $D = \max_{1 \leq i \leq n} d(i)$  est la profondeur du réseau.
  - 5 Enregistrement du nouveau parent primaire :  $\pi^0 \leftarrow \pi^0 \cup \{(j, n + 1)\}$
  - 6 Insertion de  $n + 1$  avec  $w'_{j,n+1} \sim \mathcal{U}(-0.5, 0.5)$
  - 7 Connexion des enfants de  $j$  en tant qu'enfants de  $n + 1$  :
  - 8 **for**  $i \in \{1, \dots, n\}$  tels que  $w_{j,i} \neq 0$  **do**
    - if  $d(i) > d(n + 1)$  :
      - Soit  $w'_{n+1,i} = w_{j,i}$
      - Soit  $w'_{j,i} = 0$
    - if  $d(i) = d(n + 1)$  et  $(j, i) \in \pi^0$  :
      - Soit  $w'_{n+1,i} = w_{j,i}$
      - Soit  $w'_{j,i} = 0$
      - $\pi^0 \leftarrow (\pi^0 - \{(j, i)\}) \cup \{(n + 1, i)\}$
  - 9 Ajout de nouveaux parents pour  $n + 1$  :
  - 10 **for**  $I-1$  fois **do**
    - Échantillonnage d'un neurone  $k \sim \mathcal{U}(\{i | w'_{i,n+1} = 0 \text{ et } d(i) \leq d(j)\}_{1 \leq i \leq n})$
    - Ajout de  $k$  comme parent de  $n + 1$  :  $w'_{k,n+1} \sim \mathcal{U}(-0.5, 0.5)$
    - Itérer
- 

- Le concept de parent primaire à l'étape 8 permettait lors de la création de cet algorithme d'obtenir des architectures équilibrées, ni purement profondes, ni purement larges lorsque le nombre de neurones du réseau est faible.
- Dans l'étape 4, l'échantillonnage du parent primaire se fait sous la condition  $j | d(j) < D$ . Les neurones d'output sont ainsi les seuls neurones pouvant avoir plusieurs parents primaires et ils sont d'ailleurs les seuls neurones ne pouvant pas être parents primaires eux-mêmes.

Deux types d'initialisation ont été étudiés.

- Le premier, décrit dans l'Algorithme 1 qui assure que tous les neurones sont initialisés de la même manière, qu'ils aient fait partie du réseau depuis sa création ou qu'ils aient été insérés en cours d'apprentissage. Ce mode d'initialisation sera celui retenu car il permet de comparer les réseaux expansifs aux réseaux standards en fixant le même mode d'initialisation des neurones. Il permet ainsi d'étudier le grossissement en tant

que tel, le but étant d'avoir tous les autres paramètres égaux par ailleurs.

- Un second, dont nous présenterons rapidement les résultats dans le Chapitre 3, qui consiste à fixer à l'étape 6, le poids  $w'_{j,n+1}$  à 1 pour le parent primaire et à l'étape 10,  $w'_{k,n+1} = 0$  pour les autres, ce qui permettait d'assurer la continuité de la fonction de loss lors d'ajout de neurones puisque la fonction représentée était la même avant et après l'ajout de neurones.

Ce premier modèle élaboré au début de la thèse a permis d'obtenir des premiers résultats intéressants pour la suite de nos travaux que nous discuterons dans le Chapitre 3, mais il présentait plusieurs problèmes majeurs.

Tout d'abord il ne représentait pas le modèle d'expansion générique capable d'atteindre toutes les architectures de FFNN souhaité. En effet, le nombre de parents fixes ( $I$ ), mais également la connexion de tous les enfants sur les profondeurs suivantes de l'étape 8 qui permettaient un calcul matriciel relativement simple des activations imposait également un cadre trop restrictif pour pouvoir prétendre atteindre n'importe quel FFNN. Enfin, à moins d'insérer exactement 1 neurone après chaque neurone d'entrée (pour un total de  $I$  neurones), une forme de skip connection existe puisque les neurones d'input n'ayant pas servi de *parent primaire* restent connectés aux neurones d'output.

De plus, cet algorithme s'est révélé complexe à mettre en place puisque l'implémentation requiert d'avoir une matrice d'adjacence gardant en mémoire l'ordre dans la vectorisation de chacun des neurones parents pour chacun de leurs enfants, la propagation se faisant vers l'avant. Ainsi, dans l'équation 2.1, l'activation de  $j^*$  parent primaire de  $i$  est associée à la première valeur du vecteur  $W_i$  et chaque parent  $j$  a potentiellement un ordre de vectorisation particulier et différent pour chacun de ses enfants puisqu'ils sont échantillonnés aléatoirement lors de l'étape 10 de l'Algorithme 1. Cette position de revectorisation imposait des contraintes calculatoires supplémentaires et redondantes.

C'est pour pallier à ces défauts de conception mais également pour développer un cadre d'étude de l'expansion plus simple et générique que nous avons donc proposé une seconde version améliorée de ce modèle, l'idée directrice restant la même, à savoir un modèle dans lequel il est possible d'insérer incrémentalement des nouveaux neurones, noeud par noeud, les deux seules constantes immuables étant son entrée et sa sortie.

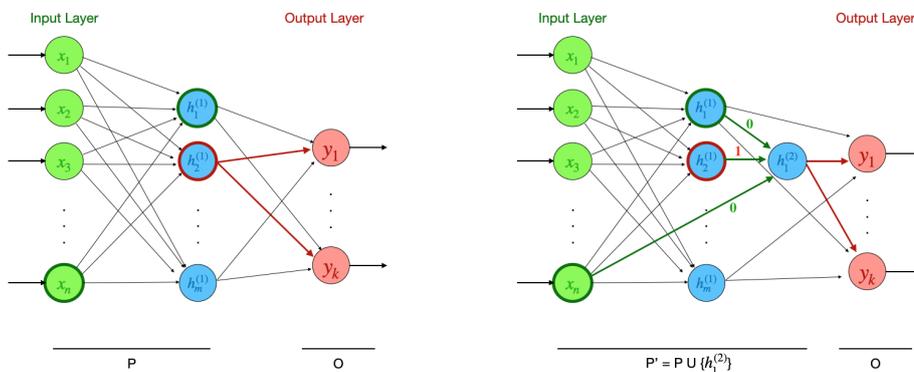
### 2.1.2 Raffinement du modèle : un cadre générique d'architecture expansive

Nous développons ici l'algorithme générique 2 finalement développé à partir de l'Algorithme 1 en résolvant les problèmes mentionnés précédemment, nous permettant ainsi cette fois d'atteindre théoriquement n'importe quelle structure de FFNN. Intuitivement, ce modèle peut être décrit ainsi : un neurone est aléatoirement choisi et un nouveau neurone qui hérite de certains de ses enfants (potentiellement tous, l'étape 4 de cet algorithme permet de choisir un seuil pour la probabilité d'hériter des enfants) est créé. Ce nouveau neurone récupère en tant que parents des neurones pré-existants aléatoirement choisis parmi les couches précédentes.

Formellement, soit  $\mathcal{P}_t$  l'ensemble des neurones non-output à l'étape d'insertion  $t$ . Soit  $\mathcal{I}$  l'ensemble des neurones d'entrée, de taille  $I$ .

Initialement,  $\mathcal{P}_0 = \{1, \dots, n_I\}$  ne contient que les  $I$  neurones d'entrée puisqu'il n'y a aucun neurone caché. La matrice de poids  $W_0$ , de même que dans l'Algorithme 1 est triangulaire supérieure avec des transitions non-nulles de l'entrée à la sortie qui sont initialisés selon un schéma d'initialisation prédéfini, comme Glorot ou la distribution uniforme  $\mathcal{U}$ . L'étape d'expansion fonctionne alors comme expliqué dans l'Algorithme 2.

Comme nous considérons les réseaux comme un graphe direct acyclique, l'ordre topologique est facilement issu d'un parcours en profondeur (*Depth-first search* DFS). Nous illustrons cet algorithme Figure 2.2.



**FIGURE 2.2** – Exemple d'une insertion :  $h_2^{(1)}$  est choisi aléatoirement et le nouveau neurone  $h_1^{(2)}$ , hérite des connexions de sortie de  $h_2^{(1)}$  (arcs rouges) et obtient des nouvelles connexions en entrées (arcs verts) initialisées aléatoirement.

L'Algorithme 2 ainsi proposé permet un framework générique de réseaux de neurones expansifs qui permet au réseau de grossir librement sans contrainte, ce qui peut amener à des réseaux parcimonieux avec potentiellement des skip-connections entre les couches. En effet, supposons l'utilisation de cet algorithme pour insérer un neurone à la fois dans un réseau et explorons quels ANN peuvent être atteints à l'issue d'appels répétés à cette procédure d'expansion. Tout d'abord, notons que tout neurone inséré est situé strictement entre les neurones d'entrée et de sortie.

Soit un FFNN quelconque de taille d'entrée  $n$  et de sortie  $m$ . Ce réseau peut être directement représenté par un graphe direct acyclique associé et donc sa matrice de poids  $W_N$  (avec  $N$  son nombre total de neurones), peut être écrite comme une matrice triangulaire supérieure  $W_N^*$  via un réordonnement topologique.

---

**Algorithm 2:** Algorithme d'Insertion Générale
 

---

- Entrées :** Poids  $W_t \in \mathbb{R}^{n \times n}$ ,  $\mathcal{P}_t$
- 1 Sorties :** Poids  $W_{t+1} \in \mathbb{R}^{(n+1) \times (n+1)}$   
 $\mathcal{P}_{t+1}$  avec  $\mathcal{P}_t \subset \mathcal{P}_{t+1}$  et  $|\mathcal{P}_{t+1}| = |\mathcal{P}_t| + 1$
- 1: Échantillonnage aléatoire d'un noeud  $j$  du réseau :  $j \sim \mathcal{U}(\mathcal{P}_t)$
  - 2: Récupération des ensembles  $\mathcal{C}_j$ ,  $\mathcal{X}_j$ ,  $\mathcal{Q}_j$ , respectivement
    - l'ensemble des noeuds enfants de  $j$  :  $\mathcal{C}_j = \{i \in \mathcal{P}_t \mid w_{j,i} \neq 0\}$ .
    - l'ensemble des noeuds topologiquement ordonnés après  $j$  :  
 $\mathcal{X}_j = \{i \in \mathcal{P}_t \mid d(i) \geq d(j) + 1\}$  (Il s'agit de noter que  $\mathcal{C}_j \subset \mathcal{X}_j$ ).
    - l'ensemble des noeuds topologiquement ordonnées avant  $j$  :  
 $\mathcal{Q}_j = \{i \in \mathcal{P}_t \mid d(i) \leq d(j) \wedge i \neq j\}$ .
  - 3: Création d'un noeud  $n + 1$  et mise à jour de  $\mathcal{P}_{t+1} = \mathcal{P}_t \cup \{n + 1\}$
  - 4: Initialisation de  $W_{t+1} = \begin{bmatrix} W_t & [0]_n \\ [0]_n^T & 0 \end{bmatrix}$  et  $w_{j,n+1}^{(t+1)} = \text{Init}()$ , avec  $\text{Init}$  un système d'initialisation quelconque au choix.
  - 5: Choix du nombre de neurones à hériter : Soit  $\alpha = \text{random}(0, |\mathcal{C}_j|)$
  - 6: Héritage de certains enfants : Échantillonnage de  $\alpha$  neurones différents  $k \sim \mathcal{U}(\mathcal{C}_j)$  et pose de  $w_{n+1,k}^{(t+1)} = w_{j,k}^{(t)}$  et  $w_{j,k}^{(t)} = 0$ . Notons l'ensemble des noeuds ainsi hérités  $\mathcal{C}_{j,\alpha}$ .
  - 7: Choix d'un nombre aléatoire d'enfants dans les profondeurs consécutives qui ne sont pas dans  $\mathcal{C}_{j,\alpha}$  : Soit  $\beta = \text{random}(\max(0, 1 - |\mathcal{C}_{j,\alpha}|), |\mathcal{X}_j \setminus \mathcal{C}_{j,\alpha}|)$
  - 8: Ajout de nouveaux enfants : Échantillonnage de  $\beta$  neurones différents  $k \sim \mathcal{U}(\mathcal{X}_j \setminus \mathcal{C}_{j,\alpha})$  et pose de  $w_{n+1,k}^{(t+1)} \sim \text{Init}()$
  - 9: Choix d'un nombre aléatoire de parents sur des profondeurs précédentes : Soit  $\gamma = \text{random}(0, |\mathcal{Q}_j|)$ .
  - 10: Échantillonnage de  $\gamma$  différents neurones parents  $k \sim \mathcal{U}(\mathcal{Q}_j)$  et pose de  $w_{k,n+1}^{(t+1)} \sim \text{Init}()$
  - 11: Choix d'un nombre aléatoire  $\delta = \text{random}(0, |\mathcal{Q}_j|)$  de  $k$  neurones sur des profondeurs précédentes pouvant chacun perdre  $i_k = \text{random}(0, |\mathcal{C}_k|)$  connexions.
  - 12: Échantillonnage de  $\delta$  neurones  $k \sim \mathcal{U}(\mathcal{Q}_j)$  et  $i \sim \mathcal{C}_k$  connexions à 0  $w_{k,i} = 0$
  - 13: Réordonnement topologique de  $\mathcal{P}_{t+1}$  tel que  $W_{t+1}$  soit triangulaire
  - 14: **retour de**  $W_{t+1}, \mathcal{P}_{t+1}$
- 

De plus, puisque :

- L'étape 4 permet d'avoir une matrice de la taille désirée.
- Les étapes 5 et 8 donnent : le vecteur de poids vertical inséré représentant les liens entre le neurone inséré et les neurones suivants n'a qu'une contrainte, le neurone inséré ne peut être lié à lui-même ou à aucun autre neurone sur la même profondeur (la connexion est fixée à 0 pour les neurones sur la même profondeur).
- Les étapes 9 et 10 donnent : le vecteur de poids horizontal inséré représentant les

liens entre les neurones précédents et le neurone inséré n'a également qu'une seule contrainte : le neurone inséré n'est lié ni à lui-même, ni à des neurones sur la même profondeur (la connexion est fixée à 0 pour les neurones sur la même profondeur).

Ces contraintes permettent au réseau de ne pas créer de cycles lors d'expansions successives.

- Les étapes 10 à 12 permettent de supprimer toute connexion précédemment existante si non désirée.
- L'étape 13 est possible grâce aux contraintes empêchant le réseau de créer des cycles lors d'expansions successives et nous donne une matrice triangulaire supérieure dont la seule contrainte est de représenter un DAG (donc sans cycle ; ainsi chaque ligne est accessible à la condition d'un lien à 0 pour les neurones de profondeurs précédentes).

Cet algorithme permet d'atteindre n'importe quelle matrice triangulaire supérieure de n'importe quelle taille supérieure  $N \geq n + m$ ,  $N - (n + m)$  étant le nombre d'étapes d'insertion nécessaires pour cela. Alors n'importe quel FFNN, sparse ou non, avec des skip-connections ou non peut être atteint avec cet algorithme. Cependant, les architectures sparse étant non optimisées au regard des calculs parallèles effectués dans les GPUs, ce genre de topologies n'est probablement pas la plus désirable. Il est cependant possible d'adapter cette formalisation générique du grossissement pour atteindre une architecture particulière cible, telle qu'un FFNN dense ainsi que nous le proposons dans l'Algorithme 3 détaillé dans la Section 2.2 suivante.

## 2.2 Expansion contrôlée : atteindre une architecture prédéfinie

L'algorithme contraint (Algorithme 3) que nous proposons ici diffère peu de l'Algorithme 2 proposé précédemment et en réalité l'adapte simplement pour atteindre des architectures classiques de FFNN denses.

---

### Algorithm 3: Algorithme d'expansion contrainte

---

**Entrées :** Poids  $W_t \in \mathbb{R}^{n \times n}$ ,  $\mathcal{P}_t$ , ensemble des neurones non input  
 $\mathcal{O}$ , l'ensemble des neurones de sortie,  $H$  : taille désirée de la couche cachée,  
 $\mathcal{I}'_t$  : ensemble d'entrées déjà utilisées dans l'étape 3,  
 $\mathcal{H}_t$  ensemble des neurones déjà insérés sur cette couche

**Sorties :** Poids  $W_{t+1} \in \mathbb{R}^{(n+1) \times (n+1)}$ ,  
 $\mathcal{I}'_{t+1}$   
 $\mathcal{H}_{t+1}$  avec  $\mathcal{H}_t \subset \mathcal{H}_{t+1}$  et  $|\mathcal{H}_{t+1}| = |\mathcal{H}_t| + 1$   
 $\mathcal{P}_{t+1}$  avec  $\mathcal{P}_t \subset \mathcal{P}_{t+1}$  et  $|\mathcal{P}_{t+1}| = |\mathcal{P}_t| + 1$

- 1: Si on peut insérer un neurone après un neurone d'input :
- 2: **if**  $\mathcal{I} \setminus \mathcal{I}'_t \neq \emptyset$  **then**
- 3: Échantillonnage aléatoire d'un noeud d'entrée  $j$  non encore échantillonné :  
 $j \sim \mathcal{U}(\mathcal{I} \setminus \mathcal{I}'_t)$
- 4: Mise à jour de  $\mathcal{I}'_{t+1} = \mathcal{I}'_t \cup \{j\}$ .
- 5: **else**
- 6: Échantillonnage aléatoire d'un noeud d'entrée  $j$  quelconque :  $j \sim \mathcal{U}(\mathcal{I})$ .
- 7: **end if**
- 8: Création d'un nouveau noeud  $n + 1$  :  $\mathcal{P}_{t+1} = \mathcal{P}_t \cup \{n + 1\}$
- 9: Initialisation de  $W_{t+1} = \begin{bmatrix} W_t & [0]_n \\ [0]_n^T & 0 \end{bmatrix}$
- 10: Lien avec les parents :  $w_{i,n+1}^{(t+1)} \sim \text{Init}()$   $\forall i \in \mathcal{I}$ .
- 11: Héritage des enfants :
- 12: **if**  $\mathcal{I} \setminus \mathcal{I}'_t \neq \emptyset$  **then**
- 13:  $w_{n+1,i}^{(t+1)} = w_{j,i}^{(t)}$   $\forall i \mid w_{j,i} \neq 0$
- 14: **else**
- 15:  $w_{n+1,i}^{(t+1)} \sim \text{Init}()$   $\forall i \in \mathcal{O}$ .
- 16: **end if**
- 17: Réordonnement topologique de  $\mathcal{P}_{t+1}$  tel que  $W_{t+1}$  est triangulaire
- 18: **if**  $|\mathcal{I}'_{t+1}| = H$  **then**
- 19: Suppression des skip-connections :  $\forall (a, b) \in (\mathcal{I} \setminus \mathcal{I}'_t, \mathcal{P}_{t+1} \setminus \mathcal{H}_{t+1}), w_{(a,b)} = 0$ .
- 20:  $\mathcal{I}'_{t+1} = \emptyset$  {Début d'une nouvelle couche}
- 21: **end if**
- 22: **Retour de**  $W_{t+1}, \mathcal{H}_{t+1}, \mathcal{P}_{t+1}, \mathcal{I}'_{t+1}$

---

Ainsi :

- Nous spécifions la taille de la couche cachée que nous sommes en train de construire afin d'atteindre une architecture prédéfinie.

- Le choix de l’initialisation est laissé à la discrétion de l’utilisateur.
- Nous contraignons les neurones à être insérés directement après les neurones d’entrée. Ainsi dès qu’une couche est "complétée", elle est considérée comme une couche de sortie et ne peut plus être modifiée.
- Tant que la taille demandée d’une couche cachée n’est pas atteinte, deux neurones ne peuvent être insérés après la même entrée, pour qu’une seule couche soit construite à la fois.
- Si la couche cachée est plus grande que l’entrée, de nouvelles connexions sont créées (les vecteurs poids des neurones de sortie gagnent en taille), mais le neurone tiré n’a pas de rôle précis.

Cet algorithme permet de créer des couches denses de manière incrémentale, ce qui sera utilisé dans la suite de notre travail. Ainsi en contraignant notre algorithme général, nous pouvons choisir la structure finale du réseau et l’adapter aux calculs sur GPUs.

Plus précisément, notre travail a été d’explorer pourquoi les GrowNN permettaient d’atteindre des résultats aussi bons que ceux mentionnés Section 1.3.2. Ainsi notre but n’a pas été d’explorer des heuristiques avancées pour faire grossir nos modèles ou pour atteindre des modèles extrêmement complexes, mais plutôt d’adopter une stratégie d’expansion simple et précise pour nous concentrer sur les mécanismes que le grossissement induisent.

De plus, il avait été prouvé dans des travaux précédents de LI et TALWALKAR [122] et NEGRINHO et al. [140] que les techniques avancées de NAS étaient souvent seulement marginalement supérieures à la recherche aléatoire, nous nous sommes donc basés bien souvent sur cette dernière afin de pouvoir nous concentrer sur les conséquences du grossissement sur le comportement du réseau.

Dans le chapitre suivant nous allons présenter et discuter des premiers résultats obtenus à l’aide de cet algorithme contraint, utilisé à la fois pour générer des réseaux simples mais également très complexes. Puis dans la Partie II, nous viserons à expliquer ces résultats et les conséquences du grossissement sur le comportement du réseau.

---

## CHAPITRE 3 : RÉSEAUX DE NEURONES EXPANSIFS : RÉSULTATS ET ANALYSE

*Toute science crée une nouvelle ignorance*

---

– Henri Michaux

---

3.1	Résultats et analyse de l’initialisation des nouveaux neurones . . .	42
3.2	Comparaison des réseaux expansifs avec des réseaux standards à architecture donnée . . . . .	45

---

Dans ce chapitre nous allons détailler les premiers résultats bruts des modèles présentés précédemment Chapitre 2. Ces résultats encourageants permettront de motiver l’exploration de leur origine que nous étudierons dans la Partie II.

Dans un premier temps nous étudierons l’influence de l’initialisation des neurones insérés et nous motiverons le choix du schéma d’initialisation adopté dans la suite. Il est à noter que la plupart des méthodes ajoutant de nouveaux neurones le font en initialisant de manière aléatoire les nouveaux neurones (voir les travaux de WU et al. [204] et HUNG et al. [90]), et nous suivrons finalement également cette méthode.

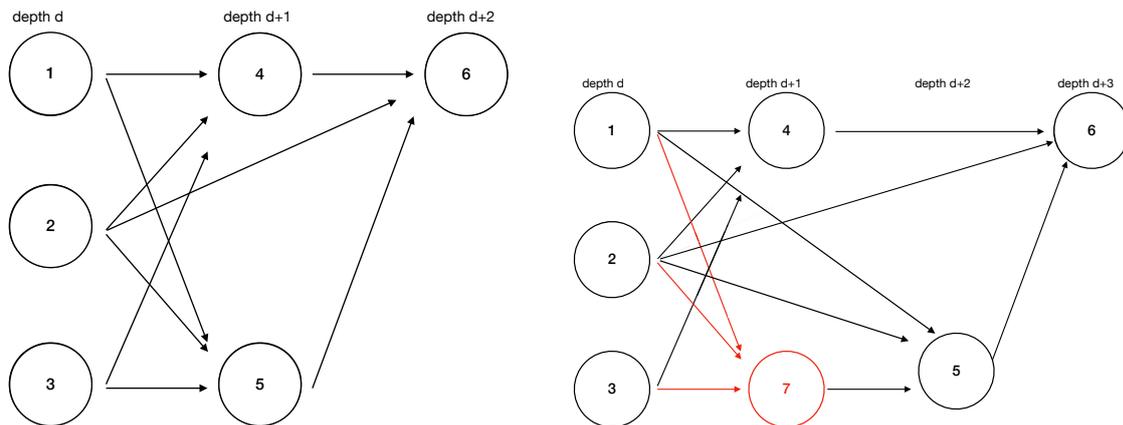
Nous nous concentrerons ensuite sur la comparaison entre les réseaux issus de ces étapes d’expansion et des réseaux standards aux architectures identiques, dans un premier temps sur des réseaux simples, puis plus complexes.

### 3.1 Résultats et analyse de l'initialisation des nouveaux neurones

Il s'agit dans cette section de distinguer deux types d'initialisation : une initialisation garantissant la continuité de la fonction représentée par le réseau et une initialisation suivant un schéma standard d'initialisation d'ANNs.

#### 3.1.1 Initialisation par continuité de la fonction de coût

Tout d'abord l'initialisation qui assure la continuité de la fonction de coût. Nous désignons cette méthode d'insertion et d'initialisation par "insertion continue". Celle ci assure la continuité de la fonction de coût et donc de l'apprentissage et peut être ainsi utile pour augmenter la taille d'un réseau face à des données arrivant par flux, par exemple des données issues des réseaux sociaux ou des logs de systèmes. Cela peut donc être appliqué dans le cas du Continual Learning (dont une revue est faite par PARISI et al. [148]) dont nous avons déjà discuté au Chapitre 1. La continuité d'apprentissage permettrait de ne pas perdre les performances sur les données précédemment traitées, et les neurones insérées pourraient apprendre des nouveaux paramètres relatifs aux nouvelles données en évitant ainsi l'oubli catastrophique. Pour garantir de ne pas avoir de *catastrophic forgetting*, il faudrait cependant geler les poids des anciens neurones afin de ne pas dégrader leurs connaissances et faire en sorte de limiter l'impact des nouveaux neurones lors des phases d'inférence sur les tâches sur lesquelles ils n'ont pas été entraînés, par exemple à l'aide d'un système de portes (ou *gating*).



(a) Avant insertion : dans cet exemple 5 avait été inséré avec 3 comme parent primaire (b) Insertion avec 3 comme parent primaire. 5 est décalé d'une profondeur et son enfant 6 également par suite.

FIGURE 3.1 – Exemple d'héritage parent/enfant avant et après insertion avec l'initialisation assurant

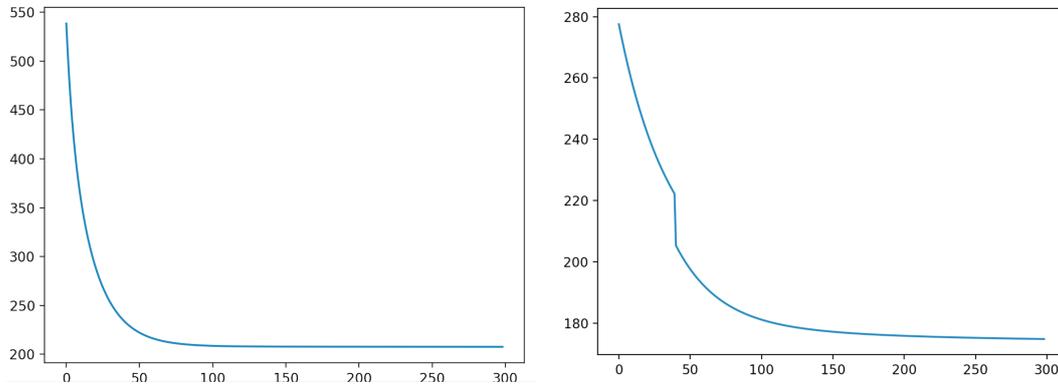
$$\text{la continuité : } \begin{cases} w'_{1,7} = 0, \\ w'_{2,7} = 0 \\ w'_{3,7} = 1 \\ w'_{7,5} = w_{3,5} \end{cases}$$

Cette initialisation nécessite de fixer initialement le poids d'entrée entre le neurone à partir duquel on insère et le nouveau neurone à 1, tandis que les poids de tous les autres neurones parents sont initialement fixés à 0. Cela nécessite également d'employer la fonction d'activation ReLU qui a la particularité d'être égale à elle-même lorsqu'on la compose par elle-même. Ainsi en reprenant la Figure 2.2 du Chapitre 2, nous pouvons illustrer cette initialisation Figure 3.1, dont nous développons ici le calcul, la préactivation de sortie de 5 après insertion de 7 est :

$$\begin{aligned}
 a_5 &= \sum_i w_{i,5} z_i + b_5 = w_{1,5} z_1 + w_{2,5} z_2 + w'_{7,5} z_7 + b_5 \\
 &= w_{1,5} z_1 + w_{2,5} z_2 + w'_{7,5} \text{ReLU}(w'_{1,7} z_1 + w'_{2,7} z_2 + w'_{3,7} z_3) + b_5 \\
 &= w_{1,5} z_1 + w_{2,5} z_2 + w'_{7,5} \text{ReLU}(z_3) + b_5 = w_{1,5} z_1 + w_{2,5} z_2 + w'_{7,5} z_3 + b_5 \\
 &= w_{1,5} z_1 + w_{2,5} z_2 + w_{3,5} z_3 + b_5.
 \end{aligned}$$

Ce qui est exactement la préactivation de sortie de 5 avant insertion de 7, la fonction représentée par le réseau est donc conservée par l'insertion de 7.

Ainsi, si l'insertion est réalisée lorsque la diminution du coût d'entraînement commence à stagner, on peut s'attendre à une "saute" continue de la fonction de coût d'entraînement à l'époque post-insertion puisque le nouveau paramètre inséré permet d'apprendre plus d'informations. Nous présentons les allures des courbes de coût d'entraînement sur des données synthétiques pour un FFNN standard à une couche cachée (à gauche) et le même FFNN subissant une expansion à la quarantième étape d'apprentissage (à droite) Figure 3.2.



(a) Courbe de coût d'entraînement d'un FFNN à une couche cachée de 10 neurones avec fonction d'activation ReLU sur des données synthétiques de classification binaire.

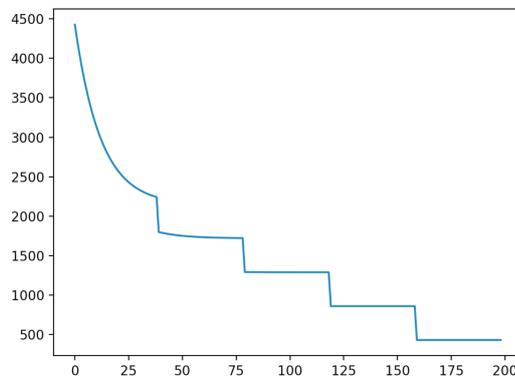
(b) Courbe de coût d'entraînement du même FFNN auquel on a ajouté un neurone initialisé de manière à garantir la continuité du coût à la quarantième époque.

**FIGURE 3.2** – Illustration de l'effet de l'expansion avec "initialisation continue" sur la fonction de coût d'un FFNN.

En répétant le processus d'insertion plusieurs fois, nous obtenons des profils de coût semblables à celui présenté Figure 3.3 dans lequel, la fonction de coût diminue par paliers et pour les dernière époques, ne semble plus diminuer que grâce aux insertions. Pour cette figure, nous avons de nouveau entraîné un FFNN standard à une couche cachée et à fonction d'activation ReLU sur des données synthétiques de classification binaire auquel nous ajoutons un nouveau paramètre toutes les 40 époques. Il s'agit de noter que les données

synthétiques ayant été générées à nouveau pour chaque expérience, les valeurs du coût ne sont pas à prendre en compte, seulement le profil des courbes.

Cette méthode d'initialisation permet ainsi la continuité de la fonction de coût et de la fonction représentée par le réseau lors de son expansion mais limite le réseau, non seulement au niveau de la fonction d'activation mais également en introduisant un schéma d'initialisation des nouveaux neurones avec un grand nombre de connexions à zéro pouvant amener à un apprentissage très ralenti voire paralysé. De plus, d'autres méthodes ne requérant pas de schéma d'initialisation aussi limitant mais empêchant tout de même l'oubli catastrophique existent, comme par exemple la réinjection de données anciennes pour éviter l'oubli, ou encore le gel de certains paramètres appris afin de garantir une performance minimale. Enfin comme mentionné au Chapitre 2, notre but n'est pas d'obtenir un modèle d'expansion avec les meilleures performances possibles, mais de comparer des modèles expansifs simples à des modèles standards. En ce sens, pour pouvoir comparer les deux types d'apprentissage, toutes choses étant égales par ailleurs, nous calquons l'initialisation des nouveaux neurones sur l'initialisation classique des réseaux standards.



**FIGURE 3.3** – Illustration de l'effet d'une insertion de neurones répétée toutes les 40 époques dans un FFNN initialement à une couche cachée à fonction d'activation ReLU sur la fonction de coût.

### 3.1.2 Initialisation standard pour les nouveaux neurones

Afin d'explorer le fonctionnement de tels réseaux comparé à celui de réseaux standards, nous initialisons les neurones insérés de la même manière que les neurones initialement présents dans le réseau dans la suite de notre travail. Ainsi, dans la suite de ce rapport, tous les neurones insérés suivront la même procédure d'initialisation que les neurones des réseaux standards auxquels nous les comparerons. Cette initialisation suivant la plupart du temps l'implémentation de la couche Linear de Pytorch, à savoir une initialisation aléatoire  $\sim \mathcal{U}(\frac{-1}{\sqrt{I}}, \frac{1}{\sqrt{I}})$  où  $I$  est la taille de l'entrée de la couche.

## 3.2 Comparaison des réseaux expansifs avec des réseaux standards à architecture donnée

En suivant ainsi le schéma d’initialisation précisé Section 3.1.2, nous comparons dans cette section les résultats de GrowNN avec des FFNN ayant le même nombre de paramètres finaux. Dans cette partie nous ne présenterons que les résultats finaux de test et nous explorerons les raisons de ce résultat afin de fournir une partie d’explication dans la Partie II.

### 3.2.1 Résultats avec des petits modèles

Les modèles discutés ici sont implémentés en PyTorch et évalués sur AGNews et MNIST introduit par LECUN et al. [116].

**AGNews** : AG News (AG’s News Corpus) est un sous-jeu de données du corpus de A. Gulli d’articles de presse construit en assemblant les titres et les descriptions des domaines d’articles des 4 classes les plus peuplées (“World”, “Sports”, “Business”, “Sci/Tech”) du corpus original. AG News contient 30,000 exemples d’entraînement et 1900 exemples de test par classe. Ce corpus peut être trouvé [ici](#).

**MNIST** : Le corpus de référence de chiffres écrits à la main MNIST consiste en 60,000 exemples d’entraînement et 10,000 exemples de test.

Nous comparons dans cette première série d’expériences présentée Table 3.1, les modèles expansifs obtenus avec l’Algorithme 1 avec un FFNN ayant exactement le même nombre final de paramètres (après toutes les insertions sur le GrowNN). Notre objectif est d’étudier si le simple fait d’ajouter progressivement des nouveaux neurones plutôt que de les apprendre tous dès le début modifie les performances du réseau, tous les autres hyper-paramètres étant par ailleurs égaux (nombre final de paramètres, fonctions d’activation, initialisation des paramètres, taille du lot, algorithme d’optimisation, ...).

Nous entraînons toutes les architectures jusqu’à atteindre environ 100% d’exactitude (ou *accuracy*) d’entraînement avec la descente de gradient stochastique (SGD). Pour éviter le surapprentissage, les modèles d’architecture désirée sont sélectionnés en fonction de leur performance sur un jeu de validation et non sur le jeu d’entraînement.

