

Introduction au Deep Learning

Basé sur Keras-Tensorflow

Vincent Boyer
vincent.boyer1@live.fr

Mars 2018

Contents

1	Préambule	15
2	Paradigme de l'Induction	16
2.1	Qu'est-ce que l'Induction ?	16
2.2	Les différentes formes d'apprentissage	17
2.2.1	Apprentissage supervisé	17
2.2.2	Apprentissage non supervisé	18
2.2.3	Apprentissage semi-supervisé	19
2.2.4	Apprentissage par transfert	20
2.2.4.1	Instance Transfer	22
2.2.4.2	Feature Representation Transfer	22
2.2.4.3	Parameter Transfer	23
2.2.4.4	Relational Knowledge Transfer	23
2.2.4.5	Approches et configurations	23
2.2.5	Apprentissage par renforcement	24
2.3	Formalisation d'un problème d'Induction	25
2.4	Quelques principes d'induction	26
2.4.1	Le principe ERM	26
2.4.2	Le principe de décision bayésienne	27
2.5	Evaluation de l'apprentissage	28
2.6	Théorie de l'apprentissage	29
2.6.1	PAC Learning (Probably Approximately Correct)	29
2.6.1.1	Qu'est-ce qu'un problème d'apprentissage ?	29
2.6.1.2	Formalisation du PAC Learning	31
2.6.1.3	PAC Learning et espace d'hypothèses	32
2.6.1.4	Agnostic PAC Learning	32
2.6.1.5	Généralisation aux espaces de concepts continus	35
2.6.2	Exemple d'une étude par PAC Learning - Le principe ERM dans le cas fini	35
2.7	Théorie de la régularisation	37
2.7.1	Contexte et environnement	37
2.7.2	Définition de la régularisation	38
2.7.3	Les formes de la régularisation	39
2.8	No Free-lunch Theorem, un fondamental théorique (et grande désillusion ?)	39
3	Les Fondamentaux	42
3.1	Deep Learning, le fer de lance du Machine Learning	42
3.2	Un neurone: le perceptron	44
3.3	Le biais	45
3.4	Le Perceptron Multicouche	45
3.5	L'apprentissage du neurone	47
3.5.1	Etape Forward: Fonction de coût	47

3.5.2	Etape Backward: Rétropropagation du gradient	49
3.5.2.1	Les méthodes de descente	49
3.5.2.2	La direction de descente	50
3.5.2.3	Algorithme des méthodes à direction de descente	50
3.5.2.4	Détermination de la direction de descente	50
3.5.2.5	Rétropropagation du gradient - A FAIRE	52
3.5.2.6	Fonction d'activation	52
3.6	Descente du gradient et optimiseur	55
3.6.1	Approche par Batch, Minibatch et Stochastique	55
3.6.1.1	Approche par Batch	55
3.6.1.2	Approche Stochastique	55
3.6.1.3	Approche par MiniBatch	56
3.6.2	La problématique de la détermination du pas - A FAIRE	57
3.6.3	Aperçu des optimizer	57
3.6.3.1	SGD Annealing	57
3.6.3.2	Approche par cycle: Cyclical Learning Rate	59
3.6.3.3	Détermination des bornes supérieures et inférieures des valeurs du pas: le LR Range test	60
3.6.3.4	Approche par cycle: Exploring Stochastic Gra- dient Descent with Warm Restarts (SGDR)	61
3.6.3.5	1Cycle Policy et Super-Convergence: A FAIRE	62
3.6.3.6	Snapshot Ensembles	62
3.6.3.7	Calcul du moment: Vanilla Momentum	63
3.6.3.8	Calcul du moment: Nesterov Accelerated gradi- ent (NAG)	63
3.6.3.9	Méthode adaptative: Adagrad	64
3.6.3.10	Méthode adaptative: AdaDelta	64
3.6.3.11	Méthode adaptative: RMSprop	66
3.6.3.12	Méthode adaptative: Adam	66
3.6.3.13	Méthode adaptative: AdaMax	68
3.6.3.14	Méthode adaptative: Nadam	68
3.6.3.15	Adam et correctifs théoriques - AdamW	69
3.6.3.16	Adam et correctifs théoriques - AMSGrad	71
3.6.3.17	Adam et correctifs théoriques - Nostalgic Adam	72
3.6.3.18	Méthode adaptative cyclique - AdamWR	74
3.6.3.19	Discriminative Fine-Tuning	74
3.6.3.20	Et les méthodes d'optimisation du 2nd ordre ? - A FAIRE	76
3.6.3.21	SGD, calcul distribué et parallélisation - A FAIRE	76
3.6.3.22	Inférence de l'optimizer - Neural Optimization Search	76
3.6.3.23	Inférence de l'optimizer - LSTM approach	76
3.6.3.24	Quel optimizer choisir ?	76
3.6.4	Gradient noise	77
3.6.5	Gradient Clipping	77
3.7	ReLU et les dangers du gradient	77