Pour chaque jeu de données, nous testons deux différentes tailles de lot, à savoir 10 et 100 pour MNIST, et 1 et 16 pour AGNews. Pas de représentation vectorielle pré-entraînée des mots n’a été utilisée pour AGNews. L’entropie croisée catégorielle (*categorical cross entropy*) est utilisée comme fonction de coût et les learning rate initiaux sont de  $10^{-4}$  sur AGNews et  $10^{-3}$  pour MNIST. De plus, nous testons deux tailles finales de modèles, avec 10 et 500 neurones cachés pour MNIST et avec 10 et 100 neurones cachés pour AGNews. Concernant les growNN, nous laissons le modèle apprendre les nouveaux neurones pour quelques époques avant d’en insérer de nouveau. Le schéma d’insertion a été le suivant :

- (MNIST, 10 neurones finaux) 1 nouveau neurone toutes les 10 époques dès l’époque 5
- (AGNews, 10 neurones finaux) 1 nouveau neurone par 10 époques dès l’époque 5

- (MNIST, 500 neurones finaux) 50 nouveaux neurones toutes les 10 époques entre les époques 10 et 60
- (AGNews, 100 final hidden neurons) 10 nouveaux neurones toutes les 10 époques depuis l'époque 5

<b>Corpus : MNIST</b>		
<b>Modèle</b>	<b>Taille du lot</b>	<b>Exactitude de test</b>
<b>10 neurones cachés</b>		
<b>GrowNN</b>	10	84%
	100	77.91%
<b>FFNN</b>	10	84.03%
	100	77.74%
<b>500 neurones cachés</b>		
<b>GrowNN</b>	10	<b>94.98%</b>
	100	<b>94.84%</b>
<b>FFNN Architecture a</b>	10	87.27%
	100	66.86%
<b>FFNN Architecture b</b>	10	84.57%
	100	64.17%
<b>FFNN Architecture C</b>	10	84.26%
	100	74.01%

(a) Résultats sur MNIST

<b>Corpus : AGNews</b>		
<b>Modèle</b>	<b>Taille du lot</b>	<b>Exactitude de test</b>
<b>10 neurones cachés</b>		
<b>GrowNN</b>	1	90.2%
	16	88.2%
<b>FFNN</b>	1	90.1%
	16	88.3%
<b>100 neurones cachés</b>		
<b>GrowNN</b>	1	90.7%
	16	89.5%
<b>FFNN Architecture a</b>	1	90.5%
	16	89.0%
<b>FFNN Architecture b</b>	1	90.4%
	16	89.0%
<b>FFNN Architecture c</b>	1	90.5%
	16	88.9%

(b) Résultats sur AGNews

TABLE 3.1 – Exactitude de Test pour différents neurones post entraînement.

La seule différence entre les deux méthodes d’entraînement est qu’un modèle gagne des neurones progressivement avant d’atteindre son architecture finale tandis que l’autre l’a déjà dès le début de l’entraînement. Enfin, pour les modèles les plus gros (500 neurones cachés pour MNIST et 100 neurones cachés pour AGNews), plusieurs architectures finales ont été testées afin de comparer des architectures de profondeurs diverses au réseau expansif pour comparer à un certain spectre d’architectures finales atteignables par notre algorithme.

- (MNIST, 500 neurones cachés finaux)
  - **architecture a** : 1 couche cachée de 500 neurones
  - **architecture b** : 2 couches cachées de 400 puis 100 neurones
  - **architecture c** : 3 couches cachées de 300 puis 150 puis 50 neurones
- (AGNews, 100 neurones cachés finaux)
  - **architecture a** : 1 couche cachée de 100 neurones
  - **architecture b** : 2 couches cachées de 80 puis 20 neurones
  - **architecture c** : 3 couches cachées de 60 puis 30 puis 10 neurones

En testant plusieurs architectures, nous vérifions que les résultats présentés Table 3.1 sont bien issus du fait de grossir lui-même.

Nous effectuons également des expériences sur RoBERTa dans les sections suivantes avant d’analyser ces résultats en fin de chapitre.

### 3.2.2 Croissance de la tête de classification de RoBERTa

Comme mentionné Chapitre 1, les Transformers ont été originellement introduits par VASWANI et al. [191] avant d’être raffinés par DEVLIN et al. [41] où les auteurs proposent un modèle de langage basé sur un transformer bidirectionnel nommé BeRT. Ce modèle a été pré-entraîné en utilisant un vaste corpus de langage non annoté via des tâches d’apprentissage non supervisé. Il s’agit d’un modèle de représentation vectorielle du langage pré-entraîné (pre-trained embeddings) à la base de nombreux modèles états de l’art pour le TALN. Les travaux menant à BeRT ont donné naissance à des travaux similaires utilisant soit d’autres tâches d’apprentissage non supervisé (voir par exemple les travaux de YANG et al. [208], DONG et al. [46] et JOSHI et al. [96]), ou alors en utilisant des modèles plus gros encore ou des corpora de pré-entraînement plus fournis (voir les travaux de LAN et al. [114], RAFFEL et al. [157], LIU et al. [130] et YANG et al. [208]). Ces modèles de langage sont ensuite adaptés via un entraînement spécifique à la tâche étudiée, appelé fine-tuning. RoBERTa, introduit par LIU et al. [129] se base sur la stratégie de pré-entraînement de langage masqué de BeRT (*language masking strategy*), mais affine le modèle original avec un choix différent de tâches et de conditions.

À l’image des modèles état de l’art en TALN contemporains, RoBERTa est un réseau complexe et très volumineux, faisant face au problème de sur-paramétrisation. Il est composé d’un corps de 355 millions de paramètres (pour RoBERTa large) stockés dans 24 couches de self-attention, avec une tête dédiée à la tâche construite par dessus. Pour la classification,

cette tête est communément appelée tête de classification et permet d’avoir en sortie le nombre de classes désiré.

Dans les expériences présentées ci-après, ce nombre de paramètres est conservé strictement, sans ajouter plus de complexité au corps du modèle. La tête de classification est composée d’un FFNN standard avec 1024 neurones cachés. Nous proposons donc de remplacer cette tête de classification par un FFNN simple et d’insérer progressivement 1024 neurones cachés pour atteindre la même architecture que le réseau standard.

Nous utilisons le jeu de données standard CoLA introduit par WARSTADT, SINGH et BOWMAN [199] pour apprendre et évaluer notre modèle expansif et le modèle RoBERTa standard. Ce jeu de données est composé de 9594 exemples d’entraînement et 1063 exemples de test et consiste à classer les phrases en deux classes : les phrases grammaticalement correctes et incorrectes en langue anglaise.

Afin d’étudier l’impact de l’expansion, nous insérons progressivement les neurones pendant la phase d’entraînement qui consiste en 10 époques. Ainsi, 1024 neurones sont insérés en 3 phases distinctes : 1/3 après la seconde époque, 1/3 après la 5e et le reste après la 7e époque d’entraînement. La taille de lot employée est 16 et les résultats sont rapportés Table 3.2. Nous utilisons ici l’Algorithme 3 pour construire une couche dense en tout point similaire à la couche de l’architecture originale. Les résultats rapportés sont la moyenne sur 10 expériences avec des *random seeds* aléatoires. La variance pour les deux types de modèles est inférieure à 0.1% et n’a donc pas été rapportée.

Modèle	Exactitude de test
GrowNN	68.2%
Standard	68%

TABLE 3.2 – Performances comparées d’un RoBERTa standard et expansif sur COLA

### 3.2.3 Croissance de la structure interne de RoBERTa

Dans cette section nous présentons les résultats obtenus en faisant grossir la partie interne (le "corps") de RoBERTa. L’idée de cette partie est d’apprendre deux modèles de même taille finale sur les tâches de classification binaire de MetaEval et ce de deux manières différentes :

- un premier modèle ayant sa taille finale dès le début et à qui toutes les tâches sont présentées les unes après les autres.
- un second modèle qui grossit entre chaque entraînement sur une nouvelle tâche.

Le corpus de référence MetaEval introduit par SILEO et MOENS [171]<sup>1</sup> comporte une collection de 101 tâches des TALN pour effectuer du méta-apprentissage et de l’apprentissage multi-tâches, incluant plus de 50 tâches de classification binaire. La taille des jeu de données est variable entre quelques centaines d’exemples jusqu’à plusieurs milliers. Nous nous contenterons dans cette partie de la trentaine de tâches de classification binaire à moins de

1. <https://github.com/sileod/metaeval>

20000 exemples d'apprentissage par souci de temps de calcul. Le détail des jeu de données utilisés est donné Chapitre 9 où nous en étudierons d'autres en plus de ceux utilisés ici.

Tâche	Modèle expansif	Modèle à taille fixe
recast_sentiment	<b>0.91</b>	0.5
persuasiveness-strength	<b>0.72</b>	0.61
sarcasm	<b>0.8</b>	0.5
binary	<b>0.74</b>	0.59
movie_rationales	<b>0.9</b>	0.5
irony	<b>0.7</b>	0.57
persuasiveness-eloquence	<b>0.79</b>	0.76
recast_verbnet	<b>0.71</b>	0.45
recast_megaveridicality	<b>0.86</b>	0.68
syntax+semantics	<b>1</b>	0.6
wic	<b>0.7</b>	0.55
persuasiveness-specificity	<b>0.79</b>	0.55
squinky-formality	<b>0.96</b>	0.88
emobank-dominance	<b>0.68</b>	0.65
emobank-arousal	<b>0.69</b>	0.67
political-media-bias	<b>0.77</b>	0.72
rotten_tomatoes	<b>0.85</b>	0.75
boolq	<b>0.71</b>	0.64
wnli	0.83	0.83
mrpc	<b>0.8</b>	0.66
hate	0.5	<b>0.57</b>
political-media-audience	0.79	<b>0.80</b>
persuasiveness-relevance	<b>0,66</b>	0,64
squinky-implicature	<b>0,76</b>	0,54
emobank-valence	<b>0,86</b>	0,63
squinky-informativeness	<b>0,92</b>	0,6
rte	<b>0,63</b>	0,5
tweet_global_warming	<b>0,83</b>	0,75

**TABLE 3.3** – Performances comparées d'un RoBERTa sans et avec expansion sur les tâches de MetaEval. La colonne *Exactitude de test* reporte la valeur d'exactitude de test du modèle sur la tâche spécifique juste après l'entraînement sur la tâche concernée. Ces valeurs sont des moyennes sur 10 runs où l'ordre de traitement des tâches a été tiré aléatoirement à chaque fois. La variance moyenne est de 0.03.

Il s'agit également de noter que les tâches ayant des objectifs différents les unes des autres, des têtes de classification spécifiques à la tâche courante seront utilisées. Le nombre de neurones ajoutés entre chaque tâche est directement issu du travail de KAWAGUCHI et HUANG [104] dans lequel les auteurs prouvent théoriquement que la descente de gradient

peut trouver un minimum global si le nombre de paramètres augmente linéairement en fonction du nombre d'exemples d'entraînement. En utilisant ce résultat (qui est de plus optimal), nous choisissons d'ajouter  $\frac{N_{\text{task}}}{6}$  nouveaux neurones dans les couches cachées des réseaux denses de sortie de chacune des 6 couches de self-attention lors du passage à une nouvelle tâche *task*, où  $N_{\text{task}}$  est le nombre d'exemples d'entraînement de *task*. Le nombre final de paramètres ajouté après l'entraînement sur  $k$  tâches sera donc  $\sum_{1 \leq i \leq k} N_i$ , où  $N_i$  est le nombre d'exemples d'entraînement de la tâche  $i$ . Pour le modèle ne subissant pas d'expansion pendant l'entraînement, nous ajoutons la somme de tous ces paramètres avant l'entraînement qui consistera à présenter les tâches les unes après les autres sur le même modèle (en changeant simplement la tête de classification pour chaque tâche). Tous les hyper-paramètres utilisés sont identiques pour les deux méthodes. Nous reportons les résultats Table 3.3.

En répétant les expériences avec un ordre aléatoire des tâches à chaque fois, les modèles ainsi entraînés déploient une exactitude de test moyenne de 0.5 pour le modèle de taille fixe. Cela revient sur ces tâches de classification binaire à une forme de choix aléatoire de la part du modèle qui est donc soumis à un fort phénomène d'oubli catastrophique en moyenne. Pour le modèle à architecture expansive, l'exactitude moyenne de test est de 0.54, ce qui est à peine mieux. Nous pourrions développer des heuristiques complexes ou des astuces algorithmiques afin d'obtenir des modèles à experts semblables aux travaux de SHAZEER et al. [169], mais notre but ici et pour le moment n'est pas d'optimiser au mieux des modèles en quête de performances état de l'art mais bien de mettre en valeur et de comprendre les mécanismes sous-jacents à l'expansion d'un réseau de neurones.

### 3.2.4 Analyse des résultats

Les résultats obtenus Table 3.1 sont obtenus avec des réseaux peu profonds et sur des tâches standards mais relativement simples. Nous pouvons observer que les réseaux expansifs obtiennent de meilleures performances et que la différence avec les réseaux standards s'accroît lorsque le nombre de neurones insérés augmente.

Un autre résultat intéressant que nous pouvons observer est que le fait de grossir ne semble pas avoir d'impact négatif sur l'exactitude des réseaux complexes Table 3.2. Nous pouvons noter que la tête de classification de RoBERTa ne représente qu'environ 0.6% du nombre total de paramètres du modèle, ce qui peut expliquer que les performances entre les deux types de modèle sont similaires.

De plus, dans le cas de très petits réseaux (avec seulement 10 neurones cachés dans la Table 3.1), les performances sont extrêmement similaires peu importe le mode d'entraînement, ce qui est très certainement dû à la très faible capacité du réseau qui ne peut pas apprendre plus à cause de sa taille limitante.

Nous pouvons également observer deux comportements différents lorsqu'un nombre plus important de neurones est inséré. D'un côté, lorsque nous comparons les deux types d'apprentissage sur MNIST, les growNN tendent à avoir de meilleures performances, tandis que sur AGNews, les modèles ont tous des performances similaires (qu'ils soient expansifs ou non, et avec 10 neurones cachés ou 100 neurones cachés). De même, avec l'entraînement

d'une structure complexe sur CoLA, aucune différence significative des performances n'est observable entre les deux méthodes d'entraînement. Cela est dû selon nous au fait que le nombre de paramètres inséré est trop faible pour avoir un impact réel sur l'exactitude puisque la plupart de l'information est apprise dans les embeddings d'entrée (pré-entraînés et simplement spécialisés sur CoLA ou non pré-entraînés sur AGNews).

En revanche, la Table 3.3 montre que lorsque le nombre de neurones introduits est suffisant pour représenter l'information d'entrée et donc assurer "l'entraînabilité" (*trainability*) du réseau, le réseau expansif a en moyenne de meilleures performances qu'un réseau à taille fixe, même si les expériences reportées dans cette Table ne permettent pas une comparaison exacte puisque le nombre de paramètres du réseau expansif est inférieur à celui du réseau à taille fixe. Elle permet cependant de mettre en valeur une disparité de performances entre les deux types de réseau, disparité que nous essaierons d'expliquer, au moins en partie, dans la Partie II, notamment dans le Chapitre 6 où nous compléterons les Tables 3.1 et 3.2 avec une métrique que nous expliquerons et motiverons Chapitres 4 et 5.

Les résultats obtenus dans cette Partie montrent donc qu'un algorithme d'expansion relativement naïf est compétitif par rapport aux algorithmes standards. Il s'agira enfin de noter que la mesure de performance est calculée sur un sous-ensemble de test des données et que de meilleures performances sur ce genre de jeu de données de test n'est pas forcément indicateur d'une meilleure généralisation comme nous en discuterons Partie III.



## **Deuxième partie**

### **Étude de la surface de la fonction de coût des réseaux de neurones**



---

## CHAPITRE 4 : ÉTUDE DE L'ÉTAT DE L'ART : LES SURFACES DES FONCTIONS DE COÛT

*La liaison fortuite des atomes est l'origine  
de tout ce qui est.*

---

– Démocrite

---

4.1	Les Surfaces des fonctions de coût des ANNs . . . . .	56
4.2	Planéité et Performances . . . . .	57

---

Dans ce chapitre, nous introduisons la notion de surface de la fonction de coût (ou surface de *loss*) à proximité du minimum trouvé par les ANNs grâce à une descente de gradient dans l'espace des paramètres. Plus particulièrement nous nous intéressons à la quantification du volume pris par ce minimum dans cet espace, la *planéité*. Dans un premier temps, nous allons définir dans ce chapitre les outils nécessaires à son analyse, ainsi que les travaux majeurs nous permettant de relier cette mesure aux propriétés des réseaux de neurones artificiels. Dans les chapitres suivants de cette partie, nous analyserons la planéité dans le contexte des réseaux expansifs et montrerons que le fait de progressivement introduire de nouveaux neurones dans le réseau mène à une exploration particulière de l'espace des paramètres et à des minima plus plats.

## 4.1 Les Surfaces des fonctions de coût des ANNs

Minimiser la fonction de coût d'un ANN est un problème d'optimisation hautement non convexe, cependant les algorithmes relativement simples de descente de gradient mis en pratique par la communauté scientifique semblent permettre d'apprendre des réseaux de neurones sans se heurter à des minima largement sous-optimaux. Toutefois les résultats théoriques sur ce sujet montrent la difficulté d'apprendre un ANN puisqu'il peut y avoir un nombre exponentiel de minima locaux distincts comme montré par AUER, HERBSTER et WARMUTH [4] et SAFRAN et SHAMIR [166]. De plus, il a été prouvé dès 2002 que l'entraînement d'un réseau de neurones à un seul neurone avec une certaine variété de fonctions d'activation est en fait un problème NP-difficile (voir les résultats de SIMA [173]).

Les résultats expérimentaux et théoriques divergent donc puisqu'il a été observé par GOODFELLOW, VINYALS et SAXE [65] et DAUPHIN et al. [40] que l'entraînement d'un FFNN (parcimonieux à l'image d'un CNN ou dense) ne rencontre pas de minimum local sous optimal. La raison reste toutefois une question ouverte et GOODFELLOW, VINYALS et SAXE [65] admettent que cela peut être simplement dû au fait que de tels réseaux sont construits pour être faciles à apprendre.

Pour certaines classes de réseaux, des approches théoriques existent pour atteindre le minimum global efficacement, mais ces méthodes sont impossibles à mettre en place dans les faits comme expliqué par JANZAMIN, SEDGHI et ANANDKUMAR [92], HAEFFELE et VIDAL [73] et SOLTANOLKOTABI [176] car elles requièrent des informations additionnelles complexes comme une connaissance de la distribution des données et une initialisation des poids adaptée par exemple ; ou car elles modifient la structure du réseau et son objectif (voir l'article de GAUTIER, NGUYEN et HEIN [60]). KAWAGUCHI [101] a par ailleurs montré, que malgré la non convexité de la fonction de coût et donc la complexité d'apprentissage d'un modèle, tout minimum local d'un réseau profond est un minimum global, ce qui limite cette complexité. De même, KAWAGUCHI et KAEHLING [102] montrent qu'en ajoutant un neurone spécial par neurone de sortie, tous les minima locaux sont supprimés et il ne reste que le minimum global. Quoiqu'extrêmement intéressant, ce résultat est également impossible à mettre en pratique puisque la transformation proposée aplanit en fait l'espace de la fonction de coût et rend la descente de gradient très difficile.

Dans le cas des réseaux sur-paramétrés, de nombreux résultats traitent du caractère atteignable d'un minimum global. Les minima locaux sont ainsi montrés comme souvent proches du minimum global par CHOROMANSKA et al. [33]. De même, SAFRAN et SHAMIR [166] montrent que pour un réseau à une couche cachée tel que le nombre de paramètres est supérieur au nombre d'exemples d'apprentissage, il est possible de générer une initialisation permettant d'atteindre l'optimum global via une descente de gradient. Dans ce cas en réalité, il est avancé par VIET DANG [192] que presque tous les points critiques sont globalement optimaux. Ainsi, pour ce qui est désigné comme "réseaux larges", presque tout minimum local est globalement optimal car les données d'apprentissage peuvent être linéairement séparées dans la large couche. De plus, pour certains algorithmes de descente de gradient, la proximité entre minima locaux et global est avérée, à l'image du SGD pour lequel XING et al. [206] montrent que les régions de loss dans lesquelles il évolue sont semblables à des vallées.

Enfin, il a été théoriquement prouvé par KAWAGUCHI et HUANG [104] que si le nombre de paramètres évolue de manière linéaire en fonction du nombre d'exemples d'apprentissage, le modèle est "entraînable", c'est-à-dire qu'il peut converger vers un minimum global. Ce résultat dresse également une borne inférieure du nombre de paramètres minimum nécessaires pour converger, explique en partie la réussite des réseaux modernes et indique également une marche à suivre dans le cas de données arrivant en flux dans le contexte de GrownNN, permettant d'augmenter la taille du réseau à un rythme contrôlé et raisonnable. Il est à noter pour le lecteur intéressé que la plupart des travaux récents sur les propriétés globales de ces surfaces de coût sont discutés par SUN et al. [180], dans lequel les résultats théoriques et empiriques sont étudiés.

Par delà ces considérations de convergence pure, la capacité de généraliser à des données inconnues est souvent liée à "l'apparence" des minima trouvés pendant la phase d'apprentissage et en particulier à leur planéité. Ainsi des travaux empiriques, CHAUDHARI et al. [23] et KESKAR et al. [105] montrent par leurs résultats que des minima aigus (appelés minima *sharp* en anglais, en opposition à *flat*) mènent à des performances de généralisation dégradées par rapport à des minima plats. Dans un travail récent, WEN et al. [202] montrent de manière notable qu'en lissant les minima aigus, les capacités à généraliser du réseau s'améliorent. Le principe de ce travail est de perturber plusieurs copies du même réseau en injectant du bruit, puis d'en faire une moyenne. Une idée équivalente proposée par SINHA, GARG et LAROCHELLE [175] lisse les activations plutôt qu'introduire du bruit et arrive aux mêmes conclusions.

Enfin, en traquant la planéité et en optimisant de manière jointe fonction de coût et planéité, FORET et al. [58] améliorent nettement les performances des DNNs. Cette procédure en particulier, appelée *Sharpness-Aware Minimization* (SAM) cherche des paramètres dans un voisinage au coût uniformément bas (et donc un minimum plat) et présente des résultats empiriques montrant une nette amélioration des performances du modèle sur une large variété de jeux de données.

Cependant, il s'agit d'étudier avec attention les mesures de planéité utilisées, puisqu'il a été prouvé par DINH et al. [44] qu'il est possible de modifier un réseau afin d'obtenir une mesure de planéité aussi élevée ou faible que voulue via du rescaling. Nous proposons donc dans la section suivante de rapporter des mesures répondant à ce problème et leurs résultats.

## 4.2 Planéité et Performances

Les travaux pionniers de KESKAR et al. [105] et HOCHREITER et SCHMIDHUBER [82] furent les premiers à tenter de mesurer la planéité des minima d'ANNs. Puisque des minima plats sont vus comme robustes aux perturbations des paramètres du réseau et que NOVAK et al. [144] relie la généralisation à la sensibilité du réseau aux perturbations des entrées, une mesure précise semble très intéressante pour évaluer les capacités d'un réseau.

Afin d'évaluer correctement la planéité d'un minima, des mesures invariantes au problème de redimensionnement souligné par DINH et al. [44] ont vu le jour, notamment dans les travaux de PETZKA et al. [151] et RANGAMANI et al. [158] et peuvent donc être utilisées

pour évaluer la différence de planéité des minima entre modèles de même taille, en comblant l'absence d'invariance par reparamétrisation.

À l'image d'une fonction  $C^\infty$  dans  $\mathbb{R}$  dont la planéité peut être simplement calculée par la dérivée seconde, la mesure naïve de la planéité de la fonction de coût dans l'espace des paramètres est sa Hessienne. Seulement, comme pointé par DINH et al. [44], ce genre de mesures simples peut être modifiée via des rescalings du réseau afin d'avoir une valeur de planéité choisie, n'apportant donc aucune information sur la qualité du minimum atteint.

La mesure développée par RANGAMANI et al. [158] est également basée sur la Hessienne du coût mais est elle invariante au rescaling car les auteurs utilisent une mesure alternative pour quantifier la planéité des coûts empiriques, basée sur la définition d'une variété quotient des paramètres ce qui aboutit à une mesure invariante aux reparamétrisations. Cette mesure est utilisée par les auteurs pour montrer que les minima obtenus avec de plus grands lots sont plus aigus qu'avec de plus petits lots. Cette variété engendre une étude sur un espace Riemannien, type d'études qui généralement est concentrée dans le cadre des ANNs autour de la normalisation par lot (*batch normalisation*, voir les travaux de CHO et LEE [32] et HOFFER et al. [85]). Puisque les couches de batch normalisation sont invariantes à l'échelle des transformations induites par les couches linéaires les précédant, une approche standard est de restreindre les poids de ces couches linéaires à des variétés spécifiques, par exemple proposée par HUANG et al. [87]. Ici, similairement à l'utilisation comme mesure de planéité de la norme spectrale de la Hessienne dans l'espace euclidien standard (appelée Hessienne Euclidienne par les auteurs), les auteurs montrent qu'il est possible d'utiliser le développement limité de Taylor des fonctions réelles sur une variété pour obtenir une mesure de planéité analogue en utilisant la norme spectrale de la Hessienne dans l'espace Riemannien (Hessienne Riemannienne) de la variété quotient.

Cette mesure peut être utilisée pour comparer des réseaux de même taille entre eux. En effet, il s'agit de comparer des mesures relatives à l'espace des paramètres des réseaux et il faut donc que celui-ci soit fixé. Il est donc nécessaire de comparer ce qui est comparable, à savoir deux architectures finalement identiques sur un jeu de données d'entraînement identique, ce qui nous permet d'être dans le cadre défini à la Section 1.2.2 du Chapitre 1. Il s'agit de noter qu'il a été montré par ZHANG et al. [214] que le volume des bassins d'attraction des minima est corrélé en moyenne avec leur planéité. Il est donc intuitivement possible d'assimiler la planéité d'un minimum au volume qu'il occupe dans l'espace des paramètres, si son calcul ne dépend pas d'un rescaling arbitraire, par exemple en utilisant la norme spectrale proposée par RANGAMANI et al. [158].

Enfin, puisque les métriques basées sur des mesures de planéité performant mieux que la plupart des autres mesures complexes pour évaluer la généralisation comme montré par JIANG et al. [94] et que le fait de croître progressivement un réseau change intrinsèquement la manière d'explorer l'espace des paramètres, il est naturel d'étudier la planéité des minima atteints par les GrowNN et de la comparer à celle des minima atteints de manière standard. Notre but dans le Chapitre 6 sera d'explorer si un réseau de neurones expansif pendant sa phase d'apprentissage atteint des minima possédant des surfaces de coût avec un meilleur comportement qu'en entraînant un FFNN standard, les deux réseaux ayant finalement le même nombre de paramètres et ce en utilisant la mesure développée dans [158] que nous avons présentée ici.

# CHAPITRE 5 : UNE MANIÈRE PARTICULIÈRE D'EXPLORER L'ESPACE DES PARAMÈTRES : LES RÉSEAUX DE NEURONES EXPANSIFS

*Chaque découverte scientifique est  
passionnante parce qu'elle ouvre un  
univers de questions.*

– Boris Cyrulnik

5.1	Notations et Hypothèses . . . . .	61
5.2	Expression Théorique : Comparaison des volumes des minima . . .	62

La relation entre la planéité de la surface de coût et la généralisation du modèle est toujours sujette à débat dans la littérature. Nous ne prétendons pas donner une réponse définitive à ce problème ouvert, mais nous apporterons cependant de nouvelles preuves expérimentales chapitre 6 qui soutiennent cette relation présumée. Dans ce chapitre, nous justifions théoriquement que dans le contexte général des FFNN, les GrowNN atteignent des minima plus plat en moyenne que les réseaux standards.

Intuitivement, la preuve que nous détaillons ci-après compare deux méthodes d'entraînement :

- lorsque nous considérons l'espace de recherche complet en termes de paramètres, l'algorithme de descente de gradient (par exemple le SGD) converge vers un optimum qui dépend principalement de la distance aux paramètres initiaux et non à la planéité de cet optimum.
- en considérant un de ces paramètres bloqué (donc avant l'étape d'ajout de ce paramètre au réseau), l'algorithme est contraint à rester sur un hyperplan  $H$  de l'espace de

paramètres complet avant de converger dans un optimum de cet espace qui intersecte  $H$  ou un point selle de cet espace complet qui est un minimum dans  $H$ . Nous ignorons dans la suite ces points selles puisqu'ils ne changent pas les conclusions de la preuve comme nous le montrerons.

Le coeur de la preuve réside dans le fait qu'il y a en moyenne plus de minima plats sur  $H$  que dans l'espace total (proportionnellement) car un minimum plat occupe un volume plus important qu'un minimum aigu dans l'espace total et l'hyperplan  $H$  est donc plus susceptible d'intersecter un minimum plat qu'un minimum aigu. Détaillons maintenant cette preuve.

## 5.1 Notations et Hypothèses

Soit  $R_e(\theta) \in \mathbb{R}^+$  le coût ou risque empirique (ou fonction de loss d'entraînement) à minimiser, avec  $\theta \in \mathbb{R}^d$  les paramètres du modèle. Supposons comme hypothèse préliminaire que  $R_e(\theta)$  a  $N$  minima locaux :  $(\theta_1^*, \dots, \theta_N^*)$  uniformément répartis sur le domaine de  $\theta$ . Chacun de ces minima locaux est associé avec un volume local  $V(\theta_i^*)$ , appelé **bassin d'attraction** (*basin of attraction*), qui est défini comme le volume autour de  $\theta_i^*$  tel que si un algorithme de descente de gradient entre dans ce volume, il converge presque sûrement vers  $\theta_i^*$  voir l'article de ERHAN et al. [52]. Ainsi, il est intéressant de noter que ces volumes dépendent du learning rate de l'algorithme (c'est à dire la distance parcourue dans l'espace des paramètres à chaque étape d'apprentissage) mais surtout que la courbe de  $R_e(\theta)$  dans l'espace des paramètres  $\theta$  est entièrement pré-définie par le couple  $\alpha$ , jeu de données  $d$  où  $\alpha$  est l'architecture du réseau choisi, ce qui rejoint notre analyse Section 1.2.2. Notons  $(O, \vartheta_1, \dots, \vartheta_d)$  le système de coordonnées cartésiennes dans l'espace complet de paramètres de  $\mathbb{R}^d$ .

Nous faisons alors les hypothèses suivantes :

- **Hyp1** : À partir de paramètres  $\theta^0 \in \mathbb{R}^d$  issus d'une initialisation aléatoire qui n'est pas déjà dans un bassin d'attraction quelconque, nous supposons que la descente de gradient converge vers l'optimum local le plus proche :

$$\hat{\theta} = \arg \min_i \|\theta_i^* - \theta^0\|^2$$

Cette hypothèse est relativement forte mais il est possible de la remplacer par une version plus faible tout en gardant les mêmes résultats, à savoir que la descente de gradient converge vers un optimum  $\theta^*$  avec une probabilité dépendant de la distance à cet optimum. Toutefois, l'Hypothèse 1 est simple à interpréter visuellement et permet au raisonnement suivant d'être simple à suivre.

- **Hyp2** :  $\theta^0$  n'est pas déjà dans un bassin d'attraction

Cette hypothèse n'est pas strictement nécessaire puisque la situation est simple dès lors que nous nous situons déjà dans un bassin d'attraction et que cela ne change donc pas la conclusion principale de la suite. Cependant cette hypothèse permet d'omettre certains cas limites rares et de simplifier ainsi les calculs des volumes attendus.

À l'aide de ces hypothèses, nous allons donc comparer les volumes attendus  $V(\hat{\theta})$  des minima locaux vers lesquels converge la descente de gradient, respectivement avec une étape d'expansion et sans.

## 5.2 Expression Théorique : Comparaison des volumes des minima

Commençons par la descente de gradient standard dans l'espace complet de dimension  $d$ . Puisque  $\theta^0$  et tous les  $\theta_i^*$  sont uniformément répartis sur cet espace et puisque l'hypothèse 1 ne dépend que de la distance entre  $\theta^0$  et  $\theta_i^*$ , la descente de gradient convergera aléatoirement vers un des minima et :

$$E[V(\hat{\theta})] = \sum_i p(\hat{\theta} = \theta_i^*) V(\theta_i^*) = \frac{1}{N} \sum_i V(\theta_i^*) \quad (5.1)$$

Étudions désormais le cas du modèle expansif. Il est plus facile de commencer dans l'espace final (post-expansion) de dimension  $d$  et d'en extraire le sous-espace de dimension  $d - 1$  qui existait avant que le modèle ne croisse : nous supposons pour la clarté de la preuve qu'un seul paramètre est ajouté lors de l'expansion.

Comme remarqué lors de la section précédente, l'espace de  $\mathbb{R}^d$  est exactement le même que dans le cas standard avec les mêmes optima locaux ( $\theta_i^*$ ). En partant de cet espace à  $d$  dimensions, si nous "supprimons" une dimension, par exemple  $\vartheta_1$ , nous pouvons visualiser la surface de coût dans l'espace avant expansion. "Supprimer"  $\vartheta_1$  est équivalent à fixer la première coordonnée de tout point à 0 :  $\theta_{i,1}^* = 0$ , ce qui définit un hyperplan  $H$ , illustré en jaune dans la Figure 5.1, hyperplan sur lequel se déroule l'entraînement avant l'ajout de paramètre.

Afin de calculer la probabilité que  $H$  intersecte le volume autour d'un optimum local ( $\theta_i^*$ ) de  $\mathbb{R}^d$ , il est plus simple d'approximer le volume  $V(\theta_i^*)$  avec l'hypercube minimal aligné aux axes enfermant ce volume :

$$[\theta_{i,1}^* - \tau_{i,1} : \theta_{i,1}^* + \tau'_{i,1}] \times \cdots \times [\theta_{i,d}^* - \tau_{i,d} : \theta_{i,d}^* + \tau'_{i,d}]$$

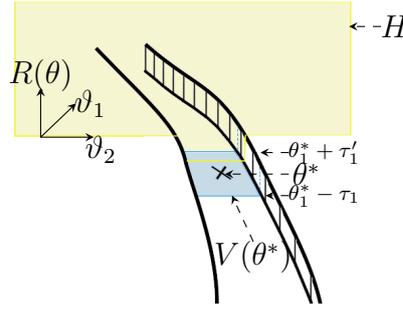
comme illustré en bleu dans la Figure 5.1. On peut ensuite relier le volume de cet hypercube au volume cherché comme suit :

$$V(\theta_i^*) \simeq \prod_{j=1}^d (\tau'_{i,j} - \tau_{i,j})$$

Nous supposons enfin **Hyp3** :  $\tau'_{i,1} - \tau_{i,1}$  est conditionnellement indépendant de  $(\tau'_{i,j} - \tau_{i,j})_{2 \leq j \leq d}$  étant donné  $V(\theta_i^*)$ .

Cette dernière hypothèse est motivée par le fait qu'il n'y a pas de direction privilégiée pour les hypercubes contenant les bassins d'attraction ; en moyenne la largeur du bassin d'attraction selon la première dimension est la même que selon toute autre dimension, donc

$$E[V(\theta_i^*)] \simeq \prod_{j=1}^d E[\tau'_{i,j} - \tau_{i,j}] \simeq E[\tau'_{i,1} - \tau_{i,1}]^d \quad (5.2)$$



**FIGURE 5.1** – Illustration de la surface de coût dans l'espace  $\mathbb{R}^d$  avec  $d = 2$  autour de l'optimum local  $\theta^*$ . L'axe vertical représente la valeur du risque empirique du modèle pour tous paramètres dans  $\mathbb{R}^d$ . Nous illustrons un scénario possible, communément admis pour les DNN où les minima locaux sont connectés selon des formes semblables à des vallées et  $\theta^*$  est dans une telle vallée. L'hyperplan jaune est l'espace de recherche restreint avant l'étape d'expansion et la surface bleue est le bassin d'attraction de  $\theta^*$ .

Soit  $p(H_i)$  la probabilité que  $H$  intersecte le bassin d'attraction  $V(\theta_i^*)$ . Puisque la localisation des optima dans l'espace  $\mathbb{R}^d$  est distribuée uniformément, et puisque  $H$  a pour équation directrice  $\vartheta_1 = 0$ , la probabilité que  $H$  intersecte  $V(\theta_i^*)$  est proportionnel à  $\tau'_{i,1} - \tau_{i,1}$ .

Ainsi,

$$p(H_i) = \frac{1}{V_1}(\tau'_{i,1} - \tau_{i,1}) \quad (5.3)$$

où  $V_1$  est la largeur du domaine selon la première dimension.

Nous avons pour le moment deux équations qui expriment intuitivement que  $H$  a plus de chance d'intersecter un bassin d'attraction au volume plus grand.

D'autre part, l'optimisation dans le growNN procède comme suit :

1.  $\theta^0$  est uniformément échantillonné sur  $H$
2. La descente de gradient converge vers  $\hat{\theta}^H \in H$  qui est dans le bassin d'attraction de soit un des minima  $\theta_i^*$  en dimension supérieure, soit un des  $S$  points selles  $\{\theta_1^S, \dots, \theta_S^S\}$  sur  $H$
3. Un paramètre est ajouté au modèle
4. La descente de gradient procède ensuite dans l'espace complet de  $\mathbb{R}^d$ , en commençant de  $\hat{\theta}^H$  et en convergeant vers  $\hat{\theta} \in \{\theta_1^*, \dots, \theta_N^*\}$

Le volume moyen final est donc :

$$E[V(\hat{\theta})] = \sum_{i=1}^N V(\theta_i^*) p(\hat{\theta} = \theta_i^*)$$

$$p(\hat{\theta} = \theta_i^*) = \sum_{j=1}^N p(\hat{\theta} = \theta_i^*, \hat{\theta}^H \sim \theta_j^*) + \sum_{j=1}^S p(\hat{\theta} = \theta_i^*, \hat{\theta}^H = \theta_j^S)$$

où  $\hat{\theta}^H \sim \theta_j^*$  signifie que  $\hat{\theta}^H$  a convergé sur  $H$  vers un point dans le bassin d'attraction de  $\theta_j^*$ , ce qui est possible uniquement si  $H$  intersecte  $V(\theta_j^*)$ .

$$\begin{aligned} p(\hat{\theta} = \theta_i^*, \hat{\theta}^H = \theta_j^S) &= p(\hat{\theta} = \theta_i^* | \hat{\theta}^H = \theta_j^S) p(\hat{\theta}^H = \theta_j^S) \\ &= \frac{1}{N} \times \frac{1}{(S + K)} \end{aligned}$$

avec  $K = \sum_i p(H_i)$  le nombre d'optima réels dont le bassin d'attraction intersecte  $H$ .

Ainsi,

$$\sum_{j=1}^S p(\hat{\theta} = \theta_i^*, \hat{\theta}^H = \theta_j^S) = \frac{S}{N(S + K)}$$

L'autre terme donne :

$$p(\hat{\theta} = \theta_i^*, \hat{\theta}^H \sim \theta_j^*) = p(\hat{\theta} = \theta_i^* | \hat{\theta}^H \sim \theta_j^*) p(\hat{\theta}^H \sim \theta_j^*)$$

La définition d'un bassin d'attraction donne :

$$p(\hat{\theta} = \theta_i^* | \hat{\theta}^H \sim \theta_i^*) = 1$$

et

$$p(\hat{\theta} = \theta_i^* | \hat{\theta}^H \sim \theta_{j \neq i}^*) = 0$$

Donc

$$p(\hat{\theta} = \theta_i^*) = p(\hat{\theta}^H \sim \theta_i^*) + \frac{S}{N(S + K)}$$

Nous pouvons décomposer le terme restant selon si  $H$  intersecte  $V(\theta_i^*)$  :

$$p(\hat{\theta}^H \sim \theta_i^*) = p(\hat{\theta}^H \sim \theta_i^* | H_i) p(H_i) + p(\hat{\theta}^H \sim \theta_i^* | \bar{H}_i) (1 - p(H_i))$$

Le second terme étant nul si  $H$  n'intersecte pas  $V(\theta_i^*)$ , donc :

$$p(\hat{\theta}^H \sim \theta_i^*) = p(\hat{\theta}^H \sim \theta_i^* | H_i) p(H_i)$$

À cause de la distribution uniforme des points critiques, la probabilité que la descente de gradient converge tout d'abord vers un des points critiques positionnés sur  $H$  est uniforme, donc :

$$p(\hat{\theta}^H \sim \theta_i^* | H_i) = \frac{1}{S + K}$$

Ainsi

$$p(\hat{\theta} = \theta_i^*) = \frac{S + Np(H_i)}{N(S + \sum_j p(H_j))}$$

Et donc finalement :

$$E[V(\hat{\theta})] = \frac{1}{N} \sum_{i=1}^N V(\theta_i^*) \frac{S + Np(H_i)}{S + \sum_j p(H_j)}$$

Pour les modèles expansifs, le volume moyen du minimum atteint est la moyenne pondérée de tous les minima existants. Il est ainsi simple d'analyser quel minimum a le poids le plus important :

$$\frac{S + Np(H_i)}{S + \sum_j p(H_j)} > 1 \Leftrightarrow p(H_i) > E[p(H_i)]$$

D'après les équations Eq-5.3 et Eq-5.2, nous avons  $E[p(H_i)] = \frac{1}{V_1} E[V(\theta_i^*)]^{\frac{1}{d}}$ , donc nous avons en réalité partitionné l'ensemble des optima  $\theta_i^*$  en deux sous-ensembles disjoints : les plus grands optima, avec :  $p(H_i) > \frac{1}{V_1} E[V(\theta_i^*)]^{\frac{1}{d}}$  et les plus petits, avec :  $p(H_i) \leq \frac{1}{V_1} E[V(\theta_i^*)]^{\frac{1}{d}}$ .

En appliquant l'équation Eq- 5.2 à chacune de ces sous-ensembles, nous pouvons montrer qu'en moyenne le premier sous-ensemble correspond aux optima avec un volume supérieur à la moyenne de tous les volumes. En moyenne, les minima ayant un poids plus important sont donc également les minima avec le volume le plus grand. Le volume final attendu est donc plus élevé avec des modèles expansifs qu'avec des modèles standards qui suivent l'équation Eq- 5.1.

Afin de finaliser le lien entre volume et planéité, nous supposons finalement qu'une mesure insensible au rescaling telle que celle proposée par RANGAMANI et al. [158], est corrélée au volume  $V(\theta_i^*)$ . Cette relation peut être sujet à débat, comme dans le travail de ZHANG et al. [214], mais même si des variations substantielles peuvent être observées, ce genre de mesure de planéité est corrélée en moyenne avec le volume.

Ainsi, ce résultat montre qu'en moyenne un GrowNN converge vers un minimum local plus plat qu'un réseau de neurones standard. Il s'agit de noter que ceci ne dépend pas de l'algorithme d'expansion, simplement du fait de croître en lui-même.

Les hypothèses faites en cours de démonstration sont à nos yeux valables car :

- **Hyp3** : En contraignant l'algorithme d'expansion selon une éventuelle direction privilégiée permettrait de garder la preuve correcte, réduisant néanmoins sa généralité à de tels algorithmes plus complexes.
- Nous avons également supposé qu'il existait un hypercube aligné sur les axes pour chaque minimum tel qu'un algorithme de descente de gradient bien calibrée converge avec probabilité 1 vers cet optimum s'il entre dans cet hypercube. Il s'agit du plus gros hypercube circonscrit au bassin d'attraction. Cette approximation n'a que deux effets : premièrement il faut un plus grand nombre d'étapes pour que la descente de gradient converge dans cet hypercube que dans le réel bassin d'attraction, enfin la probabilité d'intersection  $p(H)$  est en réalité un peu plus élevée que ce qui est calculé, mais les conclusions restent inchangées.



---

## CHAPITRE 6 : LES RÉSEAUX DE NEURONES EXPANSIFS CONVERGENT VERS DES MINIMA PLATS

*La science consiste à passer d'un étonnement à un autre.*

---

– Aristote

---

6.1	Détails des conditions expérimentales . . . . .	68
6.2	Résultats et Analyse . . . . .	69

---

Dans ce Chapitre, nous complétons les résultats des Chapitres 3 et 5 en étendant les expériences du premier dans le cadre de l'étude de la planéité des ANNs et en montrant expérimentalement que la preuve théorique du second se vérifie dans les faits. Nous présentons ainsi dans ce chapitre des résultats expérimentaux tendant à montrer que les GrowNN convergent vers des minima plus plats que leur contrepartie standard.

## 6.1 Détails des conditions expérimentales

Nous reprenons ici les hypothèses expérimentales du Chapitre 3 et nous concentrons sur les cas de l'expansion de la tête de classification de RoBERTa et de l'expansion d'un réseau linéaire (expériences sur MNIST et CoLA). Le but de ce Chapitre est de vérifier si l'intuition formalisée et prouvée au Chapitre 5, à savoir que les GrowNN convergent vers des minima plus plats qu'un FFNN standard, se vérifie dans les faits. Nous compléterons enfin ces résultats au Chapitre 9 en utilisant une mesure de généralisation développée au Chapitre 8.

Pour rappel, les neurones insérés suivent à leur initialisation une distribution uniforme entre  $-\frac{1}{\sqrt{I}}$  et  $\frac{1}{\sqrt{I}}$ , où  $I$  est la taille de l'entrée pour suivre le schéma d'initialisation implémenté dans les couches Linear de Pytorch<sup>1</sup>. Les expériences menées suivent l'Algorithme 3 afin de pouvoir reconstruire des couches Linear standards en fin d'expansion et donc d'atteindre exactement le même nombre de paramètres finaux que les FFNN auxquels nous comparons les GrowNN pour pouvoir utiliser la mesure de planéité de RANGAMANI et al. [158], afin d'être dans le même espace.

Ces expériences ont été menées sur un GPU TITAN X, le temps de calcul le plus important étant d'évaluer la planéité en utilisant la mesure développée par RANGAMANI et al. [158] via le code disponible sur [github](#), ce qui a pris plusieurs heures pour chaque expérience (le temps exact dépendant du jeu de données choisi).

Nous présentons les résultats dans la section suivante.

---

1. voir Linear : <https://pytorch.org/docs/stable/generated/torch.nn.Linear.html>

## 6.2 Résultats et Analyse

Corpus : MNIST			
Modèle	Taille du lot	Exactitude de test	Norme Spectrale
<b>10 neurones cachés</b>			
GrowNN	10	84%	<b>2.23</b>
	100	77.91%	<b>24.29</b>
FFNN	10	84.03%	38.01
	100	77.74%	68.39
<b>500 neurones cachés</b>			
GrowNN	10	<b>96.98%</b>	<b>4.45</b>
	100	<b>94.84%</b>	<b>179.01</b>
FFNN architecture a	10	87.27%	1030.07
	100	66.86%	12111.28
FFNN architecture b	10	84.57%	820.54
	100	64.17%	21517.66
FFNN architecture c	10	84.26%	1253.72
	100	74.01%	12893.59

(a) Résultats sur le corpus MNIST

Corpus : AGNews			
Modèle	Taille du lot	Exactitude de test	Norme Spectrale
<b>10 neurones cachés</b>			
GrowNN	1	90.2%	<b><math>9.83 \cdot 10^{-5}</math></b>
	16	88.2%	<b>3.22</b>
FFNN	1	90.1%	$7.91 \cdot 10^{-4}$
	16	88.3%	20.83
<b>100 neurones cachés</b>			
GrowNN	1	90.7%	<b>0.57</b>
	16	89.2%	<b>31.31</b>
FFNN architecture a	1	90.5%	3.43
	16	89.0%	69.53
FFNN architecture b	1	90.4%	7.02
	16	89.0%	60.89
FFNN architecture c	1	90.5%	3.25
	16	88.9%	73.28

(b) Résultats sur AGNews

**TABLE 6.1** – Exactitude de test et norme spectrale aux minima pour différents réseaux de neurones.

Tout d'abord, nous pouvons remarquer Table 6.1 à l'aide de nos expériences avec des modèles de petite taille que les résultats de RANGAMANI et al. [158], à savoir que les lots de

Modèle	Exactitude de test	Norme Spectrale Riemannienne
GrowNN	68.2%	$1.8945 \cdot 10^5$
FFNN	68%	$6.7778 \cdot 10^5$

TABLE 6.2 – Exactitude de test et Norme Spectrale aux Minima sur COLA

tailles plus élevées mènent à des minima plus aigus sont confirmés. Nos résultats montrent également qu’insérer des neurones pendant la phase d’entraînement mène bien à des minima plus plats. Ces résultats sur des réseaux simples montrent également qu’une norme spectrale riemannienne plus faible semble être corrélée à de meilleures performances de généralisation puisque les exactitudes de test sont meilleurs pour les GrowNN.

Une autre remarque pour compléter les conclusions du Chapitre 3 est que ces résultats se transmettent bien à des réseaux plus complexes, et ce en insérant seulement 0.6% du nombre total de paramètres du modèle. En effet, les résultats de la Table 6.2 montrent que faire croître la tête de classification d’un modèle pré-entraîné type Transformer mène à un optimum plus plat, sans toutefois impact significatif sur l’accuracy.

Ces résultats tendent à montrer que le paradigme d’expansion, de plus en plus utilisé dans le domaine du NAS, est un atout non négligeable pour atteindre des minima plus plats, qui semblent permettre de meilleures performances de généralisation que leurs contreparties aigues.

Il s’agit toutefois de noter que même si plusieurs travaux empiriques semblent montrer un lien réel, aucune preuve théorique ne garantit qu’un minimum plat se traduit systématiquement par une meilleure généralisation (du moins à notre connaissance au moment d’écrire ce manuscrit). Ainsi, nos résultats avec d’extrêmement petits réseaux (le cas avec seulement 10 neurones cachés Table 6.1), les performances sont extrêmement similaires malgré une forte disparité des résultats de planéité. Ceci est probablement dû, comme nous le remarquons au Chapitre 3, aux tailles extrêmement limitées des réseaux qui ne peuvent pas apprendre plus dû à ce facteur limitant. Ce facteur limitant semble également se retrouver dans les cas des tâches de TALN où le nombre de neurones insérés semble trop faible pour avoir un impact réel sur l’accuracy tandis que cela permet tout de même d’atteindre des minima plus plats. Ainsi, si la planéité corrèle bel et bien avec la généralisation, un minimum plus plat n’est pas forcément un meilleur minimum.

En effet, en imaginant une surface de coût avec un minimum global unique entouré d’une large région extrêmement plane. En comparant la planéité de la région plane avec celle du minimum, on pourrait conclure à tort que le minimum est moins bon que l’autre région. La planéité est donc une mesure complémentaire à évaluer avec attention, comme mentionné par ANDRIUSHCHENKO et al. [2].

Il faut alors s’interroger sur la manière standard d’évaluer la généralisation d’un modèle, à savoir de l’évaluer lors d’une phase d’inférence sur un jeu de données de test séparé de celui d’entraînement mais issu comme ce dernier d’un split du corpus d’origine. En effet, dans le cas de modèles de langage par exemple, il pourrait être argumenté qu’une meilleure généralisation est synonyme de meilleure représentation sous-jacente du langage en général et pas forcément de meilleures performances de test sur un sous corpus dont

l'indépendance avec le corpus de train et la représentativité du langage considéré peuvent parfois être remises en question. Ces questions sont de vastes sujets de recherche et restent des problèmes ouverts à ce jour.

Nous étudions dans la partie III finale d'autres mesures de généralisation, tout d'abord en en présentant un état des lieux dans le Chapitre 7, puis en développant un risque non supervisé que nous développons comme une méthode de régularisation Chapitre 8 et que nous adaptons en méthode d'évaluation de généralisation Chapitre 9. La mesure que nous présentons permet d'ajouter une méthode d'évaluation des modèles supplémentaires dans le cas de la classification binaire, ce qui permet d'apporter de la robustesse aux calculs de généralisation, en utilisant un ensemble de mesures plutôt qu'une seule.



## **Troisième partie**

# **Étude des performances de généralisation**



---

## CHAPITRE 7 : MESURES DE GÉNÉRALISATION ET ÉVALUATION DE PERFORMANCES : ÉTAT DE L'ART

*Il n'y a pas de science sans imagination  
comme il n'y a pas d'art sans faits.*

---

– Vladimir Nabokov

---

7.1	Comment mesurer la généralisation d'un ANN . . . . .	76
7.2	Au delà du surapprentissage pour les LLM . . . . .	77

---

Avec la réussite des applications du Deep Learning dans un vaste champ de domaines, la compréhension et l'évaluation des performances des modèles restent relativement mystérieuses. Il est cependant crucial de comprendre les mécanismes sous-jacents à la généralisation des modèles utilisés afin de fournir des garanties minimales dans certains scénarios, mais également pour améliorer les performances générales des outils dérivés de cette technologie et ainsi concevoir de meilleurs modèles.

Nous étudions dans ce chapitre les diverses mesures de généralisation existantes, puis développerons dans le chapitre suivant une méthode non supervisée pour améliorer les classifieurs binaires avant de l'adapter en mesure de généralisation et de l'utiliser pour analyser les réseaux de neurones grossissants étudiés dans les deux parties précédentes.

## 7.1 Comment mesurer la généralisation d'un ANN

La compréhension théorique des réseaux de neurones et de leurs performances est toujours aujourd'hui un problème ouvert, même si de nombreux travaux apportent des fragments d'explication, permettant d'avoir à disposition un certain bagage couvrant de nombreux aspects du problème. Ainsi, pour évaluer les modèles, de nombreuses mesures de généralisation ont vu le jour et une revue complète de celles-ci a d'ailleurs été proposée par JIANG et al. [94].

Du point de vue théorique, de nombreuses tentatives de comprendre les performances de généralisation des modèles de Deep Learning ont récemment vu le jour, notamment les études de NEYSHABUR, TOMIOKA et SREBRO [141], BARTLETT, FOSTER et TELGARSKY [11], NEYSHABUR, BHOJANAPALLI et SREBRO [142], GOLOWICH, RAKHLIN et SHAMIR [62], ARORA et al. [3], NAGARAJAN et ZICO KOLTER [139] et LONG et SEDGHI [131]. L'approche la plus communément suivie est d'approcher des bornes de généralisation typiquement en évaluant la borne supérieure de l'erreur sur le jeu de données de test à partir de quantités calculables sur le jeu de données d'entraînement.

Cependant, évaluer de telles bornes de manière précise est extrêmement complexe et les méthodes actuelles ne permettent pas pour l'instant de capter précisément le comportement de généralisation. Ces mesures appartiennent à trois différentes familles :

- l'analyse PAC-Bayésienne avec les travaux de MCALLESTER [133], KAROLINA DZIUGAITE et ROY [99] et NEYSHABUR et al. [143] et plus récemment de BEHBOODI, CESA et COHEN [12];
- la dimension VC (pour Vapnik-Chervonenkis) et le travail fondateur de VAPNIK et CHERVONENKIS [190];
- les bornes basées sur des calculs de normes avec notamment les articles de NEYSHABUR, TOMIOKA et SREBRO [141], BARTLETT, FOSTER et TELGARSKY [11], NEYSHABUR, BHOJANAPALLI et SREBRO [142] et LEDENT et al. [119]

D'autres méthodes, cette fois empiriques permettent également de caractériser les performances de généralisation des DNN sans devoir dériver des métriques complexes comme par exemple les mesures de planéité mentionnées Section II issues des travaux de KESKAR et al. [105], mais également par exemple sur la mesure de Fisher-Rao introduite par LIANG et al. [124]. Cependant, la corrélation empirique n'entraîne pas forcément une relation causale entre une mesure et la généralisation comme mentionné par KAROLINA DZIUGAITE et ROY [99].

Que l'analyse de la généralisation soit théorique ou empirique, un composant fondamental de celle-ci est la notion de mesure de complexité qui est une mesure monotonement dépendante de certains aspects de la généralisation. Une telle mesure de complexité devrait se comporter ainsi : plus cette dernière est faible, plus le saut de généralisation l'est étalemment. Une telle mesure de complexité peut dépendre des propriétés intrinsèques du modèle comme son architecture, l'algorithme d'optimisation employé ou encore les données d'entraînement mais ne peut pas dépendre du jeu de données de test utilisé pour l'évaluation.

JIANG et al. [94] proposent une analyse complète d'une quarantaine de mesures de complexité courantes pour évaluer les propriétés de généralisation de CNN pour des tâches de

vision par ordinateur et prouve que les mesures de planéité sont parmi les plus performantes pour évaluer la généralisation d'un modèle. Une mesure telle que celle que nous employons dans nos travaux, développée par RANGAMANI et al. [158] est donc effectivement adaptée pour évaluer la généralisation des modèles grossissants, sous réserve que les architectures comparées possèdent la même structure puisque ces mesures de complexité dépendent des propriétés du modèle. Cependant pour des modèles à plusieurs millions de paramètres comme les Transformers pré-entraînés pour des tâches de TALN (les *Large Language Models* ou LLM), ces mesures sont très coûteuses d'un point de vue calculatoire à mettre en place de manière récurrente.

Enfin pour les LLMs, le paradigme d'entraînement consiste en un pré-entraînement sur de vastes quantités de données non annotées (*pre-training*) avant une "spécialisation" sur une tâche en particulier via réglage fin (*fine-tuning*), effectuée souvent de manière supervisée sur des corpora adaptés. Les jeux de données de fine-tuning sont ainsi généralement beaucoup plus petits que le nombre de paramètres constituant le modèle, ce qui est souvent la cause du surapprentissage, les LLM perdant ainsi certaines de leurs propriétés originales de généralisation au profit d'une hyper-spécialisation sur la tâche courante. Il s'agit donc pour le modèle d'atteindre de bonnes performances sur la tâche courante tout en gardant de bonnes capacités de généralisation, sous-tendant une bonne représentation du langage.

## 7.2 Au delà du surapprentissage pour les LLM

De nombreux travaux récents montrent à la fois théoriquement, avec notamment l'article de DUBOST et al. [47], et expérimentalement (voir par exemple une méta analyse sur un grand ensemble de compétitions Kaggle par ROELOFS et al. [161]) que le surapprentissage pourrait ne pas être en réalité un problème majeur en Machine Learning.

Comme mentionné précédemment cependant, les LLM et modèles équivalents sont sensibles au surapprentissage lors de réglage fin sur des jeux de données de taille modeste et ainsi perdre en généralisation, c'est le cas de l'*agressive fine-tuning* mentionné par JIANG et al. [93] qui mène souvent à du surapprentissage sur la tâche. Les performances de généralisation du modèle peuvent ainsi devenir faibles sur des données de test comme montré par DEVLIN et al. [41], PHANG, FÉVRY et BOWMAN [152] et LEE, CHO et KANG [120], mais également sur des données hors du domaine de la tâche de fine-tuning ou sur des tâches similaires comme étudié par KARIMI MAHABADI, BELINKOV et HENDERSON [98] et AGHAJANYAN et al. [1].

Pour résoudre ce problème, de nombreuses approches comme celles de JIANG et al. [93], GOUK, HOSPEDALES et PONTIL [67], CHEN et al. [26], LEE, CHO et KANG [120] et XU et al. [207] améliorent les résultats grâce à un corpus de validation pour optimiser les hyper-paramètres ou même modifier les poids du modèle. Il est ainsi intéressant d'étudier dans ce contexte la notion de généralisation à l'aide de mesures peu sensibles au surapprentissage. Dans ce cadre, utiliser un ensemble de mesures capables de capturer l'information de généralisation semble être plus recommandable que de se contenter d'une seule.

Ainsi, pour les classifieurs binaires, nous complétons dans la suite la mesure de planéité des réseaux avec une approximation non supervisée du risque réel du classifieur.

Nous décrivons tout d'abord dans le Chapitre 8 cette mesure et l'appliquons pour optimiser un modèle après son entraînement avant de l'utiliser comme mesure de généralisation dans le Chapitre 9 pour évaluer le saut de généralisation sans aucun jeu de données de validation.

---

# CHAPITRE 8 : RISQUE NON SUPERVISÉ : MESURE DE GÉNÉRALISATION ET MÉTHODE DE RÉGULARISATION

*Ne tenez pour certain que ce qui est démontré.*

---

– Isaac Newton

---

8.1	Risque Non Supervisé : Présentation Théorique . . . . .	80
8.2	Le Risque Non Supervisé comme Méthode de Régularisation : le post-tuning . . . . .	85

---

Dans ce chapitre, nous présentons un risque de classification binaire non supervisé original dérivé des résultats de BALASUBRAMANIAN, DONMEZ et LEBANON [7]. Ce risque non supervisé peut être appliqué à tout classifieur binaire fournissant en sortie un score scalaire, incluant des réseaux de DL. Nous montrerons que ce risque peut être utilisé dans le cadre d'un entraînement non supervisé, en supposant les classifieurs proches de leurs optima. Ainsi ce résultat peut être utilisé pour post-tuner des réseaux déjà optimisés, améliorant encore leurs performances de généralisation. Nous adapterons ensuite dans le Chapitre 9 ce risque pour l'utiliser comme une mesure de généralisation peu sensible au surapprentissage.

## 8.1 Risque Non Supervisé : Présentation Théorique

Les méthodes de DL non supervisées se concentrent généralement sur l'apprentissage de la représentation des données, (voir les travaux de MIKOLOV et al. [137] et RADFORD, METZ et CHINTALA [156]) et sur le partitionnement profond, avec quelques articles supplémentaires se détachant de ces paradigmes majoritaires, notamment les travaux de METZ et al. [136], KILINC et UYSAL [109] et GOLTS, FREEDMAN et ELAD [63]. Les modèles génératifs dominent ce domaine grâce à leur capacité à capturer les structures sous-jacentes aux observations qui constituent la seule information accessible dans un contexte purement non supervisé. Par exemple, les méthodes de détection d'anomalie dans le texte utilisent des modèles issus de cette famille, notamment par YAP [209] où les auteurs adaptent des GANs, qui avaient déjà montré des résultats remarquables pour des tâches similaires sur des images, pour générer des phrases non problématiques et ensuite calculer un score d'anomalie pour du texte.

Notre proposition ici est basée sur les deux hypothèses décrites par BALASUBRAMANIAN, DONMEZ et LEBANON [7] et les complète avec une troisième. Ces hypothèses que nous détaillerons permettent un entraînement non supervisé des classifieurs binaires en exploitant une proportion connue des classes. Ceci est assez similaire aux travaux faits dans le domaine du LLP (*Learning with Label Proportion*), notamment par QI et al. [154] et FISH et REYZIN [57], mais également dans une moindre mesure avec des méthodes plus génériques d'apprentissage faiblement supervisé exploitant l'incertitude sur les exemples d'entraînement comme par ZANTEDESCHI, EMONET et SEBBAN [211].

### 8.1.1 Risque non supervisé proposé

Soit un classifieur binaire avec comme paramètres  $\theta$  et comme sortie scalaire, le score  $f(x) \in \mathbb{R}$  pour une entrée  $x$ . Soit  $y \in \{0, 1\}$  la vraie classe de  $x$  : nous supposons non connue la valeur de  $y$  puisque notre approche est non supervisée. La décision de classification est la classe  $\hat{y} = 0 \Leftrightarrow f(x) \leq 0$ , et  $\hat{y} = 1 \Leftrightarrow f(x) > 0$ .

Le risque théorique de classification avec la fonction Hinge Loss est défini comme suit :

$$\begin{aligned}
 R(\theta) &= E_{p(x,y)} [(1 - f(x) \cdot (2y - 1))_+] \\
 &= P(y = 0) \int p(f(x) = \alpha | y = 0) (1 + \alpha)_+ d\alpha + \\
 &\quad P(y = 1) \int p(f(x) = \alpha | y = 1) (1 - \alpha)_+ d\alpha
 \end{aligned} \tag{8.1}$$

Dans la suite, nous exploitons trois hypothèses comme mentionné précédemment afin d'approcher Eq 8.1 et d'en dériver une fonction de coût utilisable pour un entraînement non supervisé. Ces hypothèses sont :

- La distribution conditionnelle par classe des scores  $p(f(x)|y)$  est Gaussienne ;
- Le prior par classe  $P(y)$  est connu ;

- Les deux Gaussiennes  $p(f(x)|y = 0)$  et  $p(f(x)|y = 1)$  sont bien séparées lorsque  $\theta$  est proche d'un optimum.

Les deux premières hypothèses sont issues de BALASUBRAMANIAN, DONMEZ et LEBANON [7], dans lequel les auteurs donnent des arguments à la fois théoriques et expérimentaux pour les justifier. En résumé :

- Le théorème central limite prouve que  $p(f(x)|y)$  tend effectivement vers une Gaussienne à mesure que le nombre de dimensions augmente et un certain nombre de résultats expérimentaux montre que cela est souvent également vrai en faible dimension. Nous avons également réalisé un certain nombre d'expériences confirmant ces résultats, que nous détaillons dans la Section 8.1.2.
- Pour la seconde hypothèse,  $P(y)$  peut être estimé en pratique dans de nombreuses situations. Par exemple, la prévalence d'une maladie est souvent connue, même si la classe d'une observation spécifique est inconnue. De plus nous montrons dans la Section 8.1.2 que les performances du classifieur semblent être robustes à des estimations relativement grossières de  $P(y)$ , jusqu'à  $\pm 10\%$ . Enfin, dans le cas de l'usage de ce risque comme mesure de généralisation sans corpus de validation dans le cadre d'un entraînement supervisé comme c'est le cas dans la Section 9.1, nous bénéficions des valeurs exactes de ces priors.
- Enfin, nous montrons dans la Section 8.1.2 que la troisième hypothèse est bel et bien vérifiée autour d'un optimum, ce qui nous contraint tout de même à initialiser correctement notre classifieur avant de l'apprendre dans le cas de l'usage de ce risque comme risque d'entraînement non supervisé, mais également le cas des LLM qui sont désormais bien souvent pré-entraînés de manière à avoir des performances excellentes même dans un contexte "few-shot" ou "zero-shot" (voir l'analyse de BROWN et al. [17]).

Nous proposons alors de réécrire le risque comme suit :

$$R(\mu, \sigma) = P(y = 0) \int N(\alpha; \mu_0, \sigma_0)(1 + \alpha)_+ d\alpha + P(y = 1) \int N(\alpha; \mu_1, \sigma_1)(1 - \alpha)_+ d\alpha$$

où  $\mu = (\mu_0, \mu_1) \in \mathbb{R}^2$  et  $\sigma^2 = (\sigma_0^2, \sigma_1^2) \in \mathbb{R}^{+2}$  sont respectivement les moyennes et variances des Gaussiennes correspondantes  $p(f(x)|y = 0; \theta)$  et  $p(f(x)|y = 1; \theta)$ , et

$$N(\alpha; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\alpha-\mu)^2}{2\sigma^2}}$$

En supprimant la non-linéarité, nous obtenons :

$$\begin{aligned}
 R(\mu, \sigma) &= P(y = 0) \int_{-1}^{+\infty} N(\alpha; \mu_0, \sigma_0) d\alpha + \\
 &P(y = 0) \int_{-1}^{+\infty} \alpha N(\alpha; \mu_0, \sigma_0) d\alpha + \\
 &P(y = 1) \int_{-\infty}^1 N(\alpha; \mu_1, \sigma_1) d\alpha - \\
 &P(y = 1) \int_{-\infty}^1 \alpha N(\alpha; \mu_1, \sigma_1) d\alpha
 \end{aligned}$$

Nous savons que :

$$\int_a^b N(x; \mu, \sigma) dx = \frac{1}{2} \left( \operatorname{erf} \left( \frac{b - \mu}{\sigma \sqrt{2}} \right) - \operatorname{erf} \left( \frac{a - \mu}{\sigma \sqrt{2}} \right) \right)$$

et :

$$\int_a^b x N(x; \mu, \sigma) dx = \mu \int_a^b N(x; \mu, \sigma) dx - \sigma^2 [N(x; \mu, \sigma)]_a^b$$

Donc :

$$\begin{aligned}
 \int_a^b x N(x; \mu, \sigma) dx &= \frac{\mu}{2} \left( \operatorname{erf} \left( \frac{b - \mu}{\sigma \sqrt{2}} \right) - \operatorname{erf} \left( \frac{a - \mu}{\sigma \sqrt{2}} \right) \right) - \\
 &\sigma^2 (N(b; \mu, \sigma) - N(a; \mu, \sigma))
 \end{aligned}$$

Ce qui nous donne :

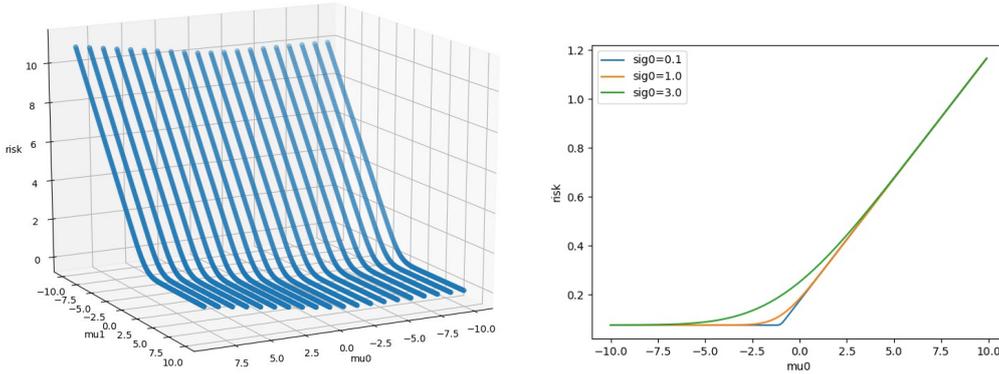
$$\begin{aligned}
 R(\mu, \sigma) &= \frac{P(y = 0)}{2} (1 + \mu_0) \left( 1 - \operatorname{erf} \left( \frac{-1 - \mu_0}{\sigma_0 \sqrt{2}} \right) \right) + P(y = 0) \sigma_0^2 N(-1; \mu_0, \sigma_0) + \\
 &\frac{P(y = 1)}{2} (1 - \mu_1) \left( 1 + \operatorname{erf} \left( \frac{1 - \mu_1}{\sigma_1 \sqrt{2}} \right) \right) + P(y = 1) \sigma_1^2 N(1; \mu_1, \sigma_1) \quad (8.2)
 \end{aligned}$$

Afin d'utiliser ce risque comme une fonction de coût pour un algorithme de descente de gradient stochastique dans les frameworks modernes de Deep Learning, nous devons réécrire cette expression comme une fonction dérivable qui dépend des paramètres  $\theta$ . Nous analysons dans la suite la fonction  $R(\mu, \sigma)$  et donnons finalement l'expression permettant ceci.