3.7.1	Vanishing Gradient	78
3.7.2	Exploding Gradient	78
3.7.3	Fonction ReLu et Dead ReLU	78
3.7.3.1	Les avantages et le danger de ReLU	78
3.7.3.2	Leaky ReLU (LReLU)	79
3.7.3.3	Randomized ReLU (RReLU)	80
3.7.3.4	Parameterized ReLU (PReLU)	80
3.7.3.5	Exponential ReLU (ELU)	81
3.7.3.6	Concatenated ReLU (CReLU)	82
3.7.3.7	Scaled Exponential Linear Units (SELU)	82
3.8	Initialisation des hyperparamètres - A FAIRE	83
3.8.1	Initialisation des poids	83
3.8.2	Initialisation et détermination des hyperparamètres - A FAIRE	84
3.9	Jeu d'apprentissage et spécificités	84
3.10	Prédiction multi-label et multi-classe	85
3.10.1	Généralités	85
3.10.2	Prédiction et distribution de données	87
3.10.3	Méthodes d'apprentissage	88
3.10.3.1	Au niveau des données	88
3.10.3.2	Au niveau du modèle	89
3.11	Calcul matriciel et neurones	90
4	Normalisation des données	93
4.1	Centrer les données	94
4.2	Réduire les données	94
4.3	Centrer-Réduire les données	94
4.4	Analyse en Composante Principale	95
5	Régularisation et sur-apprentissage	97
5.1	Le sur-apprentissage	97
5.2	Limiter le sur-apprentissage	98
5.3	Régularisation	99
5.3.1	L1-Régulation	99
5.3.2	L2-Régulation	100
5.3.3	ElasticNet-Régulation	100
5.4	Max norm constraints	100
5.5	DropOut	101
5.6	DropConnect	101
5.7	DisturbLabel	102
5.8	Dense-Sparse-Dense training	103
5.9	Batch Normalization	103
5.10	Critère d'arrêt de l'apprentissage	104
5.10.1	Early Stopping	106
5.10.1.1	Generalization Loss (GL_α)	106

5.10.1.2	Quotient of Generalization Loss and Progress (PQ_{alpha})	107
5.10.1.3	Successive Generalization Error (UP_s)	107
5.10.2	Arrêt supervisé	108
5.11	Data Augmentation	108
5.12	Complexité de l'architecture et mémoire	109
5.13	Transfer Learning	111
5.13.0.1	Inductive Transfer - Parameter Transfer	111
5.13.0.2	Transductive Transfer - Domain Adaptation - A FAIRE	114
5.14	Réseaux de neurones et méthodes d'Ensemble - A FAIRE	114
5.14.1	Bagging - A FAIRE	114
5.14.2	Boosting - A FAIRE	114
6	Réseaux convolutifs	115
6.1	Généralités	115
6.2	Couche d'Entrée	117
6.3	Couche de Convolution	117
6.3.1	Nature d'une couche de convolution	117
6.3.2	Interprétation graphique	118
6.3.3	Couche de Pooling	121
6.3.3.1	Global Pooling	124
6.3.3.2	k-Max Pooling	125
6.3.4	Couche de Unpooling	126
6.3.5	Couche de convolution 1*1	127
6.3.6	Couche de convolution dilatée	128
6.3.7	Couche de convolution transposée	128
6.3.8	Couche de Depthwise/Pointwise	129
6.4	Conversion d'une couche Full-Connected en couche de convolution	131
6.5	Convolutional Neural Network (CNN) et Fully Convolutional Network (FCN)	132
6.6	Les modèles convolutifs de référence	133
6.6.1	AlexNet	133
6.6.2	ZF Net	133
6.6.3	VGG Net	133
6.6.4	GoogleNet/Inception	134
6.6.5	Xception	135
6.6.6	Microsoft Resnet	137
6.6.7	ResNet et améliorations	137
6.6.7.1	Wide Residual Network	138
6.6.7.2	Aggregated Residual Transformations for Deep Neural Networks (ResNext)	138
6.6.7.3	Densely Connected Convolutionnal Networks (DenseNet)	139
6.6.7.4	Stochastic Depth	141
6.6.7.5	Residual Networks of Residual Networks: Mul- tilevel Residual Networks	142