### 8.1.2 Analyse du coût

Notons  $p_0 = P(Y = 0)$  et représentons l'Équation 8.2 comme une fonction de  $(\mu_0, \mu_1)$  dans la Figure 8.1 (gauche), pour  $p_0 = 0.1$  et  $\sigma_0 = \sigma_1 = 1$ .

En fixant  $\mu_1$ , nous pouvons voir Figure 8.1 (droite) que le risque comme fonction de  $\mu_0$  peut être approché correctement par une ReLU mise à échelle et translatée tant que les variances sont suffisamment faibles. De plus, plus  $\sigma_0$  (et  $\sigma_1$ ) sont faibles, meilleur est le



**FIGURE 8.1** – Risque non supervisé comme fonction de  $(\mu_0, \mu_1)$  (gauche), et seulement  $\mu_0$  (droite) pour  $\mu_1 = 2, \sigma_1 = 1$  et  $\sigma_0 \in \{0.1, 1, 3\}$

risque. Faire varier  $\mu_1$  et  $\sigma_1$  n'opère qu'une translation verticale de cette courbe, au-dessus de l'axe horizontal. En supposant que le risque a été minimisé selon  $\mu_1$ , le minimum global du risque peut donc être obtenu en réduisant linéairement  $\mu_0$ . Réciproquement, des risques plus faibles sont obtenus lorsque  $\mu_1$  augmente.

### 8.1.3 Approximation de la distribution bi-Gaussienne

En supposant désormais que les paramètres sont proches de l'optimum global, les deux modes  $(\mu_0, \sigma_0)$  et  $(\mu_1, \sigma_1)$  des distributions de score sont bien séparés avec un faible recouvrement. En effet, l'analyse précédente a montré qu'en se rapprochant de l'optimum,  $\mu_0$  décroissait,  $\mu_1$  augmentait, et  $\sigma_0$  et  $\sigma_1$  étaient faibles. Avec cette observation, une bonne approximation de  $\mu_0$  et  $\mu_1$  peut être calculée en séparant les scores  $f(x)$  selon le  $p_0$ -quantile  $x_{p_0}$ . Soit  $X^-$  le sous-espace de taille  $N^-$  de tous les points à gauche du  $p_0$ -quantile :

$$X^- = \{x \in X \text{ t.q. } f(x) < f(x_{p_0})\}$$

de même, à droite :

$$X^+ = \{x \in X \text{ t.q. } f(x) \geq f(x_{p_0})\}$$

Nous pouvons désormais approximer les paramètres gaussiens de manière déterministe :

$$\begin{aligned} \mu_0 &\simeq \frac{1}{N^-} \sum_{x \in X^-} f(x) & \mu_1 &\simeq \frac{1}{N^+} \sum_{x \in X^+} f(x) \\ \sigma_0^2 &\simeq \left( \frac{1}{N^-} \sum_{x \in X^-} f(x)^2 \right) - \left( \frac{1}{N^-} \sum_{x \in X^-} f(x) \right)^2 \\ \sigma_1^2 &\simeq \left( \frac{1}{N^+} \sum_{x \in X^+} f(x)^2 \right) - \left( \frac{1}{N^+} \sum_{x \in X^+} f(x) \right)^2 \end{aligned} \quad (8.3)$$

Intuitivement, réduire le risque peut être effectué en réduisant les variances  $\sigma_0$  et  $\sigma_1$  et en augmentant la distance entre les moyennes  $\mu_0$  et  $\mu_1$  dans l'espace du score du classifieur  $\{f(x)\}$ .

Il est désormais possible d'injecter les équations 8.3 dans l'équation 8.2 et obtenir un coût dérivable  $R(\theta)$  par rapport aux paramètres du réseau  $\theta$ , qui peut donc être utilisé simplement avec les outils de deep learning contemporains. Dans la section suivante, nous adaptons ce risque original dans un contexte vérifiant nos hypothèses : afin d'effectuer du post-tuning non supervisé et donc en essayant d'optimiser encore les performances d'un réseau en extrayant plus d'information sous-jacente aux données. Nous verrons également dans la Section 9.1 du Chapitre 9 que ce risque non supervisé peut être utilisé comme une mesure de généralisation peu sensible au surapprentissage et pouvant donc compléter d'autres mesures plus standards.

## 8.2 Le Risque Non Supervisé comme Méthode de Régularisation : le post-tuning

Dans cette section, nous utilisons le risque non supervisé développé précédemment comme une métrique à minimiser après la phase d'apprentissage afin d'améliorer encore les performances du modèle. L'algorithme 4 résume cette phase de post-tuning.

---

**Algorithm 4:** Entraînement non supervisé end-to-end en utilisant le risque non supervisé.

---

- Initialisation :
    - Soit une tâche de classification binaire pour laquelle la proportion d'éléments de classe 0,  $p_0$  est connue approximativement ;
    - Soit un corpus d'observations sans labels  $\{x_i\}_{1 \leq i \leq N}$  ;
    - Soit un DNN  $g_\phi(x)$  de paramètres  $\phi$  calculant une représentation vectorielle des entrées  $x$ , qui est donné en entrée d'une couche finale de classification linéaire  $f_\theta(g_\phi(x))$  de paramètres  $\theta$  ; les valeurs de  $\phi$  et  $\theta$  peuvent être issues d'un entraînement ou initialisées.
  - Itération de :
    - Calcul d'une passe avant sur les données  $\{x_i\}_{1 \leq i \leq N}$  avec les paramètres courants  $\phi, \theta$ .
    - Calcul des scores du classifieurs  $\{s_i = f_\theta(g_\phi(x_i))\}_{1 \leq i \leq n}$  sur tout le corpus  $N$ , ou sur un sous-ensemble d'observations  $n$  suffisamment grand pour supposer que la distribution des classes dans le sous-ensemble est représentative de la distribution dans le corpus entier.
    - Tri de la liste des scores  $(s_i)_{1 \leq i \leq n}$  pour calculer le  $p_0$ -quantile  $x_{p_0}$ .
    - Calcul des paramètres gaussiens  $\mu = (\mu_0, \mu_1), \Sigma = (\sigma_0, \sigma_1)$  avec les Équations 8.3.
    - Calcul du risque (Eq 8.2) avec ces paramètres.
    - Application de la différentiation automatique pour calculer  $\nabla_\theta R(\mu, \Sigma)$ , et optionnellement  $\nabla_\phi R(\mu, \Sigma)$  ;
    - Calcul d'une étape de descente de gradient pour mettre à jour  $\theta$ , et optionnellement  $\phi$ .
-

### 8.2.1 Validation expérimentale

Cette approche est évaluée ici sur le corpus Reuters-21578, pour détecter de manière non supervisée des anomalies thématiques en reproduisant les conditions expérimentales développées dans [164]. L'aire sous la courbe (AUC) est donc utilisée pour évaluer la qualité de la détection d'anomalies.

#### Détection d'anomalies sur des textes

La tâche de détection d'anomalie peut être vue comme une tâche de classification binaire où le modèle donne en sortie  $y = 0$  lorsque l'observation d'entrée  $x$  appartient à la classe normale et  $y = 1$  lorsque  $x$  est une anomalie (*outlier*). Cependant, différentes conditions expérimentales peuvent mener à des schémas d'entraînement variés, comme montré par BELLINGER et al. [13] :

- **Classification supervisée** où les exemples positifs et négatifs sont identifiés et labellisés ;
- **Modèles une classe (*One-class model*)** où le jeu de données d'entraînement est seulement composé des exemples normaux ;
- **Classification non supervisée** où il peut y avoir des exemples d'entraînement positifs et négatifs mais pas de labels.

Notre méthode appartient à la dernière catégorie : la classification non supervisée d'un classifieur binaire. Cependant, elle peut également être vue dans un certain sens comme appartenant à la seconde, puisque seule une faible proportion des exemples d'entraînement sont des anomalies. En réalité, les modèles une classe incluant souvent un paramètre  $\nu$  pour contrôler la proportion d'exemples d'entraînement pouvant se situer hors limites de la classe (voir l'article de TAX et DUIN [184]), notre approche n'est pas si différente de cette modélisation.

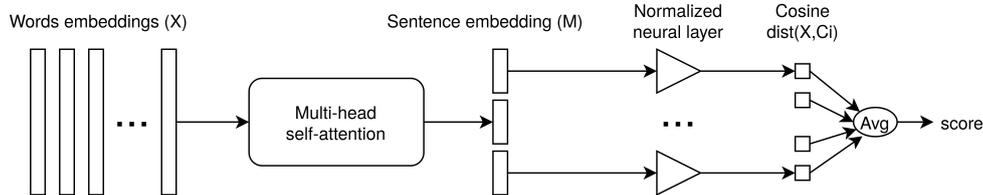
Nous nous basons dans cette section sur le modèle de détection d'anomalies textuelles proposé par RUFF et al. [164] qui est ensuite entraîné avec notre risque non supervisé original plutôt que l'objectif CVDD une-classe original, notre méthode établissant de nouveaux résultats état-de-l'art pour de la détection d'anomalies textuelles sur le corpus Reuters. Le travail proposé par RUFF et al. [164] est inspiré par la famille des modèles une-classe (étudiée notamment par SCHÖLKOPF et al. [168]), et en particulier les approches les plus récentes de cette famille reposant sur des représentations des données issues de l'apprentissage profond (notamment les travaux de RUFF et al. [163] et CHALAPATHY, MENON et CHAWLA [22]).

Le modèle de détection d'anomalies textuelles décrit par RUFF et al. [164] fonctionne comme suit : le modèle prend en entrée une séquence de taille variable de représentations vectorielles de mots pré-entraînée (*pre-trained word embeddings*) de Glove (introduit par PENNINGTON, SOCHER et MANNING [150]) et la transforme en une matrice  $M$  de plongement de document de taille fixe avec une auto-attention multi-têtes (à l'image de ce qui est

fait par LIN et al. [126]). Chacune des  $r$  têtes, ou colonnes de la matrice résultante  $M$  est une combinaison linéaire des représentations vectorielles de mots originales.

Le score d'anomalie est ensuite calculé comme étant la distance cosinus moyenne entre ces  $r$  embeddings de phrases et un ensemble de  $r$  vecteurs de contexte  $\{c_k\}_{1 \leq k \leq r}$ . Nous proposons de reformuler cette dernière étape dans le contexte des ANNs, en encodant chacun des vecteurs contexte  $c_k$  comme le paramètre  $\theta$  d'une couche linéaire sans biais, qui donne donc en sortie le produit scalaire entre ses poids  $\theta$  et l'entrée  $x$ , normalisé à la fois par la norme de Frobenius de  $\theta$  et de  $x$  :

$$f_{cos}(x) = \frac{\sum_i \theta_i x_i}{(\sum_i \theta_i^2)(\sum_i x_i^2)} \quad (8.4)$$



**FIGURE 8.2** – Modèle de détection d’anomalies textuelles; adapté de RUFF et al. [164]. Les  $r$  couches linéaires normalisées implémentent l’Équation 8.4.

Ce modèle, illustré Figure 8.2, est entraîné par RUFF et al. [164] avec la fonction de coût CVDD non supervisé; qui minimise la distance cosinus entre les exemples d’entraînement et les vecteurs contexte, apprenant ainsi  $r$  vecteurs de contexte représentant  $r$  sujets du corpus d’entraînement. Nous proposons de remplacer cette fonction par notre risque non supervisé décrit Section 8.1.1, qui apprend les paramètres du modèle pour séparer les scores des exemples d’entraînement normaux des potentielles anomalies. L’approche CVDD entraîne un modèle à auto-attention sur les représentations vectorielles de mots d’une phrase pour produire une représentation vectorielle de taille fixe de la phrase entière. Cette représentation vectorielle finale des phrases est l’entrée de nos couches linéaires normalisées comme montré Figure 8.2. En initialisant les  $r$  couches linéaires normalisées avec les paramètres obtenus par CVDD  $(c_i)_{1 \leq i \leq r}$ , et en moyennant leurs sorties, nous pouvons reproduire exactement le modèle CVDD avec les couches décrites précédemment et donc apprendre les  $(c_i)_{1 \leq i \leq r}$  avec l’algorithme 4 pour améliorer les performances finales du modèle.

Le jeu de données Reuters-21578 est composé d’articles de presse avec un sujet par article (ce sujet étant le label de l’article). Chacun des sept labels listés Table 8.1 est considérée comme la classe normale l’une après l’autre et tous les autres labels sont représentants de la classe négative. En suivant les conditions expérimentales de RUFF et al. [164], le corpus d’entraînement ne contient que des exemples positifs alors que le corpus de test contient toutes les données.

Pour l’entraînement avec notre risque non supervisé, nous avons choisi arbitrairement le critère Adam et avons entraîné le modèle pour 1000 époques avec un learning rate de

**TABLE 8.1** – Résultats d’aire sous la courbe (AUC) sur de la détection d’anomalies textuelles : les nombres en police normale dans les colonnes CVDD sont obtenus en ré-appliquant le code de RUFF et al. [164] ; les nombres correspondant issus de l’article original sont reportés en italique et entre parenthèses : il peut y avoir quelques différences, issues probablement d’environnements expérimentaux différents (GPUs, versions de bibliothèques...). Les colonnes "Eq 8.2" correspondent à notre approche. Il s’agit de noter que seule la fonction de risque change, tous les hyper-paramètres (modèle, nombre de paramètres, ...) restent identiques par ailleurs.

Normal classe	r=3	r=3	r=5	r=5	r=10	r=10
	CVDD	Eq 8.2	CVDD	Eq 8.2	CVDD	Eq 8.2
<i>earn</i>	93.9 (94.0)	<b>96.1</b>	92.7 (92.8)	<b>95.1</b>	88.2 (91.8)	<b>96.5</b>
<i>acq</i>	90.1 (90.2)	<b>92.2</b>	88.6 (88.7)	<b>92.7</b>	91.5 (91.5)	<b>94.2</b>
<i>crude</i>	89.7 (89.6)	<b>92.2</b>	92.5 (92.5)	<b>96.2</b>	96.4 (95.5)	<b>98.1</b>
<i>trade</i>	97.9 (98.3)	<b>98.2</b>	98.1 (98.2)	<b>98.1</b>	<b>99.6</b> (99.2)	99.4
<i>money-fx</i>	81.9 (82.5)	<b>89.5</b>	78.0 (76.7)	<b>91.3</b>	<b>83.1</b> (82.8)	81.1
<i>interest</i>	92.4 (92.3)	<b>94.2</b>	92.1 (91.7)	<b>95.5</b>	97.2 (97.7)	<b>98.4</b>
<i>ship</i>	96.8 (97.6)	<b>98.8</b>	92.8 (96.9)	<b>98.7</b>	96.1 (95.6)	<b>97.0</b>

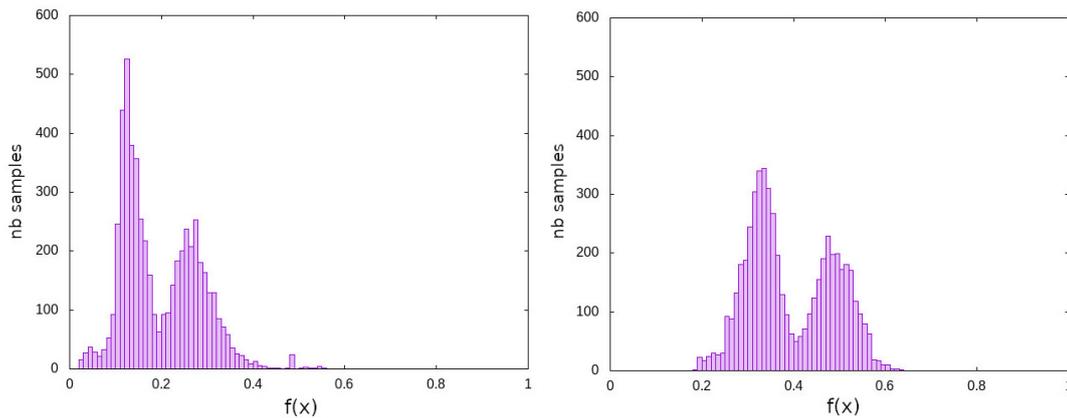
$10^{-4}$  et un momentum de  $10^{-6}$ . Ces valeurs ont été choisies pour que le risque converge avant 1000 époques sur le jeu de données d’entraînement. Le seul hyper-paramètre restant est la valeur de  $p_0$  qui représente la proportion attendue d’exemples négatifs. Nous avons essayé deux valeurs ( $p_0 = 0.1$  et  $0.5$ ) et avons choisi la valeur avec le risque le plus faible sur un jeu de données de validation. Ce jeu de données est égal au jeu de données d’entraînement (uniquement les exemples positifs) fusionné et mélangé avec 50% d’exemples négatifs aléatoires. Ce jeu de données de validation n’a pas de labels. Nous pouvons observer une amélioration quasi systématique en utilisant notre proposition de risque non supervisé, comparé avec le risque CVDD. La réduction relative moyenne du taux d’erreur obtenu avec notre proposition est de -32% avec  $r = 3$  et -43% avec  $r = 5$ , ce qui peut être considéré comme une amélioration significative puisque cela signifie que notre proposition divise pratiquement par 2 le nombre d’erreurs faites par CVDD. L’amélioration obtenue avec  $r = 10$  est plus modeste, avec une réduction de "seulement" 22% du taux d’erreur. Nous discutons de ces résultats dans la suite.

## Discussion

Discutons de ces résultats expérimentaux à la lumière des hypothèses nécessaires au modèle.

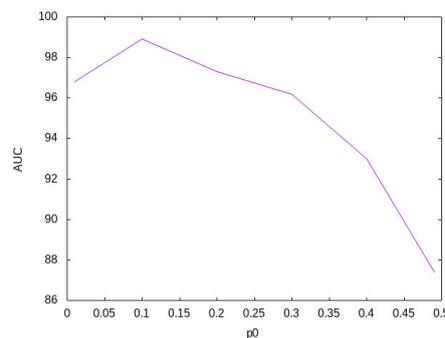
**1ère et 3e hypothèses :** La première hypothèse concerne la bi-gaussianité des scores qui résulte du théorème central limite et l’indépendance relative entre les dimensions des représentations vectorielles des phrases. Pour vérifier cette hypothèse expérimentalement, nous représentons Figure 8.3 l’histogramme des scores  $f(x)$  sur tous les exemples d’en-

entraînement avant (gauche) et après (droite) l'apprentissage non supervisé. Nous pouvons observer que l'hypothèse de bi-gaussianité est vérifiée. Nous pouvons également observer l'effet de l'entraînement non supervisé qui augmente la distance entre chacun des modes, de 0.14 à 0.17, ce qui était attendu. Cette figure confirme également notre troisième hypothèse, à savoir que la distribution initiale est déjà composée de deux modes relativement bien séparés. Ceci peut ne pas toujours arriver en pratique mais d'autres résultats expérimentaux réalisés sur divers corpora suggèrent que c'est tout de même souvent le cas, sous réserve qu'un pré-entraînement correct du classifieur est réalisé, ce qui est le cas ici avec CVDD.



**FIGURE 8.3** – Histogrammes des scores du classifieur sur le jeu de données d'entraînement, avant (gauche) et après (droite) l'entraînement non supervisé avec notre risque.

**2e hypothèse :** En ce qui concerne notre 2<sup>de</sup> hypothèse à propos de la répartition des classes et du prior  $p_0$ , nous illustrons Figure 8.4 l'AUC de test pour plusieurs valeurs de  $p_0$  d'entraînement.



**FIGURE 8.4** – AUC de test pour plusieurs valeurs de  $p_0$ , avec  $r = 3$  et  $nc = \text{"ship"}$ . Du fait de la symétrie de la classification binaire non supervisée,  $p_0$  varie entre 0 et 0.5

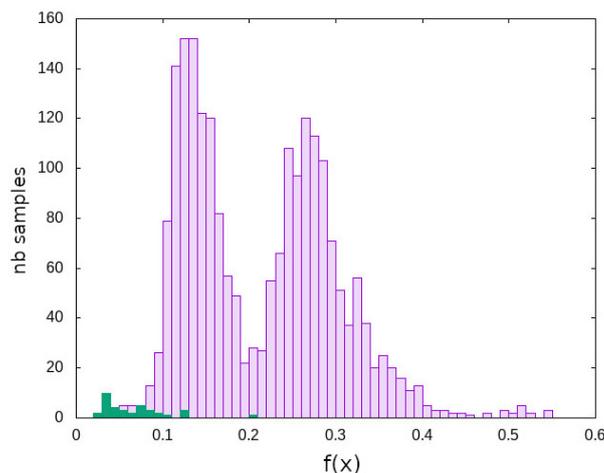
Notons que pendant la phase d'entraînement, en suivant les conditions expérimentales mentionnées utilisée par RUFF et al. [164], il n'y a pas d'anomalies explicites dans le jeu

de données d'entraînement, le  $p_0$  réel devrait donc être nul. Cependant, dans tout corpus réaliste, des exemples divergent toujours relativement aux exemples typiques et sortent ainsi de la distribution générale. De tels exemples peuvent être considérés comme des anomalies, comme nous l'illustrons dans l'analyse qualitative suivante.

La Figure 8.4 montre que l'influence de  $p_0$  est significative, mais avec une dégradation relative des performances dans l'ensemble des valeurs possibles et reste donc contrôlée.  $p_0$  devrait donc être préférablement entraîné à l'aide d'un corpus de validation comme nous l'avons fait dans nos expériences si possible. Il faut noter que dans le cas d'entraînement supervisé, les labels étant connus, la valeur réelle de  $p_0$  peut être estimée avec précision comme nous le verrons dans la Section 9.1 du Chapitre 9.

### Analyse qualitative :

Rappelons que le corpus de test est créé en choisissant un sujet unique comme classe normale et en rassemblant tous les autres sujets comme étant des anomalies. Le résultat est donc très déséquilibré. La Figure 8.5 représente la distribution des scores des classifieurs sur le corpus de test, montrant les exemples positifs et négatifs avec une couleur et une opacité différentes.



**FIGURE 8.5** – Histogramme des scores des exemples de test normaux (vert plein en bas à gauche) et négatifs (rose transparent dominant entre  $x = 0.1$  et  $x = 0.6$ ).

Nous observons que :

- À l'extrême gauche de la distribution, nous pouvons trouver tous les exemples de la classe "normale", par exemple :  
*A 24-hour strike by Belgian public employees protesting against a government pay ...*
- L'ensemble des exemples "non normaux" est distribué sur un large espace couvrant du milieu gauche à la droite de la distribution. Le modèle divise ensuite ce large ensemble en plusieurs niveaux d'anomalies :

- À l'extrême droite, les anomalies composées de seulement quelques mots, typiquement des mois : *august, july...*
- Au centre de la distribution, des exemples de taille standard avec un des sujets "non normaux", par exemple :  
*Video Jukebox Network inc said it signed a letter of intent to purchase up to 3.5 mln shares of the four mln shares of the company's common stock...*



---

## CHAPITRE 9 : LES RÉSEAUX DE NEURONES EXPANSIFS GÉNÉRALISENT-ILS MIEUX ?

*La science n'est rien de plus que  
l'exploration d'un miracle que nous  
n'arrivons pas à expliquer.*

---

– Ray Bradbury

---

9.1	Le Risque Non Supervisé comme Mesure de Généralisation . . . . .	94
9.2	Résultats expérimentaux . . . . .	104
9.3	Discussion . . . . .	107
A	Évolution des métriques au cours de l'entraînement . . . . .	131

---

Dans ce chapitre nous concluons notre travail en rassemblant les différents modèles et métriques proposés afin de mesurer avec le plus de précision possible les capacités de généralisation comparées des GrowNN et des FFNN standards. Nous décrivons tout d'abord le risque développé chapitre 8 comme une mesure de généralisation non supervisée, ce qui nous permettra de l'utiliser comme métrique d'early stopping sans corpus de validation ; ce qui nous permettra de compléter notre analyse de la flatness des GrowNN sur divers corpora et architectures.

## 9.1 Le Risque Non Supervisé comme Mesure de Généralisation

Nous utilisons dans cette section le risque développé précédemment comme une mesure non supervisée de généralisation. Nous décrivons ici suit l'algorithme utilisé pour ce faire avant de détailler les résultats obtenus. Nous notons  $X^-$  le sous ensemble de taille  $N^-$  de tous les points sur la gauche du  $p_0$ -quantile et  $X^+$  le sous ensemble de taille  $N^+$  de tous les points de l'autre côté du  $p_0$ -quantile.

Considérons un jeu de données d'entraînement de taille  $N$  et un classifieur  $f_\theta(\cdot)$ . Nous adaptons l'algorithme 4 afin de calculer à chaque époque d'entraînement supervisé l'approximation du risque théorique que représente notre risque non supervisé pour de la classification binaire.

Ainsi, puisque nous disposons des vrais classes des observations, la vraie valeur de  $p_0$  est disponible et les paramètres des deux gaussiennes, représentant respectivement la distribution de la classe positive et négative sont aisément calculables. Il suffit dès lors d'utiliser l'équation Eq 8.1 pour calculer le risque non supervisé tout au long de l'apprentissage pour ensuite analyser ses variations.

- Pendant la passe avant sur le jeu de données d'entraînement  $\{x_i\}_{1 \leq i \leq N}$ , stockage des scores du classifieur  $\{f_\theta(x_i)\}_{1 \leq i \leq N}$  sur tout le corpus et estimation de  $p_0$ .
- Tri de la liste des scores  $(f_\theta(x_i))_N$  et extraction du  $p_0$ -quantile  $x_{p_0}$ ; Construction de  $X^- = \{x_i | f_\theta(x_i) \leq f_\theta(x_{p_0})\}$  de taille  $N^-$  et similairement  $X^+$  de taille  $N^+$ .
- Calcul des paramètres Gaussiens
 
$$\mu = (\mu_0, \mu_1), \sigma = (\sigma_0, \sigma_1) \text{ avec}$$

$$\mu_0 = \frac{1}{N^-} \sum_{x \in X^-} f(x),$$

$$\mu_1 = \frac{1}{N^+} \sum_{x \in X^+} f(x),$$

$$\sigma_0^2 = \left( \frac{1}{N^-} \sum_{x \in X^-} f(x)^2 \right) - \left( \frac{1}{N^-} \sum_{x \in X^-} f(x) \right)^2,$$

$$\sigma_1^2 = \left( \frac{1}{N^+} \sum_{x \in X^+} f(x)^2 \right) - \left( \frac{1}{N^+} \sum_{x \in X^+} f(x) \right)^2$$
- Calcul du risque avec ces paramètres Gaussiens en utilisant l'équation Eq 8.1.

Nous évaluons cette approche sur le corpus de référence de TALN MetaEval et le corpus de référence de Machine Learning général PMLB.

### 9.1.1 Utilisation du risque non supervisé comme mesure de généralisation

#### Expériences et Résultats sur MetaEval

Le corpus de référence MetaEval introduit par SILEO et MOENS [171]<sup>1</sup> comporte une collection de 101 tâches des TALN pour effectuer du méta-apprentissage et de l'apprentissage multi-tâches, incluant plus de 50 tâches de classification binaire. La taille des jeux de données est variable entre quelques centaines d'exemples jusqu'à plusieurs milliers. Ces jeux de données sont :

- Glue introduit par WANG et al. [194] qui est un corpus de référence standard de compréhension du langage naturel (*Natural Language Understanding*, NLU), comportant néanmoins des données d'apprentissage limitées.
- Super-glue introduit par WANG et al. [195] un jeu de données destiné à poser un test de compréhension de langage plus rigoureux que GLUE. Nous en avons gardé deux tâches (boolq et wic) contenant respectivement 9000 et 5000 exemples d'apprentissage.
- Tweeteval introduit par BARBIERI et al. [9] qui consiste en sept tâches hétérogènes de classification spécifiques à Twitter. Nous n'en utilisons que 3 (hate, irony et offensive) ayant respectivement 9000, 3000 et 12000 exemples d'apprentissage.
- Ade (Adverse Drug Reaction Data) corpus v2 classification introduit par GURULINGAPPA et al. [72], un jeu de données pour classifier si une phrase est liée à l'ADE (Vrai) ou non (Faux) et qui contient 24000 exemples d'apprentissage.
- Rotten Tomatoes introduit par PANG et LEE [146], un jeu de données de reviews de films, contenant 9000 exemples d'apprentissage.
- HoVer introduit par JIANG et al. [95] un jeu de données pour l'extraction de preuves et la vérification de faits ayant 18k exemples d'apprentissage.
- Movie Rationales introduit par ZAIDAN, EISNER et PIATKO [210] un jeu de données contenant des explications de reviews de film écrites par des humains et possédant 2000 exemples d'apprentissage.
- Eraser multi RC introduit par KHASHABI et al. [107] un challenge de compréhension de lecture dans lequel les questions ne peuvent être répondues qu'en prenant en compte l'information de plusieurs phrases et qui a 24k exemples d'apprentissage.
- Le jeu de données Ethic introduit par HENDRYCKS et al. [79], corpus de référence ayant pour but de détecter des concepts s'étendant entre la justice, le "well-being", les devoirs, les vertus et la morale de bon sens. Nous n'utilisons que les data-sets Commonsense, Justice et Virtue composés respectivement de 13910, 21791 et 28245 exemples d'apprentissage et 3885, 2704 et 4975 exemples de validation.

1. <https://github.com/sileod/metaeval>

- Pragmaeval introduit par SILEO et al. [172], un corpus de référence d'évaluation de compréhension du langage naturel, duquel nous utilisons toutes les tâches binaires qui ont plusieurs tailles (toutes relativement faibles), entre 371 et 6000 exemples d'apprentissage.
- PAWS introduit par ZHANG, BALDRIDGE et HE [215], un relativement gros jeu de données de plus de 49k exemples d'entraînement pour identifier d'éventuelles paraphrases (le jeu de données disponible PAWS dans le corpus de référence metaeval étant le jeu de données  $PAWS_{Wiki}$ ).
- Crowdflower introduit par VAN PELT et SOROKIN [188] une collection de jeux de données issus de la plateforme CrowdFlower pour plusieurs tâches comme l'analyse de sentiments ou la classification d'actes de dialogue. Ces jeux de données ont des tailles entre 4000 et 5000 exemples d'apprentissage.
- BLiMP classification introduit par WARSTADT et al. [200] : un challenge pour évaluer quel modèle de langage reconnaît les phénomènes grammaticaux majeurs en anglais. Nous utilisons des jeux de données de tailles variables (entre 2000 et 52000 exemples d'entraînement).
- Recast introduit par POLIAK et al. [153] qui réutilise des jeux de données existants pour les reformuler sous forme de tâches inférentielles de NLU. Seuls les plus petits jeux de données ont été utilisés (entre 1000 et 38k exemples d'entraînement).

Nous utilisons la séparation entraînement/validation fournie pour chaque corpus et calculons le prior de class  $p_0$  pour chaque sous ensemble d'entraînement. Pour la métrique d'évaluation, nous utilisons ici l'exactitude standard, sauf pour CoLA jeu de données pour lequel nous utilisons le coefficient de corrélation de Matthews (MCC). Nous résumons les résultats Table 9.3 où nous reportons les métriques aux diverses époques d'intérêt, elles aussi indiquées. La table d'entraînement complète pour chaque data-set est fournie dans l'Annexe A.

Dans les expériences présentées ici, nous fine-tunons BERT sur de nombreux jeux de données de classification binaires de Metaeval. Le learning rate initial est de  $10^{-6}$  et nous entraînons BERT sur chaque tâche pendant 15 époques et calculons 3 mesures de "généralisation" : le coût d'entraînement (emp. risk), le coût de validation et le risque non supervisé (unsup. risk) à chaque époque, en considérant les 5 premières comme une phase de warm-up du modèle.

Ces hyper-paramètres ont été posés comme constants pour tous les jeux de données et ont été choisis de telle sorte que la courbe du coût d'entraînement tend à converger sur la plupart d'entre eux.

La phase de warm-up est nécessaire puisqu'initialement le modèle est trop loin de l'optimum empirique ce qui casserait l'hypothèse de l'équation Eq-8.1 comme montré Table 9.1. Selon chacune des métriques testées, les meilleurs modèles sont respectivement :

- le modèle à la dernière époque (15) qui minimise le coût empirique sur le corpus d'entraînement ;

Epoque	exactitude de Validation	Perf. sur le Corpus d'Entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.7659	0.5493	0.2980
Epoque 2	0.7795	0.4651	0.3063
Epoque 3	0.7810	0.4321	0.3246
Epoque 4	0.7893	0.4116	0.2778
Epoque 5	0.7885	0.3933	0.2758
Epoque 6	0.7832	0.3833	0.2748
Epoque 7	0.7878	0.3665	0.2725
Epoque 8	<b>0.7900</b>	0.3570	0.2726
Epoque 9	0.7863	0.3459	<b>0.2613</b>
Epoque 10	0.7813	0.3289	0.2676
Epoque 11	0.7795	0.3266	0.2976
Epoque 12	0.7810	0.3240	0.2709
Epoque 13	0.7778	0.3167	0.3067
Epoque 14	0.7742	0.3181	0.2867
Epoque 15	0.7719	<b>0.3141</b>	0.2876

**TABLE 9.1** – Corpus : Offensive,  $p_0 = 0.6693$  (relativement équilibré). Le modèle commence à surapprendre après l'époque 8 ; le risque proposé détecte l'époque 9 comme la meilleure (uniquement à partir du corpus d'entraînement), faisant une "erreur" d'une seule époque.

- le modèle à l'époque "**maxE**", qui minimise le coût de validation sur le corpus correspondant ;
- le modèle à l'époque "**riskE**", qui minimise le risque non supervisé sur le corpus d'entraînement.

Les exactitude de validation correspondantes à chacune de ces métriques, respectivement dénotées "**empACC**", "**maxACC**" et "**riskACC**", sont reportées Table 9.3. Pour illustrer les observations issues de cette Table, nous détaillons dans les Tables 9.1 et 9.2 les évolutions de ces mesures pendant l'entraînement pour deux différentes valeurs de  $p_0$  et ainsi pour deux différentes distributions des données. Pour ces deux jeux de données, le modèle commence à surapprendre après l'époque 8 et le risque non supervisé identifie les modèles aux meilleures performances de généralisation respectivement aux époque 9 et 8. Ceci illustre que même si la mesure proposée ne trouve pas exactement l'époque maximisant les performances sur le corpus de validation, elle identifie tout de même une époque proche, ce qui est très intéressant dans le cas où l'on voudrait utiliser cette mesure comme indication d'early-stopping sans corpus de validation, par exemple pour les langues faiblement dotées. Comme mentionné précédemment, nous fournissons toutes les tables correspondantes en Annexe A.