6.6.7.6	Sparsely Connected Convolutional Networks (SparseNet)	143
6.6.7.7	Résumé des différentes approches appliquées aux ResNet	144
6.6.8	FractalNet: Ultra-Deep Neural Networks without Residuals	144
6.6.9	Classement des architectures standards	145
6.6.10	Réseaux de neurones compressés	146
6.6.10.1	SqueezeNet	147
6.6.10.2	MobileNet	148
6.6.10.3	Modèles expérimentaux récents	150
6.6.11	Les jeux de données de référence	150
7	Encoder-Decoder	152
7.1	Généralités	152
7.2	Autoencoder	152
7.2.1	Autoencoder Undercomplete	153
7.2.2	Autoencoder Régularisé ou Overcomplete	154
7.2.2.1	Denoising Autoencoder	154
7.2.2.2	Sparse Autoencoder	155
7.2.2.3	Contractive Autoencoders	158
7.3	Variational Autoencoders	159
8	Réseaux antagonistes génératifs	162
8.1	Généralités	162
8.1.0.1	Fonction de perte	162
8.2	Difficultés d'apprentissage	165
8.3	Deep Convolutional Generative Adversarial Networks (DCGAN)	166
9	Réseaux siamois	167
9.1	Généralités	167
9.2	Triplet Loss	168
10	Réseaux récurrents	170
10.1	Généralités théoriques	170
10.1.1	Description d'un RNN	170
10.1.2	Présentation théorique	171
10.1.3	Implémentation et représentation matricielle	172
10.1.4	Apprentissage d'un RNN	172
10.1.5	Problématiques d'un RNN Vanilla	172
10.1.5.1	Unilatéralité du contexte	172
10.1.5.2	Mémoire court terme	173
10.1.5.3	Coût Calcul et représentativité de l'état caché	173
10.2	Long Short-Term Memory (LSTM) - A FAIRE	174
10.3	Gated Recurrent Unit (GRU) - A FAIRE	174
10.4	BiRNN - A FAIRE	174
10.5	Architecture-type	174

11 Deep Learning et Attention	176
11.1 Généralités	176
11.2 Soft Attention	177
11.2.1 Fonction de proximité	178
11.3 Hard Attention	178
11.4 Réseaux récurrents	179
11.4.1 Approche Bahdanau	179
11.4.2 Approche Luong	180
11.4.2.1 Attention globale et locale	181
11.4.3 Généralisation pour l'exploitation d'image: Image Captionning	183
11.5 Réseaux convolutifs	184
11.5.1 Spatial Transformer	185
11.5.1.1 Transformation d'image	185
11.5.1.2 Interpolation bilinéaire	187
11.5.1.3 Le module SNT	188
11.5.2 Squeeze-and-Excitation Networks (SENet)	190
11.5.2.1 Concurrent Spatial and Channel Squeeze and excitation	191
12 L'analyse d'image et ses formes	193
12.1 Les différentes thématiques de l'analyse d'image	193
12.2 Généralités théoriques	195
12.2.1 Théorie - Object Detection, un problème de Régression	195
12.2.2 Théorie - Landmark detection, un problème de Régression	197
12.2.3 Sliding Windows	198
12.2.4 Sliding Windows et réseau convolutif	200
12.2.5 Region Proposal	201
12.2.5.1 Selective Search	202
12.2.6 Anchor Boxes	202
12.2.7 RoI Pooling	204
12.2.8 Intersection over Union (IoU)	206
12.2.9 Non-Max Suppression	207
12.2.10 Feature Pyramid	208
12.2.10.1 Amélioration de l'architecture du réseau central	210
12.2.10.2 A faire	210
12.3 Object recognition: méthodes State-of-the-art	210
12.3.1 R-CNN - Algorithme précurseur	212
12.3.2 Spatial Pyramid Pooling - Algorithme précurseur	212
12.3.3 Fast R-CNN - Algorithme précurseur	214
12.3.4 Faster R-CNN	215
12.3.5 Différence principale entre Single shot et Region based	218
12.3.6 You Only Look Once (YOLO)	219
12.3.6.1 YOLOv1	219
12.3.6.2 YOLOv2 - YOLO9000	223
12.3.6.3 YOLOv3	227