Epoque	exactitude de Validation	Perf. sur le Corpus d'Entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.8256	0.5556	0.4868
Epoque 2	0.8240	0.4615	0.4165
Epoque 3	0.8320	0.4337	0.4157
Epoque 4	0.8352	0.4095	0.3968
Epoque 5	0.8336	0.4006	0.3895
Epoque 6	0.8352	0.3830	0.3874
Epoque 7	0.8320	0.3562	0.3734
Epoque 8	<b>0.8384</b>	0.3554	<b>0.3622</b>
Epoque 9	0.8256	0.3377	0.3652
Epoque 10	0.8288	0.3341	0.3966
Epoque 11	0.8320	0.3291	0.3845
Epoque 12	0.8304	0.3206	0.3941
Epoque 13	0.8304	0.3118	0.4068
Epoque 14	0.8304	0.3057	0.4026
Epoque 15	0.8288	<b>0.3027</b>	0.4172

**TABLE 9.2** – Corpus : Political Media Audience,  $p_0 = 0.2099$  (assez déséquilibré). Le modèle commence à surapprendre après l'époque 8 qui est également prédite comme la meilleure par le risque proposé.

**Analyse de la Table 9.3 :** Par définition, la troisième colonne **maxACC** représente l'oracle, c'est-à-dire la borne supérieure d'exactitude sur le corpus de validation et nous surlignons en gras laquelle des deux autres mesures (**empACC** et **riskACC**) en est la plus proche.

Quand il y a une différence, pour 49% des jeux de données, c'est la mesure proposée ici qui est la plus proche de l'exactitude maximale tandis que le risque empirique l'emporte pour seulement 7% des cas. Ces 7% confirment ce qui a été observé dans la littérature et notamment par JIANG et al. [94], à savoir que le coût d'entraînement est fortement corrélé au saut de généralisation. Le plus intéressant dans ces résultats est que dans 93% des cas, le risque proposé est au moins aussi bon, si ce n'est meilleur pour la moitié des cas que le risque empirique et que en considérant une méthode d'early stopping basé sur notre proposition, nous réduisons en moyenne presque de moitié le saut de généralisation **maxACC - empACC** : plus précisément de 44%. Ces résultats suggèrent que le risque proposé pourrait être un critère d'early stopping intéressant pour le fine-tuning des LLMs permettant de se passer de corpus de validation.

Corpus	maxE	riskE	empACC	riskACC	maxACC
ethics/commonsense	14	13	0.8082	<b>0.8085</b>	0.8116
ethics/justice	13	13	0.7426	<b>0.7433</b>	0.7433
ethics/virtue	14	7	<b>0.8438</b>	0.8372	0.8444
pragmeval/emobank-arousal	13	15	0.7164	0.7164	0.7178
pragmeval/persuasiveness-eloquence	8	13	0.7253	<b>0.7363</b>	0.7912
pragmeval/persuasiveness-relevance	11	14	0.6703	0.6703	0.6703
pragmeval/persuasiveness-specificity	11	12	0.7097	<b>0.7258</b>	0.7258
pragmeval/persuasiveness-strength	8	8	0.8478	0.8478	0.8478
pragmeval/emobank-dominance	8	14	0.6817	<b>0.6842</b>	0.6942
pragmeval/squinky-implicature	6	12	0.7011	<b>0.7032</b>	0.7032
pragmeval/sarcasm	6	12	0.7313	0.7313	0.7335
pragmeval/squinky-formality	11	8	0.9801	0.9801	0.9845
pragmeval/squinky-informativeness	9	7	0.9161	<b>0.9204</b>	0.9247
pragmeval/emobank-valence	8	8	0.8509	<b>0.8618</b>	0.8618
paws/labeled final	12	14	<b>0.8748</b>	0.8733	0.8755
crowdflower/tweet global warming	10	15	0.8106	0.8106	0.8144
crowdflower/political-media-audience	8	8	0.8288	<b>0.8384</b>	0.8384
crowdflower/political-media-bias	8	6	0.7664	<b>0.7728</b>	0.7840
glue/cola	12	8	0.5868	0.5868	0.5923
glue/sst2	9	6	0.9037	<b>0.9083</b>	0.9128
glue/mrpc	12	12	0.7843	0.7843	0.7843
glue/qnli	7	7	0.8912	<b>0.8949</b>	0.8949
glue/rte	9	15	0.7029	0.7029	0.7101
blimp classification/syntax semantics	7	7	0.9991	0.9991	0.9991
blimp classification/syntax+semantics	11	12	0.996	0.996	0.996
blimp classification/morphology	8	10	0.9956	<b>0.9960</b>	0.9962
blimp classification/syntax	9	13	0.9946	0.9946	0.9948
recast/recast puns	13	10	<b>0.9498</b>	0.9475	0.9509
recast/recast factuality	9	12	0.9294	<b>0.9300</b>	0.9307
recast/recast verbnet	7	15	0.7343	0.7343	0.7343
recast/recast sentiment	8	15	0.9300	0.9300	0.9317
recast/recast megaveridicality	11	6	0.8739	<b>0.8765</b>	0.8782
super glue/boolq	13	13	0.6969	<b>0.7015</b>	0.7015
super glue/wic	9	13	0.6458	0.6458	0.6458
ade corpus v2/Ade corpus v2 classification	11	11	0.9449	<b>0.9479</b>	0.9479
tweeteval/hate	11	13	0.7530	<b>0.7570</b>	0.7570
tweeteval/irony	8	11	0.6660	<b>0.6733</b>	0.6764
tweeteval/offensive	8	9	0.7719	<b>0.7863</b>	0.7900
rotten tomatoes/default	8	15	0.8340	0.8340	0.8440
hover/default	15	15	0.6090	0.6090	0.6090
movie rationales/default	11	14	0.8300	<b>0.8500</b>	0.8500
eraser multi rc/default	10	15	0.6646	0.6646	0.6705

**TABLE 9.3** – Résultats sur MetaEval : **maxACC** est la borne supérieure d’exactitude (l’oracle), qui a lieu à l’époque **maxE**; **empACC** est l’exactitude lorsque le coût empirique est minimal, principalement à l’époque 15; **riskACC** est l’exactitude à l’époque **riskE**, qui est le critère proposé d’early stopping calculé sur le corpus d’entraînement. Les nombres en gras sont les plus proches de la borne supérieure.

## Expériences sur PMLB

Le corpus de référence de Machine Learning PMLB<sup>2</sup> contient 162 jeux de données, dont 75 pour de la classification binaires, que nous listons Table 9.4. Nous filtrons ces jeux de données et enlevons ceux relevant des critères suivants :

- **trop petits** : les jeux de données avec moins de 500 exemples, car l'intervalle de confiance résultant serait trop grand et les expériences sont également plus variables et dépendantes des conditions initiales aléatoires, ce qui empêche une interprétation fiable des résultats.
- **duplicatas** : Lorsque plus de 10% des exemples sont en double, nous ne considérons pas le jeu de données puisque cela crée des dépendances entre le corpus de train et de validation.
- **pas de gain possible** : Pour ces jeux de données, nous avons effectué une expérience complète et avons comparé le maximum d'exactitude de validation (oracle) atteint pendant l'entraînement avec l'exactitude de fin d'entraînement. Lorsque la différence entre les deux est plus faible que la moitié de l'intervalle de confiance de 95% de Agresti et Coull, il n'y a pas de possibilité claire d'amélioration du modèle avec une quelconque méthode tentant de dépasser le surapprentissage.

Nous donnons aux jeux de données restants un nom court en 2 lettres afin de rendre visibles les résultats obtenus avec une validation croisée à 5 plis reportés Table 9.5.

**TABLE 9.4** – Statistiques de jeux de données de PMLB

nom court	corpus	taille	notes
	GAMETES Epistasis 2 Way 1000atts 0.4H EDM 1 EDM 1 1	1600	pas de gain possible
g1	GAMETES Epistasis 2 Way 20atts 0.1H EDM 1 1	1600	
g2	GAMETES Epistasis 2 Way 20atts 0.4H EDM 1 1	1600	
g3	GAMETES Epistasis 3 Way 20atts 0.2H EDM 1 1	1600	
g4	GAMETES Heterogeneity 20atts 1600 Het 0.4 0.2 50 EDM 2 001	1600	
g5	GAMETES Heterogeneity 20atts 1600 Het 0.4 0.2 75 EDM 2 001	1600	
h1	Hill Valley with noise	1212	
h2	Hill Valley without noise	1212	
ad	adult	48842	
	agaricus lepiota	8145	100% exactitude
	anacatdata boxing1	120	trop petit
	anacatdata boxing2	132	trop petit
	anacatdata creditscore	100	trop petit
	anacatdata lawsuit	264	trop petit
	appendicitis	106	trop petit
au	australian	690	
	backache	180	trop petit
	biomed	209	trop petit
bc	breast	699	
	breast cancer	286	doublons
bw	breast cancer wisconsin	569	
	breast w	699	doublons
bx	buggyCrx	690	
	bupa	345	trop petit
	chess	3196	pas de gain possible

*Suite page suivante*

2. <https://github.com/EpistasisLab/pmlb>

nom court	corpus	taille	notes
ch	churn	5000	
	clean1	476	trop petit
	clean2	6598	pas de gain possible
	cleve	303	trop petit
	coil2000	9822	doublons
	colic	368	doublons
	corral	160	doublons
ca	credit a	690	
cg	credit g	1000	
cx	crx	690	
	dis	3772	pas de gain possible
ge	flare	1066	doublons
	german	1000	
	glass2	163	trop petit
	heart c	303	trop petit
	heart h	294	trop petit
	heart statlog	270	trop petit
	house votes 84	435	doublons
	hungarian	294	trop petit
	hypothyroid	3163	doublons
	ionosphere	351	trop petit
	irish	500	doublons
	kr vs kp	3196	pas de gain possible
	magic	19020	doublons
	mofn 3 7 10	1324	doublons
	molecular biology promoters	106	trop petit
	monk1	556	doublons
	monk2	601	doublons
monk3	554	doublons	
pi	mushroom	8124	pas de gain possible
	mux6	128	doublons
	parity5+5	1124	doublons
	phoneme	5404	doublons
	pima	768	
	prnn crabs	200	trop petit
	prnn synth	250	trop petit
pb	profb	672	
	ring	7400	
rg	saheart	462	trop petit
	sonar	208	trop petit
	spambase	4601	doublons
	spect	267	trop petit
	spectf	349	doublons
	threeOf9	512	pas de gain possible
	tic tac toe	958	pas de gain possible
	tokyo1	959	
tw	twonorm	7400	
	vote	435	doublons
w	wdbc	569	
	xd6	973	doublons

Nous nous sommes contentés d'une simple validation croisée dans ces expériences. Les colonnes Table 9.5 sont :

- "**maxACC**" est l'exactitude maximum sur le corpus de validation qui est atteinte à l'époque "**maxE**" et qui représente l'oracle.
- "**riskACC**" est l'exactitude obtenue en choisissant la meilleure époque proposée par notre risque non supervisé, l'époque "**riskE**". Cette valeur doit être comparée avec "**empACC**" : l'exactitude à la dernière époque d'entraînement (10,000).

- "**riskGain**" est la différence entre les valeurs de riskACC et empACC, et "**confint**" est l'intervalle de confiance à 95% d'Agresti et Coull.
- "**FM**" est le mode d'échec (*failure mode*) reporté après une analyse manuelle de la courbe d'entraînement considérée. Ceci est expliqué Section 9.1.2.

Tandis que "**maxACC**" estime l'early stopping sur le corpus de validation, notre proposition "**riskACC**" l'estime sur le corpus d'entraînement.

Le learning rate utilisé est de  $10^{-4}$  et le nombre d'époques est de 10000, valeurs une nouvelle fois choisies pour que la courbe de coût d'entraînement converge convenablement pour au moins quelques jeux de données. Le réseau entraîné est un MLP dense à 3 couches : la première et la dernière sont respectivement de la taille des entrées  $d$  (dépendant du jeu de données) et du nombre de classes (=2). La couche cachée a un nombre de neurones fixé à  $\sqrt{Nd}$  afin de donner au réseau une capacité suffisante pour mémoriser le jeu de données en suivant les résultats de ZHANG et al. [213], avec  $N$  le nombre d'exemples. L'optimiseur Adam a été choisi arbitrairement

**TABLE 9.5** – Résultats sur les jeux de données de classification binaire du corpus de référence PMLB.

corpus	maxACC	riskACC	empACC	riskGain	confint	maxE	riskE	Failure Mode
w	0.98241	<b>0.97187</b>	0.96482	0.00705	0.0118	318	1957	
tn	0.97986	<b>0.97716</b>	0.96595	0.01122	0.0032	222	192	
tk	0.93742	0.91032	<b>0.91450</b>	-0.00418	0.0155	229	3443	FM2
rg	0.97932	<b>0.97568</b>	0.96730	0.00838	0.00326	143	196	
pb	0.70531	0.62052	0.62052	0.00000	0.03446	606	9999	FM3
pi	0.80086	0.71621	0.71621	0.00000	0.02831	455	9999	FM3
ge	0.76700	0.70300	<b>0.70900</b>	-0.00600	0.02619	162	7794	FM2
g1	0.61062	<b>0.54813</b>	0.53312	0.01500	0.0239	147	6193	
g2	0.72437	<b>0.65750</b>	0.65500	0.00250	0.0219	123	7678	
g3	0.56812	0.54000	<b>0.54250</b>	-0.00250	0.0242	4346	6133	FM1
g4	0.66937	<b>0.60563</b>	0.59937	0.00625	0.0230	168	5531	
g5	0.71125	<b>0.61375</b>	0.61250	0.00125	0.0222	132	6032	
h1	0.95959	<b>0.84333</b>	0.84240	0.00093	0.0113	7911	9750	
h2	0.93894	<b>0.84739</b>	0.84408	0.00331	0.0136	5972	9960	
cx	0.89420	<b>0.81594</b>	0.81449	0.00145	0.02310	913	9985	
cg	0.74900	<b>0.69100</b>	0.68200	0.00900	0.02686	389	7619	
ca	0.88986	0.83043	<b>0.83188</b>	-0.00145	0.02350	394	9980	FM2
ch	0.93400	<b>0.91860</b>	0.91800	0.00060	0.00690	804	9650	
bx	0.87971	0.83188	0.83188	0.00000	0.02439	573	9999	FM2
bw	0.98067	<b>0.96660</b>	0.96483	0.00177	0.01224	279	3033	
bc	0.96854	<b>0.95853</b>	0.95281	0.00572	0.01346	363	222	
au	0.88986	0.80580	0.80580	0.00000	0.02350	333	9981	FM2
ad	0.85336	0.81604	0.81604	0.00000	0.00314	183	9999	FM3

Ces résultats montrent que choisir la meilleure époque en utilisant notre méthode, améliore les performances du modèle sur la majorité des jeux de données quand on compare avec le choix du modèle à la dernière époque. Cependant, il arrive que notre méthode dégrade quelque peu l'exactitude et nous étudions les raisons de ces échecs dans la partie suivante 9.1.2.

### 9.1.2 Étude des modes d'échecs

Dans cette section nous proposons une analyse des modes d'échecs (*Failure Modes*, FM) de notre approche sur les expériences précédentes.

Définissons d'abord quelques termes que nous utiliserons dans cette section. Le surapprentissage apparaît lorsque le modèle n'est plus capable de généraliser au-delà des exemples d'entraînement, c'est-à-dire lorsque ses performances diminuent significativement sur un corpus de test différent. Ce phénomène est à différencier de la mémorisation qui fait allusion à la capacité du modèle à discriminer les exemples qu'il a observés pendant l'apprentissage des autres comme mentionné par CARLINI et al. [21]. Nous savons qu'un modèle peut à la fois mémoriser et avoir de bonnes performances de généralisation comme montré par DUBOST et al. [47] et FELDMAN [54]. Nous avons identifié quatre principaux modes :

**FM 0 :** En définissant la mémorisation extrême qui apparaît lorsque le score de chaque exemple d'entraînement est très proche soit de 0 ou de 1 (pour la classification binaire), ceci mène à des distributions  $p(f_{\hat{\theta}}(x)|y)$  proches de Dirac autour de 0 et 1 ce qui serait approximé par deux Gaussiennes avec une très faible variance dans l'équation Eq 8.1, le risque serait alors très faible. À cause de cette limitation, il serait mieux de calculer le risque sur un corpus différent du corpus d'entraînement, ce qui est souvent possible en pratique car notre risque est non supervisé et les données sans labels sont beaucoup plus simples à obtenir que des données avec labels.

En pratique, nous n'avons pas observé une telle situation dans nos expériences (la distribution des scores était toujours relativement lisse) mais n'excluons pas son apparition dans des situations réelles. Quand d'autres données ne sont pas accessibles, il serait bon d'injecter du bruit aléatoire à hauteur de 10% dans les données pour prévenir ce cas dégénéré. Nous n'avons pas appliqué ces recommandations dans nos expériences pour garder le protocole le plus simple possible mais elles pourraient néanmoins se montrer utiles.

**FM1 :** Ce mode apparaît lorsque l'exactitude de test reste basse, proche du choix aléatoire et oscille autour de cette basse performance durant tout l'entraînement. Dans ce cas, aucun des modèles entraînés ne généralise réellement mieux que les autres et mesurer la capacité de généralisation ne sert donc à rien. Ceci apparaît par exemple sur les jeux de données emobank-arousal, persuasiveness-relevance et sarcasm data-sets issus de pragmeval dans Metaeval. Une option pour résoudre ce problème serait de directement utiliser le risque non supervisé pour fine-tuner le modèle et le forcer ainsi vers une meilleure généralisation.

**FM2** : Ce mode apparaît lorsque le modèle surapprend très rapidement, pendant les quelques premières époques. Dans ce cas, la valeur du risque non supervisé est imprévisible et la principale condition d'applicabilité est violée. En effet, à cause du caractère initialement aléatoire des poids du modèle, la distribution des scores est proche d'une Gaussienne simple plutôt que d'une bi-Gaussienne et l'équation eq 8.1 ne peut pas être estimée convenablement dans ces conditions. Ceci apparaît par exemple pour persuasiveness-strength, emobank-dominance et squinky-implicature, jeux de données pour lesquels la meilleure exactitude était atteinte avant la phase de warm-up de 5 époques, ce qui peut être vu dans les tables dédiées dans l'Annexe A.

**FM3** : L'entraînement a été stoppé trop tôt et le modèle n'a pas encore convergé, ce qui semble typiquement être le cas sur le jeu de données Hover.

L'analyse de ces modes d'échecs suggère qu'il existe de simples contre-mesures pour contrecarrer ces problèmes expérimentaux. En particulier, les modes FM1, FM2 et FM3 peuvent être évités en utilisant d'autres architectures, learning rates et nombres d'époques. Nous n'avons pas réalisé ces expériences, requérant un protocole expérimental dédié à chaque jeu de données, ce qui n'était pas le but de notre travail.

En effet, ici nous avons montré que le risque non supervisé pouvait être utilisé pour convenablement mesurer la généralisation d'un réseau. Nous allons donc l'utiliser dans la section suivante pour apporter la touche finale à notre travail et évaluer avec un ensemble de métriques variées les différences de généralisation entre deux types de réseaux aux architectures finales égales : les GrowNN et les FFNN.

## 9.2 Résultats expérimentaux

Dans cette section finale, nous complétons finalement les résultats présentés aux Chapitres 3 puis 6 avec une dernière mesure de généralisation présentée précédemment : le risque non supervisé. Pour ceci nous nous limitons donc à des tâches de classification binaire et présentons les résultats dans la suite. Le but de ces expériences n'est pas d'atteindre des résultats états de l'art sur les corpora considérés via une optimisation intensive des hyper-paramètres du réseau, mais plutôt de comparer les modèles standards et expansifs avec un ensemble fixe d'hyper-paramètres pour isoler l'effet du grossissement sur les performances du modèle et sa surface de coût.

### 9.2.1 Comparaison des Réseaux intermédiaires

Dans nos expériences et résultats, les modèles sont comparés à architectures égales. Nous remplaçons le réseau linéaire à une couche cachée (de 1024 neurones) composant la tête de classification de RoBERTa large par un réseau expansif progressivement en 3 étapes. Nous désignons ici par le terme "sous-modèle" tous les modèles construits pendant

le grossissement, c'est-à-dire possédant strictement moins de 1024 neurones dans la couche cachée de la tête de classification. Chacun de ces sous-modèles est défini par le nombre de neurones cachés de sa tête de classification et par son mode d'apprentissage : soit "standard" s'il s'agit d'un réseau à l'architecture fixe au cours de l'apprentissage, soit "expansif" dans le cas contraire. Le but ici sera de comparer les réseaux non seulement après entraînement mais également pendant. Notons que des réseaux non expansifs avec moins de 1024 neurones cachés sont donc également créés. Les sous modèles de même taille sont entraînés pendant un nombre total d'époques identique, pendant que les modèles standards sont entraînés depuis le début pour chaque taille (et pour un nombre d'époques dépendant de leur taille). Les modèles expansifs reprennent l'entraînement pour 10 époques entre chaque étape de grossissement. Ce processus itératif est illustré Figure 9.1. Cela nous permet donc d'étudier plusieurs critères de sélection de modèle à partir duquel la nouvelle étape de grossissement doit partir :

- le dernier modèle de l'étape d'entraînement précédente (désigné par "last")
- le modèle avec la meilleure exactitude de validation lors de l'étape d'entraînement précédente (désigné par "valid")
- le modèle avec le risque non supervisé le plus faible lors de l'étape d'entraînement précédent et (désigné par "risk")

Les métriques utilisées pour comparer les sous-modèles de même taille sont donc les coût empiriques et non supervisées évaluées sur le corpus d'entraînement, la mesure de flatness et l'exactitude de test.

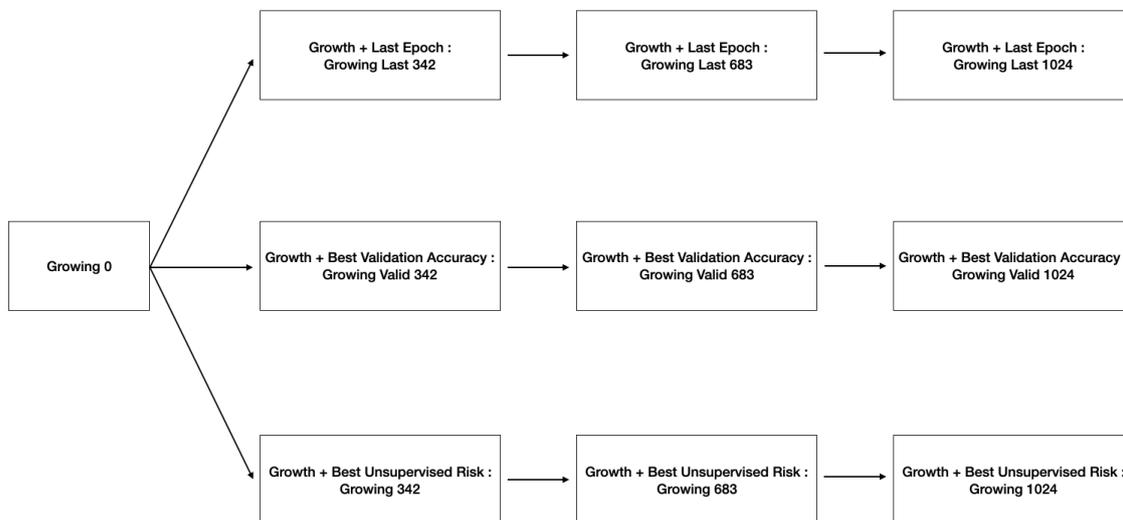


FIGURE 9.1 – Schéma de grossissement utilisé pour atteindre les modèles décrits.

Les résultats sont obtenus sur CoLA en utilisant le modèle RoBERTa-large comme structure du réseau. Il s'agit de noter que les résultats donnés ici sont tirés d'une expérience unique (ce qui explique les quelques différences avec les résultats du tableau 9.8), pouvant ainsi être peu représentatifs d'un comportement moyen et toute analyse est donc à prendre avec précautions. Cependant la dynamique mise en évidence est intéressante.

Modèle	Emp. Risk	Val. Acc.	Norme Spectrale	Unsup. Risk
Normal 0	1.002	85.5%	8.4e8	0.3003
Growing 0	1.017	85.5%	8.5e8	0.3002
Normal 342	<b>0.946</b>	85.2%	9.1e8	0.3321
Growing Last 342	1.66	83.2%	9e8	0.3319
Growing Valid 342	1.106	84.5%	8.8e8	0.3014
Growing Risk 342	1.09	<b>85.5%</b>	<b>8.6e8</b>	<b>0.3007</b>
Normal 683	<b>0.84</b>	<b>87.0%</b>	4.7e9	0.3338
Growing Last 683	1.586	84.6%	3e9	0.3338
Growing Valid 683	0.972	85.9%	1.4e9	0.3087
Growing Risk 683	0.923	86.4%	<b>1.1e9</b>	<b>0.3023</b>
Normal 1024	<b>0.98</b>	85.4%	1.9e10	0.3364
Growing Last 1024	1.36	85.3%	9.1e9	0.3344
Growing Valid 1024	1.33	85.9%	4.3e9	0.3193
Growing Risk 1024	1.10	<b>86.9%</b>	<b>3.8e9</b>	<b>0.3031</b>

**TABLE 9.6** – Évaluation des réseaux de tailles diverses proposés sur CoLA . Le risque empirique et non supervisé ainsi que la norme spectrale sont calculés sur le corpus de train. Le corps de l’architecture est RoBERTa large.

## 9.2.2 Comparaison des Réseaux finaux

Les résultats obtenus sont rapportés Tables 9.7 et 9.8 pour deux réseaux différents, de tailles différentes puisque l’un est la version distillée de la version de base de l’autre. Nous analyserons ces résultats au regard du travail présenté dans ce manuscrit Section 9.3

Corpus	Modèle	Emp. Risk	Val. Acc.	Norme Spectrale	U. Risk
<b>Commonsense</b>	Normal	<b>0.25 ± 0.03</b>	0.726 ± 0.004	$(8.3 ± 0.4) × 10^9$	0.20 ± 0.02
	Growing Last	0.31 ± 0.03	0.735 ± 0.004	$(5.3 ± 0.3) × 10^9$	0.17 ± 0.02
	Growing Valid	0.33 ± 0.02	0.73 ± 0.01	$(1.5 ± 0.4) × 10^9$	0.18 ± 0.04
	Growing Risk	0.32 ± 0.01	<b>0.74 ± 0.03</b>	<b><math>(1.3 ± 0.4) × 10^9</math></b>	<b>0.09 ± 0.01</b>
<b>CoLA</b>	Normal	<b>0.978 ± 0.002</b>	0.854 ± 0.005	$(1.7 ± 0.5) × 10^{10}$	0.33 ± 0.02
	Grow Last	1.359 ± 0.003	0.852 ± 0.002	$(9.2 ± 0.2) × 10^9$	0.33 ± 0.02
	Grow Valid	1.329 ± 0.004	0.858 ± 0.001	$(4.3 ± 0.1) × 10^9$	0.31 ± 0.01
	Grow Risk	1.107 ± 0.001	<b>0.869 ± 0.001</b>	<b><math>(3.6 ± 0.3) × 10^9</math></b>	<b>0.30 ± 0.01</b>

**TABLE 9.8** – Évaluation des réseaux proposés sur certains corpora de Metaeval. Le risque empirique et non supervisé ainsi que la norme spectrale sont calculés sur le corpus de train. Le corps de l’architecture est RoBERTa large.

Corpus	Modèle	Emp. Risk	Val. Acc.	Norme Spectrale	Unsup. Risk
Pers.-Eloquence	Normal	<b>0.14 ± 0.02</b>	0.7555 ± 0.0005	$(1.60 ± 0.58) × 10^9$	0.45 ± 0.02
	Grow Last	0.25 ± 0.03	0.755 ± 0.005	$(3.3 ± 0.2) × 10^8$	0.46 ± 0.02
	Grow Valid	0.24 ± 0.04	0.760 ± 0.005	$(1.5 ± 0.4) × 10^8$	0.45 ± 0.01
	Grow Risk	0.15 ± 0.01	<b>0.767 ± 0.001</b>	<b><math>(1.2 ± 0.3) × 10^7</math></b>	<b>0.31 ± 0.03</b>
Pers.-Specificity	Normal	<b>0.24 ± 0.01</b>	0.814 ± 0.006	$(1.88 ± 0.14) × 10^9$	0.23 ± 0.05
	Grow Last	0.340 ± 0.007	<b>0.823 ± 0.003</b>	$(2.15 ± 0.07) × 10^7$	<b>0.12 ± 0.04</b>
	Grow Valid	0.26 ± 0.02	0.822 ± 0.004	<b><math>(1.14 ± 0.52) × 10^7</math></b>	0.14 ± 0.01
	Grow Risk	0.360 ± 0.004	0.816 ± 0.002	$(3.64 ± 0.22) × 10^7$	0.21 ± 0.07
CoLA	Normal	<b>0.11 ± 0.02</b>	0.57 ± 0.02	$(4.61 ± 0.05) × 10^9$	0.41 ± 0.02
	Grow Last	0.15 ± 0.01	0.57 ± 0.02	$(2.35 ± 0.03) × 10^9$	0.39 ± 0.04
	Grow Valid	0.120 ± 0.005	<b>0.60 ± 0.01</b>	<b><math>(2.03 ± 0.09) × 10^8</math></b>	<b>0.317 ± 0.012</b>
	Grow Risk	0.13 ± 0.02	0.594 ± 0.016	$(7.65 ± 0.31) × 10^8$	0.33 ± 0.02
Emobank-Dom.	Normal	<b>0.29 ± 0.02</b>	0.63 ± 0.01	$(9.8 ± 0.5) × 10^8$	0.29 ± 0.04
	Grow Last	0.53 ± 0.04	0.62 ± 0.02	$(1.07 ± 0.30) × 10^9$	0.38 ± 0.03
	Grow Valid	0.35 ± 0.01	<b>0.69 ± 0.02</b>	<b><math>(6.22 ± 0.18) × 10^8</math></b>	<b>0.21 ± 0.04</b>
	Grow Risk	0.412 ± 0.015	0.64 ± 0.01	$(6.7 ± 0.3) × 10^8$	0.27 ± 0.02
Justice	Normal	<b>0.23 ± 0.01</b>	<b>0.57 ± 0.01</b>	$(2.1 ± 0.5) × 10^9$	0.16 ± 0.05
	Grow Last	0.35 ± 0.01	0.55 ± 0.01	<b><math>(1.22 ± 0.14) × 10^9</math></b>	0.12 ± 0.01
	Grow Valid	0.35 ± 0.07	0.56 ± 0.03	$(1.36 ± 0.12) × 10^9$	0.11 ± 0.02
	Grow Risk	0.36 ± 0.04	0.56 ± 0.02	$(1.78 ± 0.31) × 10^9$	<b>0.10 ± 0.03</b>
Offensive	Normal	<b>0.06 ± 0.01</b>	0.79 ± 0.01	$(3.02 ± 0.31) × 10^9$	0.420 ± 0.013
	Grow Last	0.19 ± 0.02	0.80 ± 0.01	$(4.78 ± 0.17) × 10^8$	0.37 ± 0.02
	Grow Valid	0.11 ± 0.01	<b>0.845 ± 0.014</b>	<b><math>(8.50 ± 0.09) × 10^7</math></b>	<b>0.22 ± 0.01</b>
	Grow Risk	0.17 ± 0.03	0.84 ± 0.02	$(9.3 ± 0.2) × 10^7$	0.25 ± 0.02
Virtue	Normal	<b>0.085 ± 0.005</b>	0.68 ± 0.01	$(2.40 ± 0.15) × 10^9$	1.04 ± 0.02
	Grow Last	0.27 ± 0.01	0.73 ± 0.02	$(2.40 ± 0.34) × 10^8$	0.57 ± 0.03
	Grow Valid	0.30 ± 0.02	<b>0.80 ± 0.03</b>	<b><math>(3.50 ± 0.37) × 10^7</math></b>	<b>0.34 ± 0.02</b>
	Grow Risk	0.23 ± 0.04	0.76 ± 0.02	$(9.30 ± 0.11) × 10^8$	0.40 ± 0.05

**TABLE 9.7** – Évaluation des réseaux proposés sur des corporas de Metaeval. Le risque empirique et non supervisé ainsi que la norme spectrale sont calculés sur le corpus de train. Le corps de l’architecture est DistilRoBeRTa base.

### 9.3 Discussion

Dans ce travail nous avons finalement rapproché une dernière mesure de généralisation en utilisant l’approximation du risque du classifieur binaire développé Chapitre 8 qui est théoriquement moins sensible au surapprentissage que le risque empirique standard. Les résultats présentés au Chapitre 6 montrent que les GrowNN atteignent effectivement des minima plus plats et les compléments apportés dans ce chapitre montrent que les minima ainsi atteints semblent permettre de meilleures performances de généralisation.

En effet, à part pour le corpus Justice pour lequel les résultats sur chaque métrique sont très proches les uns des autres quelque soit le modèle, les résultats exposés Table 9.7 montrent que :

- le risque empirique et l’approximation non supervisée du risque théorique du classifieur

mesurent bien deux risques différents : le risque empirique est toujours meilleur pour le modèle statique que pour les modèles expansifs, ce qui peut intuitivement s'expliquer par le temps plus grand accordé aux modèles standards pour le minimiser tandis que les modèles expansifs récupèrent des nouveaux paramètres initialisés aléatoirement de manière régulière.

- la plupart des modèles expansifs ont de meilleures performances sur le risque non supervisé. Cela peut mener à la question de savoir si grossir un réseau de neurones peut être interprété comme une forme de régularisation. Nous ne prétendons pas ici répondre à cette question qui peut être une piste de recherche extrêmement intéressante, cependant nous la modulons au regard du fait que le modèle standard a également été entraîné avec les meilleures pratiques en termes de régularisation (L2 et dropout) choisies par les auteurs des différents modèles utilisés.
- le risque non supervisé le plus bas atteint par tout modèle sembler corrélér largement avec les minima flat mesurés par la norme spectrale.
- le paradigme simple de grossissement que nous avons développé, à savoir choisir le meilleur modèle par rapport à une métrique choisie (exactitude de validation ou risque non supervisé) puis le faire grossir semble donner de meilleurs résultats que simplement faire grossir le dernier modèle (c'est-à-dire celui ayant connu le plus d'époques d'entraînement dans sa forme considérée).
- la meilleure métrique à choisir pour grossir semble être l'exactitude de validation ce qui peut en partie être expliqué par le fait qu'il s'agit également du critère final d'évaluation, les modèles "growing valid" pouvant donc être considérés comme agissant comme des oracles au regard de cette métrique particulière.
- en pratique, les bons résultats obtenus en augmentant la taille en se basant sur le risque non supervisé sont extrêmement intéressants car ils ne requièrent aucun corpus de validation.