12.3.7	Single Shot MultiBox Detector (SSD)	230
12.3.7.1	SSD	230
12.3.7.2	Tiny SSD	232
12.3.7.3	Deconvolutional Single Shot Detector (DSSD) .	233
12.3.8	RetinaNet	235
12.3.8.1	Focal Loss	235
12.3.8.2	Le modèle RetinaNet	236
12.3.9	Comparatif des modèles récents	237
13 Application au traitement du langage écrit		239
13.1	Introduction	239
13.1.1	La loi de Zipf et Mandelbrot	239
13.1.2	Pré-traitement d'un texte	239
13.1.2.1	Tokenization	240
13.1.2.2	Lemmatisation et racinisation - A FAIRE . . .	241
13.1.2.3	Stopwords	241
13.1.2.4	Problématique et pré-traitement	241
13.2	Représentation vectorielle - A FAIRE	242
13.2.1	Projection Word-Based - A FAIRE	242
13.2.2	Projection Character-Based	242
13.3	Classification	242
13.3.1	Classification par CNN	243
13.3.1.1	CNN for Sentence Classification	243
13.3.1.2	Améliorations notables	244
13.3.1.3	Very Deep CNN	244
13.3.1.4	Comparatif expérimental sur les architectures CNN	247
13.3.2	Classification par RNN	248
13.3.2.1	LSTM Deep Sentence Embedding	248
13.3.2.2	Discriminative RNN	248
13.3.2.3	Hierarchical Attention Networks	249
13.3.3	Classification par CNN-RNN	249
13.3.3.1	CNN-RNN	249
13.3.3.2	C-LSTM	251
13.3.3.3	AC-BLSTM	252
14 Apprentissage par Renforcement et Deep Learning		255
14.1	Approche théorique	255
14.1.1	Généralités	255
14.1.1.1	L'environnement	256
14.1.1.2	Formalisation du problème	257
14.1.1.3	Définitions fondamentales	259
14.1.1.4	L'équation fondatrice: l'équation de Bellman .	260
14.1.1.5	Model-Free et Model-Based	261
14.1.1.6	Différence temporelle	262
14.1.1.7	Un standard: l'algorithme Q-Learning	263
14.1.1.8	Approximation des valeurs d'états	266

14.2 Deep Q-Learning	266
14.3 Prioritized Experience Replay	268
14.4 Double Deep Q-Learning	270
14.5 Dueling Deep Q-Learning	271
14.6 Double Dueling Q-Learning	272
15 Ajouts à venir	273
15.1 Application à la reconnaissance vocale	273
15.2 Application au traitement du langage écrit	273
15.2.1 Traduction - Neural Machine Translation	273
15.2.2 Recherche d'informations	273
15.2.3 Désambiguïsation d'entités	273
15.3 L'analyse d'image et ses formes	273
15.3.1 Segmentation: méthodes State-of-the-art	273
15.3.2 Object Tracking: méthodes State-of-the-art	273
15.4 Machine Learning et Éthique	273
15.4.1 adversarial-examples	273
15.5 Deep Learning Bayésien	273
15.6 Neuroevolution	273
15.7 Machine de Turing neuronale	273
15.8 Geometric Deep Learning	273
15.9 Neural Architecture Search (NAS)	273
15.10 Transfer Learning	273
15.10.1 Domain Adaptation	273
15.11 Few Shot Learning	273

List of Figures

1	Différence entre apprentissage supervisé en non supervisé	19
2	Les différentes approches d'apprentissage par transfert	22
3	Configurations possibles des différentes approches de Transfer Learning	24
4	Erreur entre l'hypothèse-cible et l'hypothèse apprise	36
5	Illustration d'une régularisation non paramétrique	38
6	La fin du rêve: No Free Lunch Theorem	40
7	Relation entre profondeur du réseau et capacité d'abstraction . .	43
8	La structure d'un neurone (perceptron)	44
9	Modèle FeedForward: Le Perceptron multicouche	46
10	Illustration de la descente de gradient	52
11	Fonctions d'activation: a) Sigmoïde, b) tanh, c) ReLu et leurs dérivées associées	53
12	Différence entre minimum global, maximum global et point-selle	59
13	Comportement du pas avec Cyclical LR avec une borne supérieure constante et une borne supérieure dégressive	60
14	Exemple du LR Range test	61
15	Comportement du pas avec SGDR	62
16	Différence entre SGD et Snapshot Ensembles	62
17	Différence entre Momentum et Nesterov Momentum	63
18	Discriminative Fine-Tuning sur un réseau CNN standard	75
19	Fonction ReLu et ses Variantes	80
20	Exemple de deux filtres de phase opposée	82
21	Exemple d'un jeu de données (toute ressemblance avec des données réelles est fortuite)	93
22	Exemple de sur-apprentissage (bleu) et d'apprentissage optimal (vert)	97
23	Compromis biais-variance et impact sur le modèle	98
24	Exemple de sous-apprentissage	99
25	Comparaison d'un réseau sous DropOut et DropConnect	102
26	Relation des poids après DropOut	102
27	Dense-Sparse-Dense Routine	104
28	Non-alignement des distributions	105
29	Normalisation par Batch Normalization	105
30	a) Early Stop théorique b) Comportement-type de l'erreur de validation en situation réelle	106
31	Exemple de courbe d'erreur d'apprentissage et de validation . .	109
32	Optimisation réalisée sur les matrices de poids par Deep Compression	111
33	Relation entre l'abstraction d'un réseau convolutif et sa profondeur	116
34	Architecture d'une couche de convolution	118
35	Impact du stride sur la sortie de la convolution	120
36	Exemple d'une convolution par filtre	120