Notre approche proposée a cependant deux limitations fondamentales.

Tout d'abord, le processus de grossissement introduit un hyperparamètre supplémentaire comparé au processus standard qui est la fréquence d'agrandissement du modèle. Cet hyper-paramètre devrait être optimisé en complément des hyper-paramètres standards, ce qui pourrait augmenter le temps de développement. Cependant, cet hyper-paramètre pourrait se révéler relativement simple à tuner puisque nous observons qu'une équi-répartition des étapes de grossissement à l'intérieur d'un nombre standard d'époques donne des résultats raisonnables dans nos expériences.

Deuxièmement, la procédure précise de grossissement doit être adaptée à chaque type de réseaux de neurones et à chaque procédure d'entraînement ; par exemple dans le cas déjà mentionné des réseaux MoE par SHAZEER et al. [170] qui utilisent des architectures très parcimonieuse plutôt que de FFNN denses standards. Des algorithmes avancés de NAS pourrait être utilisés pour construire le modèle, mais dans ce cas le processus de grossissement devrait être adapté aux spécificités du système considéré ici, par exemple pour décider quelle partie du réseau grossir (la modèle de porte et/ou chaque expert ou même l'ajout d'un nouvel expert par exemple) et pour assurer une interaction optimisée entre le processus de grossissement et l'algorithme de NAS.

# CONCLUSION

Nous avons ici établi de nouvelles connections entre les réseaux de neurones grossissants, les mesures de planéité et une approximation du risque du classifieur se concentrant sur la généralisation, et avons de plus montré que les GrowNN en plus d'atteindre des minima plus plats, semblent atteindre de meilleures performances et généralisent donc mieux grâce à leur manière originale d'explorer l'espace des paramètres.

Nous avons prouvé que le mécanisme de grossissement en lui-même avait un impact déterminant sur la surface de coût autour du minimum et en développant un algorithme de grossissement simple, soumis au minimum de contraintes afin d'introduire le moins de biais humain possible, nous avons confirmé expérimentalement cette découverte.

Nous avons également développé un risque non supervisé pouvant être appliqué à tout classifieur binaire, incluant des réseaux de DL. Nous avons montré que ce risque permet de régulariser les réseaux après entraînement en séparant les deux gaussiennes de répartition des classes ce qui améliore les performances sans introduire de nouvelles connaissances, en supposant les classifieurs proches de leurs optima. Il peut également servir de mesure de généralisation non supervisée, ce qui permet d'une part de disposer d'un ensemble varié de métriques d'évaluation, et d'autre part de l'utiliser comme critère d'early stopping sans corpus de validation ce qui se révèle particulièrement intéressant pour des langages faiblement dotés.

Nous avons de plus montré empiriquement que ce risque corrèle avec la planéité des optima et le saut de généralisation entre corpus d'entraînement et de test. À l'aide du schéma de grossissement correct, les GrowNN atteignent des minima plus plats avec un risque non supervisé plus faible et souvent de meilleures performances sur un corpus de test que leurs contreparties statiques. Nous espérons que ce travail peut donner une vision nouvelle sur l'étude des propriétés des DNN et ouvrir des perspectives intéressantes à explorer dans les études de la généralisation, des réseaux de neurones à architecture dynamique et des approximations des risques théoriques des classifieurs.

Le rythme de progression des applications et de la taille des LLMs est à l'heure actuelle extrêmement impressionnant, avec un retard de la compréhension théorique de ces modèles. Ainsi, on observe des phénomènes émergents de la part des modèles les plus perfectionnés, capables par exemple de suivre une chaîne de raisonnement relativement complexe [201], sans pour autant comprendre l'origine de ces phénomènes. Des travaux très récents montrent également que l'étendu du champ de connaissances des LLMs est encore très mal compris [216]. Il apparaît donc essentiel de comprendre les mécanismes qui régissent l'apprentissage des LLMs et d'avancer encore sur le volet théorique. En effet, les efforts de calcul nécessaires pour apprendre et utiliser ces réseaux sont devenus prohibitifs pour de nombreuses organisations et la récente introduction de LLaMA [185] semble montrer qu'une alternative moins coûteuse aux modèles toujours plus gros est envisageable. Des recherches théoriques sur les LLMs et leurs représentations du monde pourraient ainsi aller dans cette voie. Il serait alors intéressant d'étudier si les LLMs apprennent une structure sous-jacente du langage et si les modèles multi-modaux (MLLMs) comme le récent Kosmos-1 [88] ont une représentation du monde cohérente. Ainsi, en admettant par exemple que le langage admet une représentation sous forme de manifold (qui serait dynamique puisque le langage évolue au cours du temps), il pourrait être intéressant d'étudier si les représentations apprises par les LLMs peuvent approcher cette structure.



# BIBLIOGRAPHIE

- [1] Armen AGHAJANYAN et al. “Better Fine-Tuning by Reducing Representational Collapse”. In : *International Conference on Learning Representations*. 2021. URL : <https://openreview.net/forum?id=OQ08SN70M1V>.
- [2] Maksym ANDRIUSHCHENKO et al. “A modern look at the relationship between sharpness and generalization”. In : *arXiv e-prints*, arXiv:2302.07011 (fév. 2023), arXiv:2302.07011. DOI : [10.48550/arXiv.2302.07011](https://doi.org/10.48550/arXiv.2302.07011). arXiv : [2302.07011 \[cs.LG\]](https://arxiv.org/abs/2302.07011).
- [3] Sanjeev ARORA et al. “Stronger generalization bounds for deep nets via a compression approach”. In : *arXiv e-prints*, arXiv:1802.05296 (fév. 2018), arXiv:1802.05296. DOI : [10.48550/arXiv.1802.05296](https://doi.org/10.48550/arXiv.1802.05296). arXiv : [1802.05296 \[cs.LG\]](https://arxiv.org/abs/1802.05296).
- [4] Peter AUER, Mark HERBSTER et Manfred K. K WARMUTH. “Exponentially many local minima for single neurons”. In : *Advances in Neural Information Processing Systems*. Sous la dir. de D. TOURETZKY, M.C. MOZER et M. HASSELMO. T. 8. MIT Press, 1995. URL : <https://proceedings.neurips.cc/paper/1995/file/3806734b256c27e41ec2c6bffa26d9e7-Paper.pdf>.
- [5] Jimmy BA et Rich CARUANA. “Do Deep Nets Really Need to be Deep?” In : *Advances in Neural Information Processing Systems*. Sous la dir. de Z. GHAHRAMANI et al. T. 27. Curran Associates, Inc., 2014. URL : <https://proceedings.neurips.cc/paper/2014/file/ea8fcd92d59581717e06eb187f10666d-Paper.pdf>.
- [6] Dzmitry BAHDANAU, Kyunghyun CHO et Yoshua BENGIO. “Neural machine translation by jointly learning to align and translate”. In : *arXiv preprint arXiv:1409.0473* (2014).
- [7] K. BALASUBRAMANIAN, P. DONMEZ et G. LEBANON. “Unsupervised Supervised Learning II: Margin-Based Classification Without Labels”. In : *JMLR* 12 (2011), p. 3119-3145.
- [8] Dana H BALLARD. “Modular learning in neural networks.” In : *Aaai*. T. 647. 1987, p. 279-284.
- [9] Francesco BARBIERI et al. “TweetEval: Unified Benchmark and Comparative Evaluation for Tweet Classification”. In : *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online : Association for Computational Linguistics, nov. 2020, p. 1644-1650. DOI : [10.18653/v1/2020.findings-emnlp.148](https://doi.org/10.18653/v1/2020.findings-emnlp.148). URL : <https://aclanthology.org/2020.findings-emnlp.148>.
- [10] A. R BARRON. “Approximation and estimation bounds for artificial neural networks”. In : *Machine learning* 14.1 (1994), p. 115-133.

- [11] Peter BARTLETT, Dylan J. FOSTER et Matus TELGARSKY. “Spectrally-normalized margin bounds for neural networks”. In : *arXiv e-prints*, arXiv:1706.08498 (juin 2017), arXiv:1706.08498. DOI : [10 . 48550 / arXiv . 1706 . 08498](https://doi.org/10.48550/arXiv.1706.08498). arXiv : [1706 . 08498](https://arxiv.org/abs/1706.08498) [cs.LG].
- [12] Arash BEHBOODI, Gabriele CESA et Taco COHEN. “A PAC-Bayesian Generalization Bound for Equivariant Networks”. In : *arXiv e-prints*, arXiv:2210.13150 (oct. 2022), arXiv:2210.13150. DOI : [10 . 48550 / arXiv . 2210 . 13150](https://doi.org/10.48550/arXiv.2210.13150). arXiv : [2210 . 13150](https://arxiv.org/abs/2210.13150) [cs.LG].
- [13] Colin BELLINGER et al. “Sampling a longer life: Binary versus one-class classification revisited”. In : *First International Workshop on Learning with Imbalanced Domains: Theory and Applications*. 2017, p. 64-78.
- [14] James BERGSTRA et Yoshua BENGIO. “Random Search for Hyper-Parameter Optimization”. In : *J. Mach. Learn. Res.* 13.null (2012), p. 281-305. ISSN : 1532-4435.
- [15] Davis BLALOCK et al. “What is the State of Neural Network Pruning?” In : *arXiv e-prints*, arXiv:2003.03033 (mars 2020), arXiv:2003.03033. DOI : [10 . 48550 / arXiv . 2003 . 03033](https://doi.org/10.48550/arXiv.2003.03033). arXiv : [2003 . 03033](https://arxiv.org/abs/2003.03033) [cs.LG].
- [16] Hervé BOURLARD et Yves KAMP. “Auto-association by multilayer perceptrons and singular value decomposition”. In : *Biological cybernetics* 59.4 (1988), p. 291-294.
- [17] Tom B. BROWN et al. “Language Models are Few-Shot Learners”. In : *arXiv e-prints*, arXiv:2005.14165 (mai 2020), arXiv:2005.14165. DOI : [10 . 48550 / arXiv . 2005 . 14165](https://doi.org/10.48550/arXiv.2005.14165). arXiv : [2005 . 14165](https://arxiv.org/abs/2005.14165) [cs.CL].
- [18] Cristian BUCILUĂ, Rich CARUANA et Alexandru NICULESCU-MIZIL. “Model Compression”. In : *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '06. Philadelphia, PA, USA : Association for Computing Machinery, 2006, p. 535-541. ISBN : 1595933395. DOI : [10 . 1145 / 1150402 . 1150464](https://doi.org/10.1145/1150402.1150464). URL : <https://doi.org/10.1145/1150402.1150464>.
- [19] Han CAI et al. “Efficient Architecture Search by Network Transformation”. In : *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI'18/IAAI'18/EAAI'18. New Orleans, Louisiana, USA : AAAI Press, 2018. ISBN : 978-1-57735-800-8.
- [20] Han CAI et al. “Path-Level Network Transformation for Efficient Architecture Search”. In : *arXiv e-prints*, arXiv:1806.02639 (juin 2018), arXiv:1806.02639. DOI : [10 . 48550 / arXiv . 1806 . 02639](https://doi.org/10.48550/arXiv.1806.02639). arXiv : [1806 . 02639](https://arxiv.org/abs/1806.02639) [cs.LG].
- [21] Nicholas CARLINI et al. “The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks”. In : *Proceedings of the 28th USENIX Conference on Security Symposium*. SEC'19. Santa Clara, CA, USA : USENIX Association, 2019, p. 267-284. ISBN : 9781939133069.

- [22] Raghavendra CHALAPATHY, Aditya Krishna MENON et Sanjay CHAWLA. “Anomaly detection using one-class neural networks”. In : *CoRR* cs.LG/1802.06360v2 (2018).
- [23] P. CHAUDHARI et al. “Entropy-sgd: Biasing gradient descent into wide valleys”. In : *Journal of Statistical Mechanics: Theory and Experiment* 2019.12 (2019), p. 124018.
- [24] Beidi CHEN et al. “SLIDE : In Defense of Smart Algorithms over Hardware Acceleration for Large-Scale Deep Learning Systems”. In : *arXiv e-prints*, arXiv:1903.03129 (mars 2019), arXiv:1903.03129. DOI : [10.48550/arXiv.1903.03129](https://doi.org/10.48550/arXiv.1903.03129). arXiv : [1903.03129](https://arxiv.org/abs/1903.03129) [[cs.DC](#)].
- [25] Liang-Chieh CHEN et al. “Searching for Efficient Multi-Scale Architectures for Dense Image Prediction”. In : *arXiv e-prints*, arXiv:1809.04184 (sept. 2018), arXiv:1809.04184. DOI : [10.48550/arXiv.1809.04184](https://doi.org/10.48550/arXiv.1809.04184). arXiv : [1809.04184](https://arxiv.org/abs/1809.04184) [[cs.CV](#)].
- [26] Sanyuan CHEN et al. “Recall and Learn: Fine-tuning Deep Pretrained Language Models with Less Forgetting”. In : *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online : Association for Computational Linguistics, nov. 2020, p. 7870-7881. DOI : [10.18653/v1/2020.emnlp-main.634](https://doi.org/10.18653/v1/2020.emnlp-main.634). URL : <https://aclanthology.org/2020.emnlp-main.634>.
- [27] Tianqi CHEN, Ian GOODFELLOW et Jonathon SHLENS. “Net2Net: Accelerating Learning via Knowledge Transfer”. In : *arXiv e-prints*, arXiv:1511.05641 (nov. 2015), arXiv:1511.05641. arXiv : [1511.05641](https://arxiv.org/abs/1511.05641) [[cs.LG](#)].
- [28] Jianpeng CHENG, Li DONG et Mirella LAPATA. “Long short-term memory-networks for machine reading”. In : *arXiv preprint arXiv:1601.06733* (2016).
- [29] Krishna Teja CHITTY-VENKATA et al. “Neural Architecture Search for Transformers: A Survey”. In : *IEEE Access* 10 (2022), p. 108374-108412. DOI : [10.1109/ACCESS.2022.3212767](https://doi.org/10.1109/ACCESS.2022.3212767).
- [30] Jason PC CHIU et Eric NICHOLS. “Named entity recognition with bidirectional LSTM-CNNs”. In : *Transactions of the association for computational linguistics* 4 (2016), p. 357-370.
- [31] Kyunghyun CHO et al. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In : *arXiv preprint arXiv:1406.1078* (2014).
- [32] Minhyung CHO et Jaehyung LEE. “Riemannian approach to batch normalization”. In : *arXiv e-prints*, arXiv:1709.09603 (sept. 2017), arXiv:1709.09603. DOI : [10.48550/arXiv.1709.09603](https://doi.org/10.48550/arXiv.1709.09603). arXiv : [1709.09603](https://arxiv.org/abs/1709.09603) [[cs.LG](#)].
- [33] A. CHOROMANSKA et al. “The loss surfaces of multilayer networks”. In : *Artificial intelligence and statistics*. PMLR. 2015, p. 192-204.
- [34] Aakanksha CHOWDHERY et al. “PaLM: Scaling Language Modeling with Pathways”. In : *arXiv e-prints*, arXiv:2204.02311 (avr. 2022), arXiv:2204.02311. DOI : [10.48550/arXiv.2204.02311](https://doi.org/10.48550/arXiv.2204.02311). arXiv : [2204.02311](https://arxiv.org/abs/2204.02311) [[cs.CL](#)].

- [35] G.W. COTTRELL et al. *Image Compression by Back Propagation: An Example of Extensional Programming*. ICS report. Institute for Cognitive Science, University of California, San Diego, 1987. URL : <https://books.google.fr/books?id=5N-vHAAACAAJ>.
- [36] George CYBENKO. “Approximation by superpositions of a sigmoidal function”. In : *Mathematics of control, signals and systems 2.4* (1989), p. 303-314.
- [37] X. DAI, H. YIN et N. K JHA. “Grow and prune compact, fast, and accurate LSTMs”. In : *IEEE Transactions on Computers* 69.3 (2019), p. 441-452.
- [38] X. DAI, H. YIN et N. K JHA. “NeST: A neural network synthesis tool based on a grow-and-prune paradigm”. In : *IEEE Transactions on Computers* 68.10 (2019), p. 1487-1497.
- [39] Amit DANIELY. “SGD learns the conjugate kernel class of the network”. In : *Advances in Neural Information Processing Systems* 30 (2017).
- [40] Yann DAUPHIN et al. “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization”. In : *arXiv e-prints*, arXiv:1406.2572 (juin 2014), arXiv:1406.2572. DOI : [10.48550/arXiv.1406.2572](https://doi.org/10.48550/arXiv.1406.2572). arXiv : [1406.2572 \[cs.LG\]](https://arxiv.org/abs/1406.2572).
- [41] J. DEVLIN et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In : *NAACL-HLT (1)*. 2019.
- [42] Rahul DEY et Fathi M SALEM. “Gate-variants of gated recurrent unit (GRU) neural networks”. In : *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*. IEEE. 2017, p. 1597-1600.
- [43] Shifei DING et al. “Evolutionary artificial neural networks: a review”. In : *Artificial Intelligence Review* 39.3 (2013), p. 251-260. ISSN : 1573-7462. DOI : [10.1007/s10462-011-9270-6](https://doi.org/10.1007/s10462-011-9270-6). URL : <https://doi.org/10.1007/s10462-011-9270-6>.
- [44] L. DINH et al. “Sharp minima can generalize for deep nets”. In : *ICML*. PMLR. 2017, p. 1019-1028.
- [45] Jeffrey DONAHUE et al. “Long-term recurrent convolutional networks for visual recognition and description”. In : *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, p. 2625-2634.
- [46] Li DONG et al. “Unified Language Model Pre-training for Natural Language Understanding and Generation”. In : *arXiv e-prints*, arXiv:1905.03197 (mai 2019), arXiv:1905.03197. arXiv : [1905.03197 \[cs.CL\]](https://arxiv.org/abs/1905.03197).
- [47] Florian DUBOST et al. “Double Descent Optimization Pattern and Aliasing: Caveats of Noisy Labels”. In : *ArXiv abs/2106.02100* (2021). eprint : [2106.02100v2](https://arxiv.org/abs/2106.02100).

- [48] Sannara EK et al. “A Federated Learning Aggregation Algorithm for Pervasive Computing: Evaluation and Comparison”. In : *2021 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. Los Alamitos, CA, USA : IEEE Computer Society, mars 2021, p. 1-10. DOI : [10.1109/PERCOM50583.2021.9439129](https://doi.org/10.1109/PERCOM50583.2021.9439129). URL : <https://doi.ieeecomputersociety.org/10.1109/PERCOM50583.2021.9439129>.
- [49] Ronen ELDAN et Ohad SHAMIR. “The power of depth for feedforward neural networks”. In : *Conference on learning theory*. PMLR. 2016, p. 907-940.
- [50] T. ELSKEN, J. H. METZEN et F. HUTTER. “Neural architecture search: A survey”. In : *The Journal of Machine Learning Research* 20.1 (2019), p. 1997-2017.
- [51] Thomas ELSKEN, Jan HENDRIK METZEN et Frank HUTTER. “Efficient Multi-objective Neural Architecture Search via Lamarckian Evolution”. In : *arXiv e-prints*, arXiv:1804.09081 (avr. 2018), arXiv:1804.09081. DOI : [10.48550/arXiv.1804.09081](https://doi.org/10.48550/arXiv.1804.09081). arXiv : [1804.09081 \[stat.ML\]](https://arxiv.org/abs/1804.09081).
- [52] Dumitru ERHAN et al. “Why Does Unsupervised Pre-training Help Deep Learning?” In : *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Sous la dir. d’Yee Whye TEH et Mike TITTERINGTON. T. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy : PMLR, 2010, p. 201-208. URL : <https://proceedings.mlr.press/v9/erhan10a.html>.
- [53] Utku EVCI et al. “GradMax: Growing Neural Networks using Gradient Information”. In : *arXiv e-prints*, arXiv:2201.05125 (jan. 2022), arXiv:2201.05125. arXiv : [2201.05125 \[cs.LG\]](https://arxiv.org/abs/2201.05125).
- [54] Vitaly FELDMAN. “Does Learning Require Memorization? A Short Tale about a Long Tail”. In : *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. New York, NY, USA : Association for Computing Machinery, 2020, 954–959. ISBN : 9781450369794. URL : <https://doi.org/10.1145/3357713.3384290>.
- [55] Santiago FERNÁNDEZ, Alex GRAVES et Jürgen SCHMIDHUBER. “An application of recurrent neural networks to discriminative keyword spotting”. In : *International Conference on Artificial Neural Networks*. Springer. 2007, p. 220-229.
- [56] Matthias FEURER et Frank HUTTER. “Hyperparameter Optimization”. In : *Automated Machine Learning: Methods, Systems, Challenges*. Sous la dir. de Frank HUTTER, Lars KOTTHOFF et Joaquin VANSCHOREN. Cham : Springer International Publishing, 2019, p. 3-33. ISBN : 978-3-030-05318-5. DOI : [10.1007/978-3-030-05318-5\\_1](https://doi.org/10.1007/978-3-030-05318-5_1). URL : [https://doi.org/10.1007/978-3-030-05318-5\\_1](https://doi.org/10.1007/978-3-030-05318-5_1).
- [57] Benjamin FISH et Lev REYZIN. “On the complexity of learning from label proportions”. In : *arXiv preprint arXiv:2004.03515* (2020).
- [58] P. FORET et al. “Sharpness-aware Minimization for Efficiently Improving Generalization”. In : *ICLR*. 2021. URL : <https://openreview.net/forum?id=6Tm1mposlRM>.

- [59] Jonathan FRANKLE et Michael CARBIN. “The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks”. In : *arXiv e-prints*, arXiv:1803.03635 (mars 2018), arXiv:1803.03635. DOI : [10 . 48550 / arXiv . 1803 . 03635](https://doi.org/10.48550/arXiv.1803.03635). arXiv : [1803 . 03635](https://arxiv.org/abs/1803.03635) [cs.LG].
- [60] Antoine GAUTIER, Quynh N NGUYEN et Matthias HEIN. “Globally Optimal Training of Generalized Polynomial Neural Networks with Nonlinear Spectral Methods”. In : *Advances in Neural Information Processing Systems*. Sous la dir. de D. LEE et al. T. 29. Curran Associates, Inc., 2016. URL : <https://proceedings.neurips.cc/paper/2016/file/1f4477bad7af3616c1f933a02bfabe4e-Paper.pdf>.
- [61] Xavier GLOROT et Yoshua BENGIO. “Understanding the difficulty of training deep feedforward neural networks”. In : *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Sous la dir. d’Yee Whye TEH et Mike TITTERINGTON. T. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy : PMLR, 2010, p. 249-256. URL : <https://proceedings.mlr.press/v9/glorot10a.html>.
- [62] Noah GOLOWICH, Alexander RAKHLIN et Ohad SHAMIR. “Size-Independent Sample Complexity of Neural Networks”. In : *arXiv e-prints*, arXiv:1712.06541 (déc. 2017), arXiv:1712.06541. DOI : [10 . 48550 / arXiv . 1712 . 06541](https://doi.org/10.48550/arXiv.1712.06541). arXiv : [1712 . 06541](https://arxiv.org/abs/1712.06541) [cs.LG].
- [63] Alona GOLTS, Daniel FREEDMAN et Michael ELAD. “Deep energy: Unsupervised Training of Deep Neural Networks”. In : *CoRR* cs.LG/1805.12355v2 (2018).
- [64] I. GOODFELLOW et al. “Generative adversarial nets”. In : *NIPS 27* (2014).
- [65] Ian J. GOODFELLOW, Oriol VINYALS et Andrew M. SAXE. “Qualitatively characterizing neural network optimization problems”. In : *arXiv e-prints*, arXiv:1412.6544 (déc. 2014), arXiv:1412.6544. DOI : [10 . 48550 / arXiv . 1412 . 6544](https://doi.org/10.48550/arXiv.1412.6544). arXiv : [1412 . 6544](https://arxiv.org/abs/1412.6544) [cs.NE].
- [66] Jianping GOU et al. “Knowledge Distillation: A Survey”. In : *arXiv e-prints*, arXiv:2006.05525 (juin 2020), arXiv:2006.05525. DOI : [10 . 48550 / arXiv . 2006 . 05525](https://doi.org/10.48550/arXiv.2006.05525). arXiv : [2006 . 05525](https://arxiv.org/abs/2006.05525) [cs.LG].
- [67] Henry GOUK, Timothy M. HOSPEDALES et Massimiliano PONTIL. “Distance-Based Regularisation of Deep Networks for Fine-Tuning”. In : *arXiv e-prints*, arXiv:2002.08253 (fév. 2020), arXiv:2002.08253. arXiv : [2002 . 08253](https://arxiv.org/abs/2002.08253) [stat.ML].
- [68] A. GRAVES, A. MOHAMED et G. HINTON. “Speech recognition with deep recurrent neural networks”. In : *IEEE ICASSP*. Ieee. 2013, p. 6645-6649.
- [69] Alex GRAVES, Navdeep JAITLEY et Abdel-rahman MOHAMED. “Hybrid speech recognition with deep bidirectional LSTM”. In : *2013 IEEE workshop on automatic speech recognition and understanding*. IEEE. 2013, p. 273-278.

- [70] Alex GRAVES et Jürgen SCHMIDHUBER. “Framewise phoneme classification with bidirectional LSTM and other neural network architectures”. In : *Neural networks* 18.5-6 (2005), p. 602-610.
- [71] Alex GRAVES et Jürgen SCHMIDHUBER. “Offline handwriting recognition with multidimensional recurrent neural networks”. In : *Advances in neural information processing systems* 21 (2008).
- [72] Harsha GURULINGAPPA et al. “Development of a benchmark corpus to support the automatic extraction of drug-related adverse effects from medical case reports”. In : *Journal of Biomedical Informatics* 45.5 (2012). Text Mining and Natural Language Processing in Pharmacogenomics, p. 885 -892. ISSN : 1532-0464. DOI : <https://doi.org/10.1016/j.jbi.2012.04.008>. URL : <http://www.sciencedirect.com/science/article/pii/S1532046412000615>.
- [73] Benjamin D. HAEFFELE et Rene VIDAL. “Global Optimality in Tensor Factorization, Deep Learning, and Beyond”. In : *arXiv e-prints*, arXiv:1506.07540 (juin 2015), arXiv:1506.07540. DOI : [10.48550/arXiv.1506.07540](https://doi.org/10.48550/arXiv.1506.07540). arXiv : [1506.07540](https://arxiv.org/abs/1506.07540) [cs.NA].
- [74] Song HAN et al. “Learning both Weights and Connections for Efficient Neural Networks”. In : *arXiv e-prints*, arXiv:1506.02626 (juin 2015), arXiv:1506.02626. DOI : [10.48550/arXiv.1506.02626](https://doi.org/10.48550/arXiv.1506.02626). arXiv : [1506.02626](https://arxiv.org/abs/1506.02626) [cs.NE].
- [75] Jiaao HE et al. “FastMoE: A Fast Mixture-of-Expert Training System”. In : *arXiv e-prints*, arXiv:2103.13262 (mars 2021), arXiv:2103.13262. DOI : [10.48550/arXiv.2103.13262](https://doi.org/10.48550/arXiv.2103.13262). arXiv : [2103.13262](https://arxiv.org/abs/2103.13262) [cs.LG].
- [76] Kaiming HE et al. “Deep residual learning for image recognition”. In : *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, p. 770-778.
- [77] Pengcheng HE et al. “Deberta: Decoding-enhanced bert with disentangled attention”. In : *arXiv preprint arXiv:2006.03654* (2020).
- [78] P. HEALY et N. S. NIKOLOV. “How to Layer a Directed Acyclic Graph”. In : *Graph Drawing*. 2001.
- [79] Dan HENDRYCKS et al. “Aligning AI With Shared Human Values”. In : *arXiv e-prints*, arXiv:2008.02275 (août 2020), arXiv:2008.02275. arXiv : [2008.02275](https://arxiv.org/abs/2008.02275) [cs.CY].
- [80] Hansika HEWAMALAGE, Christoph BERGMEIR et Kasun BANDARA. “Recurrent neural networks for time series forecasting: Current status and future directions”. In : *International Journal of Forecasting* 37.1 (2021), p. 388-427.
- [81] Geoffrey HINTON, Oriol VINYALS et Jeff DEAN. “Distilling the Knowledge in a Neural Network”. In : *arXiv e-prints*, arXiv:1503.02531 (mars 2015), arXiv:1503.02531. DOI : [10.48550/arXiv.1503.02531](https://doi.org/10.48550/arXiv.1503.02531). arXiv : [1503.02531](https://arxiv.org/abs/1503.02531) [stat.ML].

- [82] Sepp HOCHREITER et Jürgen SCHMIDHUBER. “Flat Minima”. In : *Neural Computation* 9.1 (jan. 1997), p. 1-42. ISSN : 0899-7667. DOI : [10.1162/neco.1997.9.1.1](https://doi.org/10.1162/neco.1997.9.1.1). eprint : <https://direct.mit.edu/neco/article-pdf/9/1/1/813385/neco.1997.9.1.1.pdf>. URL : <https://doi.org/10.1162/neco.1997.9.1.1>.
- [83] Sepp HOCHREITER et Jürgen SCHMIDHUBER. “Long short-term memory”. In : *Neural computation* 9.8 (1997), p. 1735-1780.
- [84] Torsten HOEFLER et al. “Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks”. In : *Journal of Machine Learning Research* 22.241 (2021), p. 1-124. URL : <http://jmlr.org/papers/v22/21-0366.html>.
- [85] Elad HOFFER et al. “Norm matters: efficient and accurate normalization schemes in deep networks”. In : *arXiv e-prints*, arXiv:1803.01814 (mars 2018), arXiv:1803.01814. DOI : [10.48550/arXiv.1803.01814](https://doi.org/10.48550/arXiv.1803.01814). arXiv : [1803.01814 \[stat.ML\]](https://arxiv.org/abs/1803.01814).
- [86] Kurt HORNİK. “Approximation capabilities of multilayer feedforward networks”. In : *Neural networks* 4.2 (1991), p. 251-257.
- [87] Lei HUANG et al. “Projection Based Weight Normalization for Deep Neural Networks”. In : *arXiv e-prints*, arXiv:1710.02338 (oct. 2017), arXiv:1710.02338. DOI : [10.48550/arXiv.1710.02338](https://doi.org/10.48550/arXiv.1710.02338). arXiv : [1710.02338 \[cs.LG\]](https://arxiv.org/abs/1710.02338).
- [88] Shaohan HUANG et al. “Language Is Not All You Need: Aligning Perception with Language Models”. In : *arXiv e-prints*, arXiv:2302.14045 (fév. 2023), arXiv:2302.14045. DOI : [10.48550/arXiv.2302.14045](https://doi.org/10.48550/arXiv.2302.14045). arXiv : [2302.14045 \[cs.CL\]](https://arxiv.org/abs/2302.14045).
- [89] Zhiheng HUANG, Wei XU et Kai YU. “Bidirectional LSTM-CRF models for sequence tagging”. In : *arXiv preprint arXiv:1508.01991* (2015).
- [90] C.-Y. HUNG et al. “Compacting, Picking and Growing for Unforgetting Continual Learning”. In : *NeurIPS*. T. 32. 2019. URL : <https://proceedings.neurips.cc/paper/2019/file/3b220b436e5f3d917a1e649a0dc0281c-Paper.pdf>.
- [91] Frank HUTTER, Lars KOTTHOFF et Joaquin VANSCHOREN. *Automated Machine Learning: Methods, Systems, Challenges*. 1st. Springer Publishing Company, Incorporated, 2019. ISBN : 3030053172.
- [92] Majid JANZAMIN, Hanie SEDGHI et Anima ANANDKUMAR. “Beating the Perils of Non-Convexity: Guaranteed Training of Neural Networks using Tensor Methods”. In : *arXiv e-prints*, arXiv:1506.08473 (juin 2015), arXiv:1506.08473. DOI : [10.48550/arXiv.1506.08473](https://doi.org/10.48550/arXiv.1506.08473). arXiv : [1506.08473 \[cs.LG\]](https://arxiv.org/abs/1506.08473).

- [93] Haoming JIANG et al. “SMART: Robust and Efficient Fine-Tuning for Pre-trained Natural Language Models through Principled Regularized Optimization”. In : *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online : Association for Computational Linguistics, juill. 2020, p. 2177-2190. DOI : [10.18653/v1/2020.acl-main.197](https://doi.org/10.18653/v1/2020.acl-main.197). URL : <https://aclanthology.org/2020.acl-main.197>.
- [94] Y.\* JIANG et al. “Fantastic Generalization Measures and Where to Find Them”. In : *ICLR*. 2020. URL : <https://openreview.net/forum?id=SJgIPJBFvH>.
- [95] Yichen JIANG et al. “HoVer: A Dataset for Many-Hop Fact Extraction And Claim Verification”. In : *arXiv e-prints*, arXiv:2011.03088 (nov. 2020), arXiv:2011.03088. arXiv : [2011.03088 \[cs.CL\]](https://arxiv.org/abs/2011.03088).
- [96] Mandar JOSHI et al. “SpanBERT: Improving Pre-training by Representing and Predicting Spans”. In : *Transactions of the Association for Computational Linguistics* 8 (2020), p. 64-77. DOI : [10.1162/tacl\\_a\\_00300](https://doi.org/10.1162/tacl_a_00300). URL : <https://aclanthology.org/2020.tacl-1.5>.
- [97] Rafal JOZEFOWICZ et al. “Exploring the limits of language modeling”. In : *arXiv preprint arXiv:1602.02410* (2016).
- [98] Rabeeh KARIMI MAHABADI, Yonatan BELINKOV et James HENDERSON. “Variational Information Bottleneck for Effective Low-Resource Fine-Tuning”. In : *arXiv e-prints*, arXiv:2106.05469 (juin 2021), arXiv:2106.05469. arXiv : [2106.05469 \[cs.CL\]](https://arxiv.org/abs/2106.05469).
- [99] Gintare KAROLINA DZIUGAITE et Daniel M. ROY. “Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data”. In : *arXiv e-prints*, arXiv:1703.11008 (mars 2017), arXiv:1703.11008. DOI : [10.48550/arXiv.1703.11008](https://doi.org/10.48550/arXiv.1703.11008). arXiv : [1703.11008 \[cs.LG\]](https://arxiv.org/abs/1703.11008).
- [100] T. KARRAS et al. “Progressive Growing of GANs for Improved Quality, Stability, and Variation”. In : *ICLR*. 2018. URL : <https://openreview.net/forum?id=Hk99zCeAb>.
- [101] K. KAWAGUCHI. “Deep Learning without Poor Local Minima”. In : *NeurIPS*. 2016, p. 586-594.
- [102] K. KAWAGUCHI et L. KAEHLING. “Elimination of all bad local minima in deep learning”. In : *AISTATS*. PMLR. 2020, p. 853-863.
- [103] K. KAWAGUCHI, L. P. KAEHLING et Y. BENGIO. “Generalization in deep learning”. In : *arXiv preprint arXiv:1710.05468* (2017).
- [104] Kenji KAWAGUCHI et Jiaoyang HUANG. “Gradient Descent Finds Global Minima for Generalizable Deep Neural Networks of Practical Sizes”. In : *arXiv e-prints*, arXiv:1908.02419 (août 2019), arXiv:1908.02419. DOI : [10.48550/arXiv.1908.02419](https://doi.org/10.48550/arXiv.1908.02419). arXiv : [1908.02419 \[stat.ML\]](https://arxiv.org/abs/1908.02419).
- [105] N. S. KESKAR et al. “On large-batch training for deep learning: Generalization gap and sharp minima”. In : *arXiv preprint arXiv:1609.04836* (2016).