37	Problématique de la dimension du filtre: gauche (valide), droite (invalidé)	121
38	Application du 0-Padding	121
39	Application du Pooling: Réduction et exemple avec la fonction max selon un filtre 2*2	123
40	Exemple de Max-Unpooling statique	127
41	Exemple de Max-Unpooling dynamique	127
42	Exemple d'une convolution dilatée	128
43	Exemple de convolution transposée: Gauche: entrée 2*2 sans stride, Droite: Entrée 3*3 avec stride de 1	129
44	Exemple d'une couche Depthwise/Pointwise avec 2 filtres	130
45	Exemple d'une conversion de couche Full-Connected vers une structure convolutive	132
46	Architecture de l'Inception module	134
47	Architecture de l'architecture Xception: à gauche, un block standard et à droite un block simplifié (topologie uniforme)	136
48	Architecture de l'architecture Xception: à gauche, un block Inception reformulé et à droite un block Inception dans sa forme extrême	136
49	Architecture du Residual Block	137
50	Intuition sur la notion de Résidus	138
51	Architecture d'un Wide Residual block (les étapes de Batch normalization) et de ReLu ne sont pas affichées) - la largeur des rectangles représentant les couches déterminent graphiquement le nombre de filtres employés pour la couche associée	139
52	Différentes représentations de l'architecture d'un block du réseau ResNext: 1) Approche ResNet, 2) Approche Inception, 3) Approche par convolution groupée	140
53	Architecture d'un réseau Densely Connected	141
54	Réseau DenseNet	141
55	Stochastic Depth routine	142
56	Exemple d'un réseau RoR de niveau 3	143
57	Différence entre un block issu de ResNet, DenseNet et SparseNet	144
58	Construction d'un bloc du réseau FractalNet	146
59	Analyse des réseaux de Deep Learning (2018)	147
60	Architecture du Fire Module	149
61	Architecture d'un Autoencoder	154
62	Architecture d'un Denoising Autoencoder	155
63	Filtre d'un k-Sparse Autoencoder selon la valeur de k sur le jeu de données MNIST	157
64	Architecture d'un Variationnal Autoencoder	161
65	Détermination du vecteur de contexte par un Variationnal Autoencoder	161
66	Generation d'image par un Variationnal Autoencoder entraîné sur MNIST	162
67	Architecture d'un réseau antagoniste génératif	163

68	Comparaison des gradients de la fonction de perte du Générateur	165
69	Exemple simple d'un réseau siamois	168
70	Représentation d'un RNN "déroulé" allant de l'instant 0 à t	171
71	Exemple d'une cellule RNN	172
72	Influence de la donnée selon sa position dans la séquence	173
73	Architecture-type d'un réseau récurrent	175
74	Performance de la traduction selon la dimension du texte d'entrée	177
75	Schématisation de l'approche Soft Attention	178
76	Schématisation de l'approche Hard Attention	179
77	Visualisation du comportement de Soft Attention et Hard Attention	179
78	Attention et réseaux récurrents	180
79	Attention selon Bahdanau	181
80	Attention selon Luong	182
81	Extraction des vecteurs de contexte à partir d'une feature map	184
82	Illustration de l'interpolation bilinéaire	188
83	Illustration du module SNT	189
84	Illustration d'une transformation par SNT avec le dataset MNIST	189
85	Comparaison entre un module ResNet standard et un module SE-Resnet	192
86	Illustration des blocs sSE et scSE	192
87	Exemple d'analyse d'image: 1) Classification, 2) Object detection, 3) Object recognition, 4) Segmantic segmentation, 5) Instance segmentation, 6) Object proposal, 7) Text Detection, 8) landmark Detection (Pose estimation), 9) Image Captionning	196
88	Exemple - Object Detection	198
89	Exemple - Landmark Detection	198
90	Exemple - Sliding Windows	200
91	Limitation du Sliding Windows	200
92	Exemple - Convolutive Sliding Windows	201
93	Exemple - Graph-based Segmentation	203
94	Exemple - Selective Search	203
95	Exemple - Anchor Boxes	204
96	Exemple - RoI Pooling	206
97	Illustration de la métrique Intersection over Union (IoU)	207
98	Illustration de la problématique de la superposition de bounding boxes	209
99	Illustration de Non-Max Detection	209
100	Illustration de l'architecture Feature Pyramid: a) Architecture générale b) Architecture des connexions latérales	211
101	Illustration d'un réseau de prédiction de bounding boxes selon R-CNN	213
102	Illustration d'un réseau R-CNN	213
103	Illustration du Spatial Pyramid Pooling	214
104	Illustration du Spatial Pyramid Pooling Network	215
105	Analogie de RoI Pooling avec Spatial Pyramid Pooling	215
106	Illustration du Fast R-CNN	216

107	Illustration du Faster R-CNN	217
108	Comparatif de performance entre R-CNN, Fast R-CNN et Faster R-CNN	217
109	Différence de la sortie des couches convolutives entre les approches Single shot et Region based	218
110	Fonctionnement général de l'algorithme YOLOv1	223
111	Architecture des couches Full-Connected de l'algorithme YOLOv1	224
112	Architecture de l'algorithme YOLOv1	224
113	Différence de centre entre un quadrillage pair et impair	225
114	Conversion des prédictions du réseau YOLOv2 selon l'anchor associée	227
115	Architecture du réseau convolutif YOLOv2	227
116	Architecture du réseau convolutif YOLOv3	230
117	Architecture du réseau SSD	232
118	Architecture de Multibox	232
119	Architecture de Deconvolutional Single Shot Detector (DSSD) . .	234
120	Architecture du Deconvolution module de DSSD	234
121	Architecture du module prédictif de DSSD	235
122	Erreur obtenue par Focal Loss selon γ	236
123	Architecture du réseau RetinaNet	237
124	Comparatif des modèles d'Object Detection	238
125	Comparatif des modèles d'Object Detection pour un mAP à 0.5 IoU	238
126	Loi de Zipf pour K=1	240
127	Architecture de CNN for Sentence Classification	243
128	Architecture de Dynamic Convolutional Neural Network	245
129	Architecture de Multi Channel Variable size CNN	245
130	Architecture de Very Deep CNN	246
131	Architecture de LSTM Deep Sentence Embedding	248
132	Architecture de Discriminative RNN	249
133	Architecture de Hierarchical Attention Networks	250
134	Architecture de CNN-RNN	251
135	Architecture de C-LSTM	252
136	Architecture de C-LSTM avec Pooling	253
137	Architecture de AC-BLSTM	254
138	Harmonisation des dimensions des feature map selon l'approche de AC-BLSTM	255
139	Graphique séquentiel de l'approche par renforcement	258
140	Algorithme du TD(0)	263
141	Algorithme du Q-Learning(0)	266
142	Algorithme du Deep Q-Learning	269
143	Algorithme du Dueling Deep Q-Learning	272

Listings

1 Préambule

Cette introduction n'a pas vocation à introduire les concepts mathématiques associés aux réseaux de neurones de manière approfondie. Elle se limitera à expliquer les fondations théoriques d'un réseau de neurones afin de fournir une vision d'ensemble et une sensibilité permettant au lecteur, par la suite, de se former de manière autonome. Aucune démonstration ou preuve mathématique ne sera explicitée afin de faciliter la compréhension et de se concentrer l'idée sous-jacente.

Aucune connaissance mathématique approfondie n'est nécessaire pour comprendre le contenu de cette introduction. Néanmoins, des bases mathématiques telles que les fondamentaux en Analyse (fonctions utilitaires, dérivé partielle et convergence) et en Statistiques/Probabilités (estimateurs, corrélation, distribution) sont nécessaires. Il est important que le lecteur maîtrise l'écriture mathématique et ses spécificités. Le niveau demandé correspond à un niveau **Bac Scientifique / Licence**.

2 Paradigme de l'Induction

2.1 Qu'est-ce que l'Induction ?

Supposons un chat et un cheval. Ces deux entités se distinguent de part leur silhouette, leur dimension, leur vitesse de déplacement etc... Il est aisément de discriminer ces deux animaux et ce, sans erreur notable. Cependant, l'observateur n'a jamais pu observer *l'intégralité* des spécimens de chats et de chevaux. A partir d'un volume limité d'observations, il a appris à *discriminer*, i.e isoler des caractéristiques permettant de réaliser une identification. Ce comportement est appelé **induction** ou encore **généralisation**.

Remarque: Le paragraphe ci-dessus illustre un cas d'*induction* implicite. En effet, dire que "*tout le monde peut discriminer...*" est une *généralisation*. Afin d'être rigoureux, il aurait fallu indiquer l'environnement où cette affirmation se vérifie (pays, régions, aptitudes de l'observateur...). Et même dans cette situation, le phénomène de *généralisation* est encore présent. En réalité, la seule affirmation véritablement valide serait comparable à "*La quasi-totalité des individus observés respectant des conditions spécifiques sont capables de réaliser cette discrimination*". La notion de *conditions* est primordiale car elle détermine les spécificités de l'environnement. Pour illustrer, il est évident qu'un individu valide et un aveugle réalisent une discrimination de ce phénomène de manière distincte.

L'objectif de l'**induction artificielle** est de déterminer un ensemble de *règles de décision* de manière *automatique* à partir d'un échantillon limité de données illustrant le phénomène étudié. Cette ensemble de règles peut avoir deux objectifs (non exclusifs):

- **Réaliser une prédiction** sur une observation spécifique. Par exemple, "Cette image représente un chat ou un chien ?".
- **Extraire une théorie du phénomène observé**. En d'autres mots, être capable de faire une prédiction et d'expliquer le phénomène. Par exemple, "Cet animal est un chat car il chasse une souris".