- [106] Salman KHAN et al. “Transformers in vision: A survey”. In : *ACM computing surveys (CSUR)* 54.10s (2022), p. 1-41.
- [107] Daniel KHASHABI et al. “Looking Beyond the Surface:A Challenge Set for Reading Comprehension over Multiple Sentences”. In : *NAACL*. 2018.
- [108] Patrick KIDGER et Terry LYONS. “Universal approximation with deep narrow networks”. In : *Conference on learning theory*. PMLR. 2020, p. 2306-2327.
- [109] Ozsel KILINC et Ismail UYSAL. “Learning Latent Representations in Neural Networks for Clustering through Pseudo Supervision and Graph-based Activity Regularization”. In : *Proc. ICLR*. 2018.
- [110] Yoon KIM et al. “Structured attention networks”. In : *arXiv preprint arXiv:1702.00887* (2017).
- [111] Diederik P KINGMA et Max WELLING. “Auto-encoding variational bayes”. In : *arXiv preprint arXiv:1312.6114* (2013).
- [112] Alex KRIZHEVSKY, Ilya SUTSKEVER et Geoffrey E HINTON. “Imagenet classification with deep convolutional neural networks”. In : *Communications of the ACM* 60.6 (2017), p. 84-90.
- [113] Oleksii KUCHARIEV et Boris GINSBURG. “Factorization tricks for LSTM networks”. In : *arXiv preprint arXiv:1703.10722* (2017).
- [114] Zhenzhong LAN et al. “ALBERT: A Lite BERT for Self-supervised Learning of Language Representations”. In : *arXiv e-prints*, arXiv:1909.11942 (sept. 2019), arXiv:1909.11942. arXiv : [1909.11942 \[cs.CL\]](https://arxiv.org/abs/1909.11942).
- [115] Yann LE CUN et Françoise FOGELMAN-SOULIÉ. “Modèles connexionnistes de l’apprentissage”. In : *Intellectica* 2.1 (1987), p. 114-143.
- [116] Y. LECUN et al. “Gradient-based learning applied to document recognition”. In : *Proceedings of the IEEE* 86.11 (1998), p. 2278-2324. DOI : [10.1109/5.726791](https://doi.org/10.1109/5.726791).
- [117] Yann LECUN et al. “Backpropagation applied to handwritten zip code recognition”. In : *Neural computation* 1.4 (1989), p. 541-551.
- [118] Yann LECUN et al. “Gradient-based learning applied to document recognition”. In : *Proceedings of the IEEE* 86.11 (1998), p. 2278-2324.
- [119] Antoine LEDENT et al. “Norm-based generalisation bounds for multi-class convolutional neural networks”. In : *arXiv e-prints*, arXiv:1905.12430 (mai 2019), arXiv:1905.12430. DOI : [10.48550/arXiv.1905.12430](https://doi.org/10.48550/arXiv.1905.12430). arXiv : [1905.12430 \[cs.LG\]](https://arxiv.org/abs/1905.12430).
- [120] Cheolhyoung LEE, Kyunghyun CHO et Wanmo KANG. “Mixout: Effective Regularization to Finetune Large-scale Pretrained Language Models”. In : *International Conference on Learning Representations*. 2020. URL : <https://openreview.net/forum?id=HkgaETNtDB>.
- [121] M. LESHNO et al. “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function”. In : *Neural networks* 6.6 (1993), p. 861-867.

- [122] L. LI et A. TALWALKAR. “Random search and reproducibility for neural architecture search”. In : *UAI*. PMLR. 2020, p. 367-377.
- [123] X. LI et al. “Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting”. In : *ICML*. PMLR. 2019, p. 3925-3934.
- [124] Tengyuan LIANG et al. “Fisher-Rao Metric, Geometry, and Complexity of Neural Networks”. In : *arXiv e-prints*, arXiv:1711.01530 (nov. 2017), arXiv:1711.01530. DOI : [10.48550/arXiv.1711.01530](https://doi.org/10.48550/arXiv.1711.01530). arXiv : [1711.01530](https://arxiv.org/abs/1711.01530) [cs.LG].
- [125] Zhouhan LIN et al. “A structured self-attentive sentence embedding”. In : *arXiv preprint arXiv:1703.03130* (2017).
- [126] Zhouhan LIN et al. “A Structured Self-Attentive Sentence Embedding”. In : *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL : [https://openreview.net/forum?id=BJC\\\_jUqxe](https://openreview.net/forum?id=BJC\_jUqxe).
- [127] Hanxiao LIU, Karen SIMONYAN et Yiming YANG. “DARTS: Differentiable Architecture Search”. In : *International Conference on Learning Representations*. 2019. URL : <https://openreview.net/forum?id=S1eYHoC5FX>.
- [128] Hanxiao LIU et al. “Hierarchical Representations for Efficient Architecture Search”. In : *International Conference on Learning Representations*. 2018. URL : <https://openreview.net/forum?id=BJQRKzba->.
- [129] Y. LIU et al. “Roberta: A robustly optimized bert pretraining approach”. In : *arXiv preprint arXiv:1907.11692* (2019).
- [130] Yinhan LIU et al. “RoBERTa: A Robustly Optimized BERT Pretraining Approach”. In : *arXiv e-prints*, arXiv:1907.11692 (juill. 2019), arXiv:1907.11692. arXiv : [1907.11692](https://arxiv.org/abs/1907.11692) [cs.CL].
- [131] Philip M. LONG et Hanie SEDGHI. “Generalization bounds for deep convolutional neural networks”. In : *arXiv e-prints*, arXiv:1905.12600 (mai 2019), arXiv:1905.12600. DOI : [10.48550/arXiv.1905.12600](https://doi.org/10.48550/arXiv.1905.12600). arXiv : [1905.12600](https://arxiv.org/abs/1905.12600) [cs.LG].
- [132] Minh-Thang LUONG, Hieu PHAM et Christopher D MANNING. “Effective approaches to attention-based neural machine translation”. In : *arXiv preprint arXiv:1508.04025* (2015).
- [133] David A. McALLESTER. “PAC-Bayesian Model Averaging”. In : *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*. COLT '99. Santa Cruz, California, USA : Association for Computing Machinery, 1999, p. 164-170. ISBN : 1581131674. DOI : [10.1145/307400.307435](https://doi.org/10.1145/307400.307435). URL : <https://doi.org/10.1145/307400.307435>.
- [134] M. McCLOSKEY et N. J. COHEN. “Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem”. English (US). In : *Psychology of Learning and Motivation - Research and Theory* 24.C (jan. 1989), p. 109-165. ISSN : 0079-7421. DOI : [10.1016/S0079-7421\(08\)60536-8](https://doi.org/10.1016/S0079-7421(08)60536-8).

- [135] Warren S McCULLOCH et Walter PITTS. “A logical calculus of the ideas immanent in nervous activity”. In : *The bulletin of mathematical biophysics* 5.4 (1943), p. 115-133.
- [136] Luke METZ et al. “Meta-Learning Update Rules for Unsupervised Representation Learning”. In : *CoRR* cs.LG/1804.00222v3 (2018).
- [137] Tomas MIKOLOV et al. “Distributed representations of words and phrases and their compositionality”. In : *Proc. NIPS*. 2013, p. 3111-3119.
- [138] Pavlo MOLCHANOV et al. “Pruning Convolutional Neural Networks for Resource Efficient Inference”. In : *arXiv e-prints*, arXiv:1611.06440 (nov. 2016), arXiv:1611.06440. DOI : [10.48550/arXiv.1611.06440](https://doi.org/10.48550/arXiv.1611.06440). arXiv : [1611.06440](https://arxiv.org/abs/1611.06440) [cs.LG].
- [139] Vaishnavh NAGARAJAN et J. ZICO KOLTER. “Deterministic PAC-Bayesian generalization bounds for deep networks via generalizing noise-resilience”. In : *arXiv e-prints*, arXiv:1905.13344 (mai 2019), arXiv:1905.13344. DOI : [10.48550/arXiv.1905.13344](https://doi.org/10.48550/arXiv.1905.13344). arXiv : [1905.13344](https://arxiv.org/abs/1905.13344) [cs.LG].
- [140] R. NEGRINHO et al. “Towards modular and programmable architecture search”. In : *NeurIPS* 32 (2019).
- [141] B. NEYSHABUR, R. TOMIOKA et N. SREBRO. “Norm-based capacity control in neural networks”. In : *COLT*. PMLR. 2015, p. 1376-1401.
- [142] Behnam NEYSHABUR, Srinadh BHOJANAPALLI et Nathan SREBRO. “A PAC-Bayesian Approach to Spectrally-Normalized Margin Bounds for Neural Networks”. In : *arXiv e-prints*, arXiv:1707.09564 (juill. 2017), arXiv:1707.09564. DOI : [10.48550/arXiv.1707.09564](https://doi.org/10.48550/arXiv.1707.09564). arXiv : [1707.09564](https://arxiv.org/abs/1707.09564) [cs.LG].
- [143] Behnam NEYSHABUR et al. “Exploring Generalization in Deep Learning”. In : *Advances in Neural Information Processing Systems*. Sous la dir. d'I. GUYON et al. T. 30. Curran Associates, Inc., 2017. URL : <https://proceedings.neurips.cc/paper/2017/file/10ce03a1ed01077e3e289f3e53c72813-Paper.pdf>.
- [144] Roman NOVAK et al. “Sensitivity and Generalization in Neural Networks: an Empirical Study”. In : *arXiv e-prints*, arXiv:1802.08760 (fév. 2018), arXiv:1802.08760. DOI : [10.48550/arXiv.1802.08760](https://doi.org/10.48550/arXiv.1802.08760). arXiv : [1802.08760](https://arxiv.org/abs/1802.08760) [stat.ML].
- [145] Albert B NOVIKOFF. *On convergence proofs for perceptrons*. Rapp. tech. STANFORD RESEARCH INST MENLO PARK CA, 1963.
- [146] Bo PANG et Lillian LEE. “Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales”. In : *Proceedings of the ACL*. 2005.
- [147] Ankur P PARIKH et al. “A decomposable attention model for natural language inference”. In : *arXiv preprint arXiv:1606.01933* (2016).
- [148] G. I. PARISI et al. “Continual lifelong learning with neural networks: A review”. In : *Neural Networks* 113 (2019), p. 54-71. ISSN : 0893-6080. DOI : <https://doi.org/10.1016/j.neunet.2019.01.012>. URL : <https://www.sciencedirect.com/science/article/pii/S0893608019300231>.

- [149] Romain PAULUS, Caiming XIONG et Richard SOCHER. “A deep reinforced model for abstractive summarization”. In : *arXiv preprint arXiv:1705.04304* (2017).
- [150] Jeffrey PENNINGTON, Richard SOCHER et Christopher D. MANNING. “GloVe: Global Vectors for Word Representation”. In : *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, p. 1532-1543. URL : <http://www.aclweb.org/anthology/D14-1162>.
- [151] Henning PETZKA et al. “A Reparameterization-Invariant Flatness Measure for Deep Neural Networks”. In : *arXiv e-prints*, arXiv:1912.00058 (nov. 2019), arXiv:1912.00058. DOI : [10.48550/arXiv.1912.00058](https://doi.org/10.48550/arXiv.1912.00058). arXiv : [1912.00058](https://arxiv.org/abs/1912.00058) [cs.LG].
- [152] Jason PHANG, Thibault FÉVRY et Samuel R. BOWMAN. “Sentence Encoders on STILTs: Supplementary Training on Intermediate Labeled-data Tasks”. In : *arXiv e-prints*, arXiv:1811.01088 (nov. 2018), arXiv:1811.01088. arXiv : [1811.01088](https://arxiv.org/abs/1811.01088) [cs.CL].
- [153] Adam POLIAK et al. “Collecting Diverse Natural Language Inference Problems for Sentence Representation Evaluation”. In : *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium : Association for Computational Linguistics, 2018, p. 67-81. DOI : [10.18653/v1/D18-1007](https://doi.org/10.18653/v1/D18-1007). URL : <https://aclanthology.org/D18-1007>.
- [154] Zhiqian QI et al. “Adaboost-llp: a boosting method for learning with label proportions”. In : *IEEE transactions on neural networks and learning systems* 29.8 (2017), p. 3548-3559.
- [155] Xinjian QIANG, Guojian CHENG et Zheng WANG. “An overview of some classical Growing Neural Networks and new developments”. In : *ICETC*. T. 3. 2010, p. V3-351-V3-355. DOI : [10.1109/ICETC.2010.5529527](https://doi.org/10.1109/ICETC.2010.5529527).
- [156] Alec RADFORD, Luke METZ et Soumith CHINTALA. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”. In : *CoRR* cs.LG/1511.06434v2 (2015). URL : <http://arxiv.org/abs/1511.06434>.
- [157] Colin RAFFEL et al. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. In : *arXiv e-prints*, arXiv:1910.10683 (oct. 2019), arXiv:1910.10683. arXiv : [1910.10683](https://arxiv.org/abs/1910.10683) [cs.LG].
- [158] A. RANGAMANI et al. “A scale invariant flatness measure for deep network minima”. In : *arXiv preprint arXiv:1902.02434* (2019).
- [159] Esteban REAL et al. “Regularized Evolution for Image Classifier Architecture Search”. In : *arXiv e-prints*, arXiv:1802.01548 (fév. 2018), arXiv:1802.01548. DOI : [10.48550/arXiv.1802.01548](https://doi.org/10.48550/arXiv.1802.01548). arXiv : [1802.01548](https://arxiv.org/abs/1802.01548) [cs.NE].
- [160] Pengzhen REN et al. “A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions”. In : *ACM Comput. Surv.* 54.4 (2021). ISSN : 0360-0300. DOI : [10.1145/3447582](https://doi.org/10.1145/3447582). URL : <https://doi.org/10.1145/3447582>.

- [161] Rebecca ROELOFS et al. “A Meta-Analysis of Overfitting in Machine Learning”. In : *Advances in Neural Information Processing Systems*. Sous la dir. de H. WALLACH et al. T. 32. Curran Associates, Inc., 2019. URL : <https://proceedings.neurips.cc/paper/2019/file/ee39e503b6bedf0c98c388b7e8589aca-Paper.pdf>.
- [162] Frank ROSENBLATT. “The perceptron: a probabilistic model for information storage and organization in the brain.” In : *Psychological review* 65.6 (1958), p. 386.
- [163] Lukas RUFF et al. “Deep one-class classification”. In : *International Conference on Machine Learning*. 2018, p. 4390-4399.
- [164] Lukas RUFF et al. “Self-Attentive, Multi-Context One-Class Classification for Unsupervised Anomaly Detection on Text”. In : *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019, p. 4061-4071.
- [165] David E RUMELHART, Geoffrey E HINTON et Ronald J WILLIAMS. “Learning representations by back-propagating errors”. In : *nature* 323.6088 (1986), p. 533-536.
- [166] Itay SAFRAN et Ohad SHAMIR. “On the Quality of the Initial Basin in Overspecified Neural Networks”. In : *Proceedings of The 33rd International Conference on Machine Learning*. Sous la dir. de Maria Florina BALCAN et Kilian Q. WEINBERGER. T. 48. Proceedings of Machine Learning Research. New York, New York, USA : PMLR, 2016, p. 774-782. URL : <https://proceedings.mlr.press/v48/safran16.html>.
- [167] Teven Le SCAO et al. “Bloom: A 176b-parameter open-access multilingual language model”. In : *arXiv preprint arXiv:2211.05100* (2022).
- [168] Bernhard SCHÖLKOPF et al. “Estimating the support of a high-dimensional distribution”. In : *Neural computation* 13.7 (2001), p. 1443-1471.
- [169] Noam SHAZEER et al. “Outrageously large neural networks: The sparsely-gated mixture-of-experts layer”. In : *arXiv preprint arXiv:1701.06538* (2017).
- [170] Noam SHAZEER et al. “Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer”. In : *arXiv e-prints*, arXiv:1701.06538 (jan. 2017), arXiv:1701.06538. DOI : [10.48550/arXiv.1701.06538](https://doi.org/10.48550/arXiv.1701.06538). arXiv : [1701.06538](https://arxiv.org/abs/1701.06538) [cs.LG].
- [171] Damien SILEO et Marie-Francine MOENS. “Analysis and Prediction of NLP Models Via Task Embeddings”. In : *ArXiv abs/2112.05647* (2021).
- [172] Damien SILEO et al. “A Pragmatics-Centered Evaluation Framework for Natural Language Understanding”. In : *arXiv e-prints*, arXiv:1907.08672 (juill. 2019), arXiv:1907.08672. arXiv : [1907.08672](https://arxiv.org/abs/1907.08672) [cs.CL].
- [173] Jiří SIMA. “Training a Single Sigmoidal Neuron is Hard”. In : *Neural Comput.* 14.11 (2002), p. 2709-2728. ISSN : 0899-7667. DOI : [10.1162/089976602760408035](https://doi.org/10.1162/089976602760408035). URL : <https://doi.org/10.1162/089976602760408035>.

- [174] Karen SIMONYAN et Andrew ZISSERMAN. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In : *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Sous la dir. d’Yoshua BENGIO et Yann LECUN. 2015. URL : <http://arxiv.org/abs/1409.1556>.
- [175] S. SINHA, A. GARG et H. LAROCHELLE. “Curriculum By Smoothing”. In : *NeurIPS*. Sous la dir. de H. LAROCHELLE et al. T. 33. Curran Associates, Inc., 2020, p. 21653-21664. URL : <https://proceedings.neurips.cc/paper/2020/file/f6a673f09493afcd8b129a0bcf1cd5bc-Paper.pdf>.
- [176] Mahdi SOLTANOLKOTABI. “Learning ReLUs via Gradient Descent”. In : *arXiv e-prints*, arXiv:1705.04591 (mai 2017), arXiv:1705.04591. DOI : [10.48550/arXiv.1705.04591](https://doi.org/10.48550/arXiv.1705.04591). arXiv : [1705.04591](https://arxiv.org/abs/1705.04591) [cs.LG].
- [177] Ryan SPRING et Anshumali SHRIVASTAVA. “Scalable and Sustainable Deep Learning via Randomized Hashing”. In : *arXiv e-prints*, arXiv:1602.08194 (fév. 2016), arXiv:1602.08194. DOI : [10.48550/arXiv.1602.08194](https://doi.org/10.48550/arXiv.1602.08194). arXiv : [1602.08194](https://arxiv.org/abs/1602.08194) [stat.ML].
- [178] Nitish SRIVASTAVA et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In : *Journal of Machine Learning Research* 15.56 (2014), p. 1929-1958. URL : <http://jmlr.org/papers/v15/srivastava14a.html>.
- [179] Kenneth O. STANLEY et al. “Designing neural networks through neuroevolution”. In : *Nature Machine Intelligence* 1.1 (2019), p. 24-35. ISSN : 2522-5839. DOI : [10.1038/s42256-018-0006-z](https://doi.org/10.1038/s42256-018-0006-z). URL : <https://doi.org/10.1038/s42256-018-0006-z>.
- [180] R. SUN et al. “The Global Landscape of Neural Networks: An Overview”. In : *IEEE Signal Processing Magazine* 37.5 (2020), p. 95-108. DOI : [10.1109/MSP.2020.3004124](https://doi.org/10.1109/MSP.2020.3004124).
- [181] I. SUTSKEVER, O. VINYALS et Q. V LE. “Sequence to sequence learning with neural networks”. In : *NIPS*. 2014, p. 3104-3112.
- [182] Richard S. SUTTON et Andrew G. BARTO. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL : <http://incompleteideas.net/book/the-book-2nd.html>.
- [183] Mingxing TAN et Quoc LE. “Efficientnetv2: Smaller models and faster training”. In : *International conference on machine learning*. PMLR. 2021, p. 10096-10106.
- [184] David MJ TAX et Robert PW DUIN. “Support vector data description”. In : *Machine learning* 54.1 (2004), p. 45-66.
- [185] Hugo TOUVRON et al. “LLaMA: Open and Efficient Foundation Language Models”. In : *arXiv e-prints*, arXiv:2302.13971 (fév. 2023), arXiv:2302.13971. DOI : [10.48550/arXiv.2302.13971](https://doi.org/10.48550/arXiv.2302.13971). arXiv : [2302.13971](https://arxiv.org/abs/2302.13971) [cs.CL].

- [186] Gregor URBAN et al. “Do Deep Convolutional Nets Really Need to be Deep and Convolutional?” In : *arXiv e-prints*, arXiv:1603.05691 (mars 2016), arXiv:1603.05691. DOI : [10.48550/arXiv.1603.05691](https://doi.org/10.48550/arXiv.1603.05691). arXiv : [1603.05691](https://arxiv.org/abs/1603.05691) [stat.ML].
- [187] Ruth URNER, Shai BEN-DAVID et Shai SHALEV-SHWARTZ. “Access to Unlabeled Data Can Speed up Prediction Time”. In : *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML’11. Bellevue, Washington, USA : Omnipress, 2011, p. 641-648. ISBN : 9781450306195.
- [188] Chris VAN PELT et Alex SOROKIN. “Designing a Scalable Crowdsourcing Platform”. In : *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’12. Scottsdale, Arizona, USA : Association for Computing Machinery, 2012, p. 765-766. ISBN : 9781450312479. DOI : [10.1145/2213836.2213951](https://doi.org/10.1145/2213836.2213951). URL : <https://doi.org/10.1145/2213836.2213951>.
- [189] Joaquin VANSCHOREN. “Meta-Learning: A Survey”. In : *arXiv e-prints*, arXiv:1810.03548 (oct. 2018), arXiv:1810.03548. DOI : [10.48550/arXiv.1810.03548](https://doi.org/10.48550/arXiv.1810.03548). arXiv : [1810.03548](https://arxiv.org/abs/1810.03548) [cs.LG].
- [190] V. N. VAPNIK et A. Ya. CHERVONENKIS. “On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities”. In : *Measures of Complexity: Festschrift for Alexey Chervonenkis*. Sous la dir. de Vladimir VOVK, Harris PAPADOPOULOS et Alexander GAMMERMAN. Cham : Springer International Publishing, 2015, p. 11-30. ISBN : 978-3-319-21852-6. DOI : [10.1007/978-3-319-21852-6\\_3](https://doi.org/10.1007/978-3-319-21852-6_3). URL : [https://doi.org/10.1007/978-3-319-21852-6\\_3](https://doi.org/10.1007/978-3-319-21852-6_3).
- [191] Ashish VASWANI et al. “Attention Is All You Need”. In : *arXiv e-prints*, arXiv:1706.03762 (juin 2017), arXiv:1706.03762. arXiv : [1706.03762](https://arxiv.org/abs/1706.03762) [cs.CL].
- [192] Nguyen VIET DANG. “Complex powers of analytic functions and meromorphic renormalization in QFT”. In : *arXiv e-prints*, arXiv:1503.00995 (mars 2015), arXiv:1503.00995. DOI : [10.48550/arXiv.1503.00995](https://doi.org/10.48550/arXiv.1503.00995). arXiv : [1503.00995](https://arxiv.org/abs/1503.00995) [math-ph].
- [193] Pascal VINCENT et al. “Extracting and composing robust features with denoising autoencoders”. In : *Proceedings of the 25th international conference on Machine learning*. 2008, p. 1096-1103.
- [194] A. WANG et al. “GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding”. In : *ICLR*. 2019. URL : <https://openreview.net/forum?id=rJ4km2R5t7>.
- [195] Alex WANG et al. “SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems”. In : *arXiv e-prints*, arXiv:1905.00537 (mai 2019), arXiv:1905.00537. arXiv : [1905.00537](https://arxiv.org/abs/1905.00537) [cs.CL].
- [196] D WANG et al. *Energy-Aware Neural Architecture Optimization with Fast Splitting Steepest Descent*. 2020. URL : <https://openreview.net/forum?id=BygSZAVKvr>.

- [197] Hongyi WANG et al. “Federated Learning with Matched Averaging”. In : *International Conference on Learning Representations*.
- [198] Huan WANG et al. “Recent Advances on Neural Network Pruning at Initialization”. In : *arXiv e-prints*, arXiv:2103.06460 (mars 2021), arXiv:2103.06460. DOI : [10.48550/arXiv.2103.06460](https://doi.org/10.48550/arXiv.2103.06460). arXiv : [2103.06460](https://arxiv.org/abs/2103.06460) [cs.LG].
- [199] A. WARSTADT, A. SINGH et S. R BOWMAN. “Neural network acceptability judgments”. In : *Transactions of the ACL 7* (2019), p. 625-641.
- [200] Alex WARSTADT et al. “BLiMP: The Benchmark of Linguistic Minimal Pairs for English”. In : *arXiv e-prints*, arXiv:1912.00582 (déc. 2019), arXiv:1912.00582. arXiv : [1912.00582](https://arxiv.org/abs/1912.00582) [cs.CL].
- [201] Jason WEI et al. “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”. In : *arXiv e-prints*, arXiv:2201.11903 (jan. 2022), arXiv:2201.11903. DOI : [10.48550/arXiv.2201.11903](https://doi.org/10.48550/arXiv.2201.11903). arXiv : [2201.11903](https://arxiv.org/abs/2201.11903) [cs.CL].
- [202] W. WEN et al. “Smoothout: Smoothing out sharp minima to improve generalization in deep learning”. In : *arXiv preprint arXiv:1805.07898* (2018).
- [203] Martin WISTUBA, Ambrish RAWAT et Tejaswini PEDAPATI. “A survey on neural architecture search”. In : *arXiv preprint arXiv:1905.01392* (2019).
- [204] L. WU et al. “Firefly Neural Architecture Descent: a General Approach for Growing Neural Networks”. In : *NeurIPS*. Sous la dir. de H. LAROCHELLE et al. T. 33. Curran Associates, Inc., 2020, p. 22373-22383. URL : <https://proceedings.neurips.cc/paper/2020/file/fdbe012e2e11314b96402b32c0df26b7-Paper.pdf>.
- [205] Yonghui WU et al. “Google’s neural machine translation system: Bridging the gap between human and machine translation”. In : *arXiv preprint arXiv:1609.08144* (2016).
- [206] C. XING et al. “A Walk with SGD”. In : *arXiv e-prints* (2018). eprint : [1802.08770](https://arxiv.org/abs/1802.08770).
- [207] Runxin XU et al. “Raise a Child in Large Language Model: Towards Effective and Generalizable Fine-tuning”. In : *arXiv e-prints*, arXiv:2109.05687 (sept. 2021), arXiv:2109.05687. arXiv : [2109.05687](https://arxiv.org/abs/2109.05687) [cs.CL].
- [208] Zhilin YANG et al. “XLNet: Generalized Autoregressive Pretraining for Language Understanding”. In : *arXiv e-prints*, arXiv:1906.08237 (juin 2019), arXiv:1906.08237. arXiv : [1906.08237](https://arxiv.org/abs/1906.08237) [cs.CL].
- [209] Tec Yan YAP. *Text Anomaly Detection with ARAE-AnoGAN*. 2020. URL : [https://digitalcommons.iwu.edu/cs\\_honproj/22](https://digitalcommons.iwu.edu/cs_honproj/22).
- [210] Omar F. ZAIDAN, Jason EISNER et Christine PIATKO. “Machine Learning with Annotator Rationales to Reduce Annotation Cost”. In : *Proceedings of the NIPS\*2008 Workshop on Cost Sensitive Learning*. 2008.
- [211] Valentina ZANTEDESCHI, Rémi EMONET et Marc SEBBAN. “Beta-risk: a new surrogate risk for learning from weakly labeled data”. In : *Advances in Neural Information Processing Systems*. 2016, p. 4365-4373.

- [212] Richard S ZEMEL et G HINTON. “Autoencoders, minimum description length and helmholtz free energy”. In : *Proceedings of the Neural Information Processing Systems*. Citeseer. 1994.
- [213] Chiyuan ZHANG et al. “Understanding deep learning requires rethinking generalization”. In : *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL : <https://openreview.net/forum?id=Sy8gdB9xx>.
- [214] Shuofeng ZHANG et al. “Why flatness does and does not correlate with generalization for deep neural networks”. In : *arXiv e-prints*, arXiv:2103.06219 (mars 2021), arXiv:2103.06219. arXiv : [2103.06219 \[cs.LG\]](#).
- [215] Yuan ZHANG, Jason BALDRIDGE et Luheng HE. “PAWS: Paraphrase Adversaries from Word Scrambling”. In : *arXiv e-prints*, arXiv:1904.01130 (avr. 2019), arXiv:1904.01130. arXiv : [1904.01130 \[cs.CL\]](#).
- [216] Yongchao ZHOU et al. “Large Language Models Are Human-Level Prompt Engineers”. In : *arXiv e-prints*, arXiv:2211.01910 (nov. 2022), arXiv:2211.01910. DOI : [10.48550/arXiv.2211.01910](#). arXiv : [2211.01910 \[cs.LG\]](#).
- [217] Barret ZOPH et Quoc LE. “Neural Architecture Search with Reinforcement Learning”. In : *International Conference on Learning Representations*. 2017. URL : <https://openreview.net/forum?id=r1Ue8Hcxg>.
- [218] Barret ZOPH et al. “Learning Transferable Architectures for Scalable Image Recognition”. In : *arXiv e-prints*, arXiv:1707.07012 (juill. 2017), arXiv:1707.07012. DOI : [10.48550/arXiv.1707.07012](#). arXiv : [1707.07012 \[cs.CV\]](#).

# **Annexes**



---

## A Évolution des métriques au cours de l’entraînement

Nous présentons dans cette annexe l’évolution des métriques suivantes lors des 15 époques d’entraînement faites sur la plupart des tâches de classification binaire proposées dans MetaEval que nous n’avons pas voulu insérer à la Section 9.1 du Chapitre 9 par souci de lisibilité.

Rappelons que les métriques observées sont

- **Exactitude de validation** est l’exactitude sur le corpus de validation.
- **Emp. Risk** est le risque empirique calculé sur le corpus d’entraînement.
- **Unsup. Risk** est le risque non supervisé proposé évalué sur le corpus d’entraînement.

Il s’agit de noter que deux valeurs sont parfois notées en gras dans la même colonne (pour les colonnes exactitude de validation et Unsup. Risk). C’est dû au fait que nous considérons les 5 premières époques comme des étapes de warm-up pour le réseau et il est possible comme nous l’avons souligné au Chapitre 9 que la meilleure valeur soit trouvée lors de ces étapes initiales d’entraînement. Dans ce cas, nous indiquons également en gras la meilleure valeur en dehors de cette période de warm-up. Ce genre de comportement est typique du FM2 étudié.