Trois questions fondamentales constituent la problématique de l'induction:

- Ais-je le droit de généraliser à partir d'un échantillon fini de donné ?
- Comment réaliser l'extrapolation ? Comment savoir si comprendre les données disponibles permettra une bonne compréhension du phénomène durant la phase de prédiction ?
- Quelle garantie peut-on avoir sur l'extrapolation réalisée ?

2.2 Les différentes formes d'apprentissage

L'apprentissage peut se présenter sous différentes formes. Le type d'apprentissage dépend de la **tâche** à accomplir mais surtout du **type** et du **volume** de données disponibles. La donnée est au coeur de l'apprentissage et sa qualité est **primordiale**¹. L'hégémonie des entreprises chinoises et américaines dans le domaine de l'intelligence artificielle² repose en partie sur le volume de données qu'ils ont emmagasiné³ au fil des années. Peu importe l'efficacité d'un algorithme d'apprentissage, si la donnée d'apprentissage est mauvaise alors la qualité de la prédiction le sera aussi !

Il existe de nombreuses formes d'apprentissage. Néanmoins, plusieurs méthodes sortent du lot du fait de leur maturité et surtout, de leur efficacité.

2.2.1 Apprentissage supervisé

L'apprentissage supervisé est la méthode d'apprentissage la plus populaire actuellement. La majorité des approches de **Deep Learning** reposent sur ce type d'apprentissage bien qu'elles tendent à se diversifier de plus en plus avec les avancées de la Recherche.

L'apprentissage supervisé permet un apprentissage à partir d'exemples dits **labellisés**. Une donnée labellisée se présente sous la forme d'un couple:

$$\mathcal{X} \rightarrow \mathcal{Y} : (x_i, y_i)_{i \in \mathbb{N}}$$

\mathcal{X} désigne une distribution qui illustre un phénomène à observer et à discriminer. Par exemple, \mathcal{X} peut être associée à une distribution sur \mathbb{R}^{784} si le phénomène à observer est caractérisé par des images (plus spécifiquement, des pixels) de chats de 784 pixels. \mathcal{Y} correspond à l'espace de prédiction. Ainsi, supposons une prédiction binaire est/n'est pas, alors $\mathcal{Y} = [0, 1]$.

Un ensemble d'apprentissage, dans le cadre supervisé, est un ensemble de données labellisées. Le volume de données nécessaires est difficile à déterminer. Néanmoins, la quantité est très importante et proportionnelle à la complexité de l'algorithme d'apprentissage et de la tâche à apprendre. Le manque de données disponibles est une des raisons du retard des pays européens du fait d'une politique sur les libertés individuelles (vie privée) plus restrictive.

L'objectif de l'apprentissage supervisé est de définir une fonction f (nommé *superviseur*) telle que $f(x) = y$ pour $(x, y) \in \mathcal{X} \times \mathcal{Y}$. Il est important de considérer qu'un ensemble d'apprentissage n'illustre pas l'intégralité d'un phénomène mais uniquement une partie. La capacité de généralisation de f sur l'ensemble des cas observables (y compris ceux non représentés dans l'ensemble d'apprentissage)

¹Définir ce qu'est une donnée *qualitative* peut être ardue.

²Et plus globalement, de la Donnée

³Au détriment, parfois, de la vie privée...

définit la qualité de la prédiction de l'algorithme. Ainsi le *superviseur* apprend à isoler des *règles* (ou des caractéristiques) qui discriminent chacune des classes représentées par l'ensemble d'apprentissage et supposant qu'elles se généralisent sur l'ensemble du phénomène. Il est donc fondamental que l'ensemble d'apprentissage soit représentatif du phénomène considéré.

Dans le cadre de l'apprentissage supervisé, nous pouvons définir **trois** problèmes-types:

- **Classification:** Une *classification* est définie lorsque $\mathcal{Y} = 1, \dots, N$, i.e \mathcal{Y} est un espace discret (et fini). Un problème de classification revient donc à étiqueter une entrée inconnue avec un label. Nous pouvons citer, par exemple, la discrimination d'images selon ce qu'elles représentent. Une classification peut être *binaire*. Nous avons donc un problème de type "est/n'est pas" ($|\mathcal{Y}| = 2$). Au contraire, une classification peut être *multiclassee*. De ce fait, elle ne cherche plus à discriminer deux classes mais n classes distinctes ($|\mathcal{Y}| = N$).
- **Régression:** Une *régression* est définie lorsque $\mathcal{Y} \in \mathbb{R}$, i.e \mathcal{Y} est continue. Une régression consiste donc à prédire une valeur. L'exemple standard de régression est la prédiction de prix ou d'une valeur dimensionnelle quelconque.
- **Prédiction structurée:** \mathcal{Y} peut être associée à une *structure*, i.e une entité dite complexe qui se présente souvent sous forme de *graphes*. Ce type de prédiction est très employé dans le secteur médical, notamment pour les problématiques associées à l'étude génétique.