Epoque	Exactitude de validation	Perf. sur le corpus d’entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.6723	0.6763	0.6436
Epoque 2	0.7181	0.6005	0.5216
Epoque 3	0.7521	0.5500	0.3600
Epoque 4	0.7640	0.5164	0.3765
Epoque 5	0.7743	0.4941	0.3551
Epoque 6	0.7835	0.4746	0.2467
Epoque 7	0.7923	0.4557	0.3292
Epoque 8	0.7967	0.4401	0.2805
Epoque 9	0.8051	0.4233	0.2126
Epoque 10	0.7977	0.4130	0.2199
Epoque 11	0.8039	0.3995	0.1924
Epoque 12	0.8093	0.3933	0.1986
Epoque 13	0.8085	0.3858	<b>0.1816</b>
Epoque 14	<b>0.8116</b>	0.3845	0.1909
Epoque 15	0.8082	<b>0.3784</b>	0.2328

TABLE 9 – Commonsense,  $p_0 = 0.5432$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.6783	0.6062	0.4792
Epoque 2	0.7101	0.4912	0.3451
Epoque 3	0.7260	0.4414	0.2837
Epoque 4	0.7326	0.4069	0.1964
Epoque 5	0.7330	0.3808	0.1823
Epoque 6	0.7382	0.3631	0.1668
Epoque 7	0.7371	0.3463	0.1655
Epoque 8	0.7371	0.3327	0.1114
Epoque 9	0.7382	0.3224	0.1141
Epoque 10	0.7422	0.3114	0.1319
Epoque 11	0.7422	0.3082	0.1411
Epoque 12	0.7408	0.3054	0.1137
Epoque 13	<b>0.7433</b>	0.3009	<b>0.0745</b>
Epoque 14	0.7426	0.3001	0.1242
Epoque 15	0.7426	<b>0.2902</b>	0.0967

TABLE 10 – Justice,  $p_0 = 0.4571$ 

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.7960	0.2891	0.6184
Epoque 2	0.8111	0.2002	0.6579
Epoque 3	0.8205	0.1593	0.6804
Epoque 4	0.8255	0.1365	0.7090
Epoque 5	0.8304	0.1119	0.7147
Epoque 6	0.8338	0.0979	0.7385
Epoque 7	0.8372	0.0858	<b>0.7271</b>
Epoque 8	0.8384	0.0726	0.7639
Epoque 9	0.8412	0.0693	0.7726
Epoque 10	0.8406	0.0587	0.7736
Epoque 11	0.8402	0.0539	0.7638
Epoque 12	0.8436	0.0509	0.7999
Epoque 13	0.8436	0.0447	0.8147
Epoque 14	<b>0.8444</b>	0.0434	0.8261
Epoque 15	0.8438	<b>0.0433</b>	0.8071

TABLE 11 – Virtue,  $p_0 = 0.9145$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.5440	2.1215	0.1773
Epoque 2	0.5920	1/0398	0.2564
Epoque 3	0.6160	0.7484	0.3512
Epoque 4	0.6560	0.5931	0.4221
Epoque 5	0.6720	0.5656	0.4330
Epoque 6	0.6960	0.5045	0.3914
Epoque 7	0.7280	0.4622	0.3756
Epoque 8	0.7200	0.4548	0.2899
Epoque 9	0.7360	0.4357	0.2940
Epoque 10	0.7280	0.3889	0.2888
Epoque 11	<b>0.7520</b>	0.3857	0.2779
Epoque 12	0.7520	0.3890	0.2515
Epoque 13	0.7520	0.3677	0.2317
Epoque 14	0.7520	0.3732	<b>0.2189</b>
Epoque 15	0.7520	<b>0.3444</b>	0.2231

TABLE 12 – Binary,  $p_0 = 0.5735$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.6447	0.7854	0.6064
Epoque 2	0.6754	0.6107	0.4784
Epoque 3	0.6915	0.5635	0.4127
Epoque 4	0.6988	0.5299	0.3655
Epoque 5	0.7135	0.5078	0.3257
Epoque 6	0.7149	0.4740	0.3118
Epoque 7	0.7164	0.4605	0.2682
Epoque 8	0.7135	0.4492	0.2485
Epoque 9	0.7032	0.4311	0.2447
Epoque 10	0.7076	0.4056	0.2342
Epoque 11	0.7149	0.3940	0.2181
Epoque 12	0.7135	0.3889	0.2333
Epoque 13	<b>0.7178</b>	0.3882	0.2086
Epoque 14	0.7149	0.3724	0.1992
Epoque 15	0.7164	<b>0.3639</b>	<b>0.1792</b>

TABLE 13 – Emobank-Arousal,  $p_0 = 0.5152$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.7362	1.0727	0.2027
Epoque 2	0.7363	0.6710	0.1821
Epoque 3	0.7473	0.6155	0.3698
Epoque 4	0.7363	0.5691	0.3030
Epoque 5	0.7473	0.5318	0.4174
Epoque 6	0.7473	0.5144	0.2675
Epoque 7	0.7802	0.5080	0.2955
Epoque 8	<b>0.7912</b>	0.4649	0.2061
Epoque 9	0.7473	0.4690	0.2084
Epoque 10	0.7123	0.4382	0.2009
Epoque 11	0.7253	0.4083	0.1849
Epoque 12	0.7363	0.4034	0.1853
Epoque 13	0.7363	0.3909	<b>0.1686</b>
Epoque 14	0.7253	0.3780	0.1993
Epoque 15	0.7253	<b>0.3768</b>	0.1845

TABLE 14 – Persuasiveness-Eloquence,  $p_0 = 0.7338$ 

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.3846	0.7192	0.8054
Epoque 2	0.6264	0.6998	0.8154
Epoque 3	0.6044	0.6847	0.8286
Epoque 4	0.6044	0.6667	0.8076
Epoque 5	0.5934	0.6512	0.8086
Epoque 6	0.6044	0.6469	0.7676
Epoque 7	0.6044	0.6289	0.7675
Epoque 8	0.6264	0.6152	0.7436
Epoque 9	0.6484	0.6040	0.7299
Epoque 10	0.6484	0.5989	0.6976
Epoque 11	<b>0.6703</b>	0.6053	0.6764
Epoque 12	0.6703	0.5758	0.6806
Epoque 13	0.6703	0.5761	0.6757
Epoque 14	0.6703	0.5775	<b>0.6543</b>
Epoque 15	0.6703	<b>0.5702</b>	0.6685

TABLE 15 – Persuasiveness-Relevance,  $p_0 = 0.4014$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.5806	0.6705	0.6158
Epoque 2	0.6290	0.6431	0.6285
Epoque 3	0.6290	0.6093	0.6045
Epoque 4	0.6452	0.5898	0.5868
Epoque 5	0.6774	0.5625	0.5822
Epoque 6	0.6935	0.5410	0.5735
Epoque 7	0.7255	0.5292	0.5468
Epoque 8	0.6935	0.5046	0.5312
Epoque 9	0.7097	0.4950	0.5079
Epoque 10	0.7097	0.4879	0.4765
Epoque 11	<b>0.7258</b>	0.4738	0.4735
Epoque 12	0.7258	0.4674	<b>0.4396</b>
Epoque 13	0.6935	0.4689	0.4470
Epoque 14	0.6935	0.4526	0.4462
Epoque 15	0.7097	<b>0.4467</b>	0.4407

TABLE 16 – Persuasiveness-Specificity,  $p_0 = 0.5496$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.8478	0.5125	0.4364
Epoque 2	0.8478	0.4851	0.4416
Epoque 3	<b>0.8696</b>	0.4552	0.4405
Epoque 4	0.8478	0.4435	0.4255
Epoque 5	0.8478	0.4165	0.3931
Epoque 6	0.8261	0.4052	0.4081
Epoque 7	0.8261	0.3911	0.4069
Epoque 8	<b>0.8478</b>	0.3798	<b>0.3665</b>
Epoque 9	0.8478	0.3589	0.3982
Epoque 10	0.8261	0.3810	0.4059
Epoque 11	0.8261	0.3567	0.3965
Epoque 12	0.8478	0.3545	0.3856
Epoque 13	0.8478	0.3362	0.3942
Epoque 14	0.8478	0.3436	0.4017
Epoque 15	0.8478	<b>0.3349</b>	0.3744

TABLE 17 – Persuasiveness-Strength,  $p_0 = 0.6173$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.6617	0.6660	0.6948
Epoque 2	0.6817	0.6246	0.5851
Epoque 3	<b>0.7005</b>	0.5995	0.5357
Epoque 4	0.6955	0.5787	0.4771
Epoque 5	0.6980	0.5644	0.4751
Epoque 6	0.6917	0.5500	0.4376
Epoque 7	0.6892	0.5358	0.4348
Epoque 8	<b>0.6942</b>	0.5238	0.4314
Epoque 9	0.6855	0.5046	0.4336
Epoque 10	0.6767	0.5016	0.3997
Epoque 11	0.6817	0.4947	0.3932
Epoque 12	0.6754	0.4837	0.3927
Epoque 13	0.6792	0.4805	0.4076
Epoque 14	0.6842	0.4750	<b>0.3849</b>
Epoque 15	0.6817	<b>0.4549</b>	0.3859

TABLE 18 – Emobank-Dominance,  $p_0 = 0.6123$ 

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.6581	0.6968	0.3835
Epoque 2	0.6860	0.5837	0.3493
Epoque 3	0.6925	0.5522	0.3397
Epoque 4	0.7011	0.5211	0.2802
Epoque 5	<b>0.7118</b>	0.5109	0.2764
Epoque 6	<b>0.7032</b>	0.5016	0.2873
Epoque 7	0.6989	0.4798	0.2139
Epoque 8	0.6968	0.4631	0.2177
Epoque 9	0.6989	0.4502	0.2280
Epoque 10	0.6989	0.4350	0.2359
Epoque 11	0.6989	0.4313	0.1889
Epoque 12	0.7032	0.4287	<b>0.1849</b>
Epoque 13	0.7032	0.4117	0.2229
Epoque 14	0.7032	0.4086	0.2115
Epoque 15	0.7011	<b>0.4017</b>	0.2319

TABLE 19 – Squinky-Implicature,  $p_0 = 0.5328$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.6503	0.7681	0.6741
Epoque 2	0.6844	0.5880	0.4162
Epoque 3	0.7143	0.5379	0.3950
Epoque 4	0.7249	0.5045	0.2586
Epoque 5	0.7292	0.4881	0.2367
Epoque 6	<b>0.7335</b>	0.4683	0.1678
Epoque 7	0.7292	0.4461	0.1889
Epoque 8	0.7249	0.4355	0.1790
Epoque 9	0.7292	0.4230	0.1246
Epoque 10	0.7249	0.4062	0.1121
Epoque 11	0.7228	0.3870	0.0981
Epoque 12	0.7313	0.3934	<b>0.0831</b>
Epoque 13	0.7313	0.3780	0.0845
Epoque 14	0.7292	0.3725	0.1029
Epoque 15	0.7313	<b>0.3717</b>	0.1086

TABLE 20 – Sarcasm,  $p_0 = 0.5024$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.9492	0.4028	0.1139
Epoque 2	0.9603	0.1649	0.1854
Epoque 3	0.9691	0.1290	0.2035
Epoque 4	0.9735	0.1156	0.2381
Epoque 5	0.9757	0.1022	0.1918
Epoque 6	0.9691	0.0962	0.2422
Epoque 7	0.9801	0.0935	0.2586
Epoque 8	0.9801	0.0757	<b>0.0273</b>
Epoque 9	0.9779	0.0749	0.2014
Epoque 10	0.9801	0.0711	0.1647
Epoque 11	<b>0.9845</b>	0.0606	0.2130
Epoque 12	0.9779	0.0590	0.2280
Epoque 13	0.9823	0.0555	0.1774
Epoque 14	0.9823	0.0499	0.2420
Epoque 15	0.9801	<b>0.0492</b>	0.1685

TABLE 21 – Squinky-Formality,  $p_0 = 0.5191$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.9054	0.3761	0.1168
Epoque 2	0.9183	0.2639	0.0658
Epoque 3	0.9204	0.2368	0.0713
Epoque 4	0.9183	0.2272	0.0914
Epoque 5	0.9161	0.2044	0.1491
Epoque 6	0.9204	0.1912	0.1065
Epoque 7	0.9204	0.1799	<b>0.0810</b>
Epoque 8	0.9183	0.1683	0.1155
Epoque 9	<b>0.9247</b>	0.1463	0.1105
Epoque 10	0.9183	0.1508	0.1290
Epoque 11	0.9161	0.1330	0.0972
Epoque 12	0.9161	0.1392	0.1651
Epoque 13	0.9161	0.1362	0.1536
Epoque 14	0.9161	0.1220	0.1763
Epoque 15	0.9161	<b>0.1085</b>	0.1277

TABLE 22 – Squinky-Informativeness,  $p_0 = 0.5410$ 

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.7873	0.9587	0.3110
Epoque 2	0.8261	0.4161	0.2861
Epoque 3	0.8385	0.3812	0.3147
Epoque 4	0.8432	0.3457	0.3526
Epoque 5	0.8478	0.3401	0.2782
Epoque 6	0.8416	0.3224	0.2104
Epoque 7	0.8571	0.3124	0.1613
Epoque 8	<b>0.8618</b>	0.2998	<b>0.1583</b>
Epoque 9	0.8524	0.2960	0.1598
Epoque 10	0.8463	0.2937	0.1779
Epoque 11	0.8494	0.2809	0.2057
Epoque 12	0.8525	0.2684	0.1871
Epoque 13	0.8525	0.2563	0.2197
Epoque 14	0.8525	0.2533	0.1947
Epoque 15	0.8509	<b>0.2531</b>	0.2511

TABLE 23 – Emobank-valence,  $p_0 = 0.5697$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.6064	0.6914	0.5813
Epoque 2	0.7184	0.5742	0.2939
Epoque 3	0.8050	0.4589	0.1525
Epoque 4	0.8306	0.3758	0.0696
Epoque 5	0.8450	0.3403	0.0642
Epoque 6	0.8586	0.3252	0.0802
Epoque 7	0.8680	0.3109	0.1114
Epoque 8	0.8595	0.2974	0.1003
Epoque 9	0.8641	0.2880	0.0909
Epoque 10	0.8546	0.2758	0.1147
Epoque 11	0.8706	0.2639	0.1436
Epoque 12	<b>0.8755</b>	0.2564	0.1426
Epoque 13	0.8534	0.2489	0.1597
Epoque 14	0.8733	0.2456	<b>0.0442</b>
Epoque 15	0.8748	<b>0.2376</b>	0.2303

TABLE 24 – Paws,  $p_0 = 0.5581$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.7386	0.9928	0.3585
Epoque 2	0.7784	0.5236	0.3625
Epoque 3	0.7727	0.4564	0.3437
Epoque 4	0.7841	0.4286	0.3719
Epoque 5	0.8068	0.3980	0.3713
Epoque 6	0.8049	0.3802	0.3549
Epoque 7	0.8125	0.3630	0.4010
Epoque 8	0.7955	0.3523	0.3783
Epoque 9	0.8125	0.3287	0.3857
Epoque 10	<b>0.8144</b>	0.3187	0.3918
Epoque 11	0.7992	0.3064	0.3878
Epoque 12	0.8125	0.3038	0.4108
Epoque 13	0.8106	0.2953	0.3780
Epoque 14	0.8106	0.3060	0.3829
Epoque 15	0.8106	<b>0.2904</b>	<b>0.3543</b>

TABLE 25 – Tweet Global Warning,  $p_0 = 0.7357$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.8256	0.5556	0.4868
Epoque 2	0.8240	0.4615	0.4165
Epoque 3	0.8320	0.4337	0.4157
Epoque 4	0.8352	0.4095	0.3968
Epoque 5	0.8336	0.4006	0.3895
Epoque 6	0.8352	0.3830	0.3874
Epoque 7	0.8320	0.3562	0.3734
Epoque 8	<b>0.8384</b>	0.3554	<b>0.3622</b>
Epoque 9	0.8256	0.3377	0.3652
Epoque 10	0.8288	0.3341	0.3966
Epoque 11	0.8320	0.3291	0.3845
Epoque 12	0.8304	0.3206	0.3941
Epoque 13	0.8304	0.3118	0.4068
Epoque 14	0.8304	0.3057	0.4026
Epoque 15	0.8288	<b>0.3027</b>	0.4172

TABLE 26 – Political Media Audience,  $p_0 = 0.2099$ 

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.7648	0.6349	0.2607
Epoque 2	0.7648	0.4786	0.2770
Epoque 3	0.7680	0.4541	0.2885
Epoque 4	0.7776	0.4440	0.2904
Epoque 5	0.7792	0.4196	0.3130
Epoque 6	0.7728	0.4048	<b>0.3004</b>
Epoque 7	0.7760	0.3947	0.3208
Epoque 8	<b>0.7840</b>	0.3892	0.3266
Epoque 9	0.7824	0.3782	0.3138
Epoque 10	0.7744	0.3638	0.3221
Epoque 11	0.7744	0.3532	0.3306
Epoque 12	0.7680	0.3488	0.3350
Epoque 13	0.7696	0.3398	0.3414
Epoque 14	0.7696	0.3306	0.3309
Epoque 15	0.7664	<b>0.3305</b>	0.3333

TABLE 27 – Political Media Bias,  $p_0 = 0.2611$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.0	0.6109	0.8488
Epoque 2	0.4173	0.5449	0.6736
Epoque 3	0.4708	0.4432	0.5289
Epoque 4	0.4995	0.4032	0.4551
Epoque 5	0.5329	0.3710	0.4562
Epoque 6	0.4940	0.3482	0.4120
Epoque 7	0.5274	0.3314	0.4029
Epoque 8	0.5868	0.3253	<b>0.3935</b>
Epoque 9	0.5601	0.3048	0.4193
Epoque 10	0.5654	0.2977	0.4030
Epoque 11	0.5870	0.2928	0.4130
Epoque 12	<b>0.5923</b>	0.2899	0.4017
Epoque 13	0.5815	0.2802	0.4438
Epoque 14	0.5920	0.2786	0.4476
Epoque 15	0.5868	<b>0.2747</b>	0.4258

TABLE 28 – CoLA,  $p_0 = 0.2956$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.9037	0.3554	0.2116
Epoque 2	0.9060	0.2708	0.1481
Epoque 3	0.9106	0.2368	0.1280
Epoque 4	0.9037	0.2085	0.1483
Epoque 5	0.9014	0.1853	0.0951
Epoque 6	0.9083	0.1723	<b>0.0938</b>
Epoque 7	0.9014	0.1585	0.1171
Epoque 8	0.9083	0.1454	0.1330
Epoque 9	<b>0.9128</b>	0.1377	0.2160
Epoque 10	0.9037	0.1320	0.1037
Epoque 11	0.8991	0.1218	0.1184
Epoque 12	0.9014	0.1190	0.1650
Epoque 13	0.9083	0.1136	0.1673
Epoque 14	0.9037	0.1112	0.2150
Epoque 15	0.9037	<b>0.1078</b>	0.2352

TABLE 29 – SST-2,  $p_0 = 0.4422$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.6936	0.6135	0.6404
Epoque 2	0.7108	0.5708	0.6426
Epoque 3	0.7230	0.5338	0.5856
Epoque 4	0.7402	0.4974	0.5747
Epoque 5	0.7451	0.4692	0.5585
Epoque 6	0.7623	0.4382	0.5362
Epoque 7	0.7598	0.4057	0.5374
Epoque 8	0.7647	0.3823	0.5204
Epoque 9	0.7770	0.3740	0.5334
Epoque 10	0.7770	0.3529	0.4931
Epoque 11	0.7819	0.3328	0.5054
Epoque 12	<b>0.7843</b>	0.3269	<b>0.4897</b>
Epoque 13	0.7794	0.3111	0.5151
Epoque 14	0.7843	0.3083	0.5049
Epoque 15	0.7843	<b>0.3068</b>	0.5040

TABLE 30 – MRPC,  $p_0 = 0.3255$ 

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.8689	0.4267	0.1627
Epoque 2	0.8784	0.3520	0.0681
Epoque 3	0.8858	0.3383	0.0327
Epoque 4	0.8869	0.3296	0.0154
Epoque 5	0.8876	0.3152	0.0105
Epoque 6	0.8891	0.2982	0.0210
Epoque 7	<b>0.8949</b>	0.2872	<b>0.0013</b>
Epoque 8	0.8923	0.2739	0.0186
Epoque 9	0.8938	0.2611	0.0017
Epoque 10	0.8934	0.2507	0.0330
Epoque 11	0.8934	0.2395	0.0054
Epoque 12	0.8934	0.2362	0.0718
Epoque 13	0.8916	0.2277	0.0228
Epoque 14	0.8912	0.2253	0.0247
Epoque 15	0.8912	<b>0.2199</b>	0.00420

TABLE 31 – QNLI,  $p_0 = 0.5001$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.5580	1.2359	0.1357
Epoque 2	0.5725	0.7635	0.3859
Epoque 3	0.6449	0.6174	0.4531
Epoque 4	0.6522	0.5605	0.4277
Epoque 5	0.6739	0.5215	0.3828
Epoque 6	0.6739	0.4917	0.3221
Epoque 7	0.6812	0.4561	0.3311
Epoque 8	0.6884	0.4398	0.2715
Epoque 9	<b>0.7101</b>	0.4068	0.2577
Epoque 10	0.6884	0.3938	0.2482
Epoque 11	0.6957	0.3834	0.2301
Epoque 12	0.6884	0.3720	0.2024
Epoque 13	0.7029	0.3570	0.1975
Epoque 14	0.6957	0.3501	0.1952
Epoque 15	0.7029	<b>0.3498</b>	<b>0.1909</b>

TABLE 32 – RTE,  $p_0 = 0.5016$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.9745	0.4379	0.033
Epoque 2	0.9929	0.0914	$2.3 \cdot 10^{-5}$
Epoque 3	0.9963	0.0304	$3.0 \cdot 10^{-6}$
Epoque 4	0.9988	0.0177	0.0272
Epoque 5	0.9988	0.0103	0.0418
Epoque 6	0.9988	0.0077	0.0010
Epoque 7	<b>0.9991</b>	0.0062	$3.1 \cdot 10^{-14}$
Epoque 8	0.9991	0.0045	0.2367
Epoque 9	0.9991	0.0050	0.1496
Epoque 10	0.9988	0.0031	0.1873
Epoque 11	0.9991	0.0023	0.0599
Epoque 12	0.9991	0.0018	0.2367
Epoque 13	0.9991	0.0023	0.1631
Epoque 14	0.9988	0.0014	0.2042
Epoque 15	0.9991	<b>0.0012</b>	0.1884

TABLE 33 – Syntax Semantics,  $p_0 = 0.4991$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.9800	0.4901	0.0060
Epoque 2	0.9920	0.0815	$9.4 \cdot 10^{-6}$
Epoque 3	0.9880	0.0681	$2.4 \cdot 10^{-8}$
Epoque 4	0.9800	0.0488	$8.4 \cdot 10^{-5}$
Epoque 5	0.9840	0.0439	0.0247
Epoque 6	0.9840	0.0464	0.1508
Epoque 7	0.9880	0.0511	0.0005
Epoque 8	0.9840	0.0351	$2.2 \cdot 10^{-12}$
Epoque 9	0.9840	0.0208	0.1586
Epoque 10	0.9880	0.0394	0.2110
Epoque 11	<b>0.9960</b>	0.0238	$1.7 \cdot 10^{-15}$
Epoque 12	0.9960	0.0135	0.0916
Epoque 13	0.9960	0.0211	0.1700
Epoque 14	0.9960	0.0135	0.1393
Epoque 15	0.9960	<b>0.0111</b>	0.0775

TABLE 34 – Syntax+Semantics,  $p_0 = 0.4907$ 

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.9851	0.2886	0.1932
Epoque 2	0.9913	0.0440	0.2264
Epoque 3	0.9940	0.0251	0.0135
Epoque 4	0.9940	0.0178	0.3002
Epoque 5	0.9953	0.0140	0.1309
Epoque 6	0.9942	0.0101	0.0109
Epoque 7	0.9942	0.0083	0.1153
Epoque 8	<b>0.9962</b>	0.0078	0.1027
Epoque 9	0.9958	0.0059	0.0129
Epoque 10	0.9960	0.0055	$8.2 \cdot 10^{-5}$
Epoque 11	0.9951	0.0038	0.1992
Epoque 12	0.9951	0.0040	0.2249
Epoque 13	0.9953	0.0028	0.2408
Epoque 14	0.9956	0.0027	0.0286
Epoque 15	0.9956	<b>0.0021</b>	0.1712

TABLE 35 – Morphology,  $p_0 = 0.5015$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.9691	0.3709	0.0177
Epoque 2	0.9878	0.1075	$3.1 \cdot 10^{-6}$
Epoque 3	0.9909	0.0585	$2.1 \cdot 10^{-7}$
Epoque 4	0.9934	0.0394	$1.1 \cdot 10^{-5}$
Epoque 5	0.9923	0.0298	0.0143
Epoque 6	0.9937	0.0211	0.0005
Epoque 7	0.9940	0.0172	0.0747
Epoque 8	0.9942	0.0135	0.0610
Epoque 9	<b>0.9948</b>	0.0116	0.090
Epoque 10	0.9940	0.0088	0.1624
Epoque 11	0.9943	0.0113	0.1528
Epoque 12	0.9938	0.0070	$1.8 \cdot 10^{-5}$
Epoque 13	0.9946	0.0098	$1.7 \cdot 10^{-5}$
Epoque 14	0.9948	0.0080	0.0260
Epoque 15	0.9946	<b>0.0069</b>	0.1988

TABLE 36 – Syntax,  $p_0 = 0.4988$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.7694	0.7024	0.3914
Epoque 2	0.8824	0.4178	0.1783
Epoque 3	0.9030	0.3396	0.0685
Epoque 4	0.9229	0.3120	0.0416
Epoque 5	0.9315	0.2873	0.0219
Epoque 6	0.9372	0.2732	0.0025
Epoque 7	0.9389	0.2555	0.0018
Epoque 8	0.9435	0.2272	0.0159
Epoque 9	0.9463	0.2063	0.0259
Epoque 10	0.9475	0.1880	<b>0.0010</b>
Epoque 11	0.9481	0.1768	0.0311
Epoque 12	0.9481	0.1607	0.0019
Epoque 13	<b>0.9509</b>	0.1571	0.0017
Epoque 14	0.9492	0.1510	0.0192
Epoque 15	0.9498	<b>0.1456</b>	0.0022

TABLE 37 – Recast Puns,  $p_0 = 0.5$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.8913	0.4420	0.0495
Epoque 2	0.9174	0.2916	0.0134
Epoque 3	0.9210	0.2499	0.0086
Epoque 4	0.9268	0.2117	0.0086
Epoque 5	0.9255	0.1834	0.0254
Epoque 6	0.9302	0.1606	0.0048
Epoque 7	0.9303	0.1384	0.0079
Epoque 8	0.9270	0.1296	0.0061
Epoque 9	<b>0.9307</b>	0.1166	0.0001
Epoque 10	0.9287	0.1037	0.0004
Epoque 11	0.9294	0.0950	0.01318
Epoque 12	0.9300	0.0845	$4.0 \cdot 10^{-5}$
Epoque 13	0.9294	0.0855	0.0003
Epoque 14	0.9296	0.0792	0.0049
Epoque 15	0.9294	<b>0.0782</b>	0.0053

TABLE 38 – Recast Factuality,  $p_0 = 0.5$ 

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.6434	0.6661	0.8305
Epoque 2	0.6434	0.6547	0.8499
Epoque 3	0.6434	0.6317	0.8548
Epoque 4	0.6363	0.6081	0.8692
Epoque 5	0.7133	0.5750	0.8546
Epoque 6	0.7203	0.5397	0.8096
Epoque 7	<b>0.7343</b>	0.4991	0.7869
Epoque 8	0.7343	0.4614	0.7305
Epoque 9	0.7343	0.4346	0.7086
Epoque 10	0.7273	0.4116	0.6614
Epoque 11	0.7203	0.3823	0.6368
Epoque 12	0.7203	0.3679	0.5873
Epoque 13	0.7273	0.3583	0.5932
Epoque 14	0.7273	0.3475	0.5586
Epoque 15	0.7343	<b>0.3470</b>	<b>0.5322</b>

TABLE 39 – Recast Verbnet,  $p_0 = 0.4034$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.6929	0.7097	0.8389
Epoque 2	0.7867	0.6876	0.8143
Epoque 3	0.9350	0.4991	0.5003
Epoque 4	0.9317	0.3156	0.2998
Epoque 5	0.9317	0.2081	0.1512
Epoque 6	0.9250	0.1510	0.0016
Epoque 7	0.9300	0.1212	0.1029
Epoque 8	<b>0.9317</b>	0.1039	0.1426
Epoque 9	0.9267	0.0968	0.0618
Epoque 10	0.9267	0.0790	0.0895
Epoque 11	0.9283	0.0715	0.2302
Epoque 12	0.9300	0.0664	0.1714
Epoque 13	0.9317	0.0667	0.2876
Epoque 14	0.9283	0.0561	0.2619
Epoque 15	0.9300	0.0531	<b>1.1 · 10<sup>-8</sup></b>

TABLE 40 – Recast Sentiment,  $p_0 = 0.5$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.8173	0.4705	0.2136
Epoque 2	0.8469	0.3716	0.2607
Epoque 3	0.8545	0.3482	0.3048
Epoque 4	0.8655	0.3282	0.2829
Epoque 5	0.8765	0.3123	0.3434
Epoque 6	0.8756	0.3139	<b>0.3398</b>
Epoque 7	0.8680	0.2992	0.3737
Epoque 8	0.8739	0.2889	0.3632
Epoque 9	0.8714	0.2840	0.3894
Epoque 10	0.8739	0.2738	0.3863
Epoque 11	<b>0.8782</b>	0.2726	0.4143
Epoque 12	0.8739	0.2596	0.4057
Epoque 13	0.8714	0.2617	0.4078
Epoque 14	0.8714	0.2510	0.3618
Epoque 15	0.8739	<b>0.2463</b>	0.4000

TABLE 41 – Recast megaveridicality,  $p_0 = 0.6667$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.6305	0.6827	0.6013
Epoque 2	0.6158	0.6260	0.4885
Epoque 3	0.6011	0.5839	0.4235
Epoque 4	0.6078	0.5539	0.3849
Epoque 5	0.6075	0.5288	0.3564
Epoque 6	0.6107	0.5054	0.3314
Epoque 7	0.6158	0.4850	0.3153
Epoque 8	0.6149	0.4647	0.2956
Epoque 9	0.6648	0.4361	0.2755
Epoque 10	0.6671	0.4252	0.2701
Epoque 11	0.6910	0.4189	0.2440
Epoque 12	0.6749	0.4055	0.2931
Epoque 13	<b>0.7015</b>	0.3927	<b>0.2428</b>
Epoque 14	0.6922	0.3822	0.2730
Epoque 15	0.6969	<b>0.3784</b>	0.2773

TABLE 42 – Boolq,  $p_0 = 0.3769$ 

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.5705	0.7704	0.5585
Epoque 2	0.5925	0.6242	0.4641
Epoque 3	0.6176	0.5754	0.4070
Epoque 4	0.6082	0.5351	0.3746
Epoque 5	0.6207	0.4999	0.3042
Epoque 6	0.6332	0.4670	0.2702
Epoque 7	0.6395	0.4367	0.2380
Epoque 8	0.6426	0.4241	0.2198
Epoque 9	<b>0.6458</b>	0.3940	0.1811
Epoque 10	0.6426	0.3661	0.1532
Epoque 11	0.6458	0.3576	0.1352
Epoque 12	0.6395	0.3426	0.1278
Epoque 13	0.6458	0.3456	<b>0.1147</b>
Epoque 14	0.6458	0.3297	0.1149
Epoque 15	0.6458	<b>0.3188</b>	0.1202

TABLE 43 – Wic,  $p_0 = 0.5$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.8938	0.3801	0.5702
Epoque 2	0.9040	0.2659	0.6289
Epoque 3	0.9259	0.2448	0.6640
Epoque 4	0.9258	0.2129	0.6754
Epoque 5	0.9381	0.1888	0.6518
Epoque 6	0.9374	0.1705	0.7150
Epoque 7	0.9428	0.1575	0.6975
Epoque 8	0.9330	0.1422	0.6940
Epoque 9	0.9415	0.1267	0.6886
Epoque 10	0.9435	0.1208	0.6906
Epoque 11	<b>0.9479</b>	0.1118	<b>0.6835</b>
Epoque 12	0.9459	0.1056	0.7044
Epoque 13	0.9479	0.1037	0.6913
Epoque 14	0.9476	0.0976	0.7014
Epoque 15	0.9449	<b>0.0927</b>	0.7206

TABLE 44 – ADE Corpus V2 classification,  $p_0 = 0.7113$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.7110	0.7078	0.3452
Epoque 2	0.7300	0.4933	0.3122
Epoque 3	0.7290	0.4495	0.2422
Epoque 4	0.7390	0.4118	0.2103
Epoque 5	0.7340	0.3926	0.1910
Epoque 6	0.7460	0.3738	0.1621
Epoque 7	0.7450	0.3495	0.1427
Epoque 8	0.7460	0.3505	0.1462
Epoque 9	0.7480	0.3181	0.1190
Epoque 10	0.7490	0.3159	0.1109
Epoque 11	<b>0.7570</b>	0.2953	0.1270
Epoque 12	0.7520	0.3019	0.0820
Epoque 13	0.7570	0.2978	<b>0.0705</b>
Epoque 14	0.7570	0.2840	0.0978
Epoque 15	0.7530	<b>0.2829</b>	0.1025

TABLE 45 – Hate,  $p_0 = 0.5797$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.6251	1.0525	0.4478
Epoque 2	0.6398	0.6523	0.5089
Epoque 3	0.6419	0.6070	0.4953
Epoque 4	0.6586	0.5721	0.4340
Epoque 5	0.6639	0.5416	0.3916
Epoque 6	0.6670	0.5308	0.3698
Epoque 7	0.6545	0.4979	0.3182
Epoque 8	<b>0.6764</b>	0.4812	0.3010
Epoque 9	0.6712	0.4665	0.2537
Epoque 10	0.6723	0.4502	0.2606
Epoque 11	0.6733	0.4336	<b>0.2497</b>
Epoque 12	0.6691	0.4212	0.2548
Epoque 13	0.6691	0.4116	0.2572
Epoque 14	0.6670	0.4005	0.2807
Epoque 15	0.6660	<b>0.4002</b>	0.2805

TABLE 46 – Irony,  $p_0 = 0.4951$ 

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.7659	0.5493	0.2980
Epoque 2	0.7795	0.4651	0.3063
Epoque 3	0.7810	0.4321	0.3246
Epoque 4	0.7893	0.4116	0.2778
Epoque 5	0.7885	0.3933	0.2758
Epoque 6	0.7832	0.3833	0.2748
Epoque 7	0.7878	0.3665	0.2725
Epoque 8	<b>0.7900</b>	0.3570	0.2726
Epoque 9	0.7863	0.3459	<b>0.2613</b>
Epoque 10	0.7813	0.3289	0.2676
Epoque 11	0.7795	0.3266	0.2976
Epoque 12	0.7810	0.3240	0.2709
Epoque 13	0.7778	0.3167	0.3067
Epoque 14	0.7742	0.3181	0.2867
Epoque 15	0.7719	<b>0.3141</b>	0.2876

TABLE 47 – Offensive,  $p_0 = 0.6693$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.8058	0.6416	0.2737
Epoque 2	0.8208	0.4102	0.1701
Epoque 3	0.8293	0.3681	0.1221
Epoque 4	0.8377	0.3371	0.1054
Epoque 5	0.8405	0.3169	0.0680
Epoque 6	0.8396	0.3054	0.0349
Epoque 7	0.8368	0.2946	0.0230
Epoque 8	<b>0.8424</b>	0.2752	0.0149
Epoque 9	0.8386	0.2743	0.0420
Epoque 10	0.8349	0.2643	0.0111
Epoque 11	0.8340	0.2539	0.0045
Epoque 12	0.8321	0.2306	0.0070
Epoque 13	0.8340	0.2385	0.0034
Epoque 14	0.8358	0.2199	0.0043
Epoque 15	0.8340	<b>0.2180</b>	<b>0.0023</b>

TABLE 48 – Rotten Tomatoes,  $p_0 = 0.5$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.5380	0.6915	0.5792
Epoque 2	0.5330	0.6191	0.4945
Epoque 3	0.5615	0.6016	0.4906
Epoque 4	0.5675	0.5886	0.4801
Epoque 5	0.5775	0.5727	0.4393
Epoque 6	0.5750	0.5584	0.4061
Epoque 7	0.5910	0.5473	0.3958
Epoque 8	0.6015	0.5349	0.3792
Epoque 9	0.5950	0.4233	0.3663
Epoque 10	0.5995	0.5145	0.3362
Epoque 11	0.5995	0.5066	0.3451
Epoque 12	0.6030	0.5003	0.3273
Epoque 13	0.6030	0.4917	0.3060
Epoque 14	0.6080	0.4893	0.2951
Epoque 15	<b>0.6090</b>	<b>0.487</b>	<b>0.2900</b>

TABLE 49 – Hover,  $p_0 = 0.3934$

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.8150	0.5536	0.2162
Epoque 2	0.7950	0.4022	0.0917
Epoque 3	0.8300	0.3520	0.0691
Epoque 4	0.8300	0.3082	0.0643
Epoque 5	0.8150	0.2838	0.0369
Epoque 6	0.8200	0.2798	0.0271
Epoque 7	0.8200	0.2525	0.0258
Epoque 8	0.8150	0.2404	0.0257
Epoque 9	0.8450	0.1970	0.0117
Epoque 10	0.8100	0.1923	0.0135
Epoque 11	<b>0.8500</b>	0.1829	0.0122
Epoque 12	0.8300	0.1956	0.0120
Epoque 13	0.8200	0.1799	0.0104
Epoque 14	0.8500	0.1623	<b>0.0048</b>
Epoque 15	0.8300	<b>0.1567</b>	0.0066

TABLE 50 – Movie Rationales,  $p_0 = 0.5$ 

Epoque	Exactitude de validation	Perf. sur le corpus d'entraînement	
		Emp. Risk	Unsup. Risk
Epoque 1	0.6182	0.7844	0.4631
Epoque 2	0.6288	0.6335	0.3909
Epoque 3	0.6493	0.6047	0.3642
Epoque 4	0.6509	0.5798	0.3579
Epoque 5	0.6612	0.5626	0.2997
Epoque 6	0.6593	0.5452	0.3065
Epoque 7	0.6537	0.5319	0.3226
Epoque 8	0.6574	0.5173	0.2909
Epoque 9	0.6655	0.5050	0.2638
Epoque 10	<b>0.6705</b>	0.5005	0.2707
Epoque 11	0.6665	0.4889	0.2955
Epoque 12	0.6643	0.4755	0.2565
Epoque 13	0.6655	0.4758	0.2749
Epoque 14	0.6655	0.4719	0.2657
Epoque 15	0.6646	<b>0.4687</b>	<b>0.2395</b>

TABLE 51 – Eraser Multi RC,  $p_0 = 0.5600$