2.2.2 Apprentissage non supervisé

L'apprentissage *non-supervisé* est caractérisé par l'exploitation de données **non labellisées**, i.e de la forme $\mathcal{X} : x_i \in \mathbb{R}$.

Comme les données ne sont pas catégorisées, il n'est pas possible d'apprendre une fonction d'association entre les données observées d'un phénomène et un étiquetage quelconque. L'objectif est donc d'exploiter les similarités entre les données afin de pouvoir les regrouper par groupes d'entités similaires appelés *cluster*.

Ce type d'approche est très populaire car elle répond à deux problématiques industrielles très importantes:

- **Difficulté de la labellisation:** Labelliser des données peut être difficile et/ou coûteux en ressources (temporelles et financières). En s'émancipant de la labellisation, de fortes économies sont réalisées.
- **Découverte de phénomène:** L'apprentissage supervisé suppose une connaissance du phénomène à discriminer. En effet, labelliser une donnée

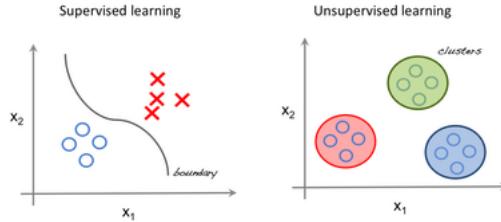


Figure 1: Différence entre apprentissage supervisé et non supervisé

nécessite de connaître ce qu'elle représente. Cette condition de *supervision* est très problématique lorsque le phénomène observé est inconnu (ou encore partiellement connu). Au contraire, l'apprentissage non supervisé, en ignorant la condition de supervision, s'émancipe de l'obligation de connaissance du phénomène. Cette approche est donc capable d'apprendre un phénomène inconnu et de ce fait, en découvrir de nouveaux et les expliciter.

Cette qualité est une plus-value au niveau industriel car elle permet de détecter les tendances et l'évolution des comportements. Elle est, par exemple, très exploitée dans le cadre de l'analyse comportementale des utilisateurs, des tendances de consommation etc...

- **Nombre de classes variable:** L'apprentissage supervisé nécessite un nombre de classes fini et constant. Au contraire, l'apprentissage non supervisé permet une prédiction indépendante d'une condition préalable sur les classes.

L'apprentissage supervisé apprend des règles durant son apprentissage qui lui permettra de discriminer une donnée inconnue. Au contraire, l'apprentissage non supervisé n'est pas capable de définir une classe par discrimination. En effet, l'apprentissage non supervisé apprend à détecter les similarités entre les données. De ce fait, il n'est pas capable de discriminer une classe car il n'a pas conscience des règles qui la constitue ni même de son existence. Cette émancipation d'une connaissance *a priori* des classes permet une plus grande souplesse d'utilisation mais aussi une impossibilité de catégorisation. L'apprentissage non supervisé repose sur l'idée que deux données similaires sont associées à une même catégorie. Ainsi, il est possible de définir des groupements de données présentant un comportement similaire mais à la labellisation inconnue. La différence entre les deux méthodes d'apprentissage est représentée sur la Figure 1.

2.2.3 Apprentissage semi-supervisé

Une approche alternative, nommée *apprentissage semi-supervisé*, propose d'exploiter des données **labellisées** et **non labellisées**. Elle est donc comparable à un

mélange entre apprentissage *supervisé* et *non supervisé*.

Cette forme d'apprentissage cherche à combiner les avantages des deux approches tout en corrigeant mutuellement leurs défauts propres. L'objectif principal de cette méthode d'apprentissage est de limiter le volume de données labellisées nécessaires à l'apprentissage supervisé en reposant sur la capacité d'association des données que procure l'apprentissage non supervisé. Ainsi, à partir d'un volume restreint de données labellisées, l'algorithme sera capable d'améliorer ses règles de discrimination à partir de données non labellisées grâce aux capacités non supervisées de l'algorithme.

Les approches standards semi-supervisées sont l'*auto-apprentissage* et le *co-apprentissage*.

L'*auto-apprentissage* cherche à apprendre un superviseur à partir d'un volume restreint de données. Le superviseur discrimine alors des données non labellisées et les prédictions à forte confiance sont ajoutées à l'ensemble d'apprentissage. Un nouveau apprentissage est alors de nouveau réalisé avec le nouveau ensemble d'apprentissage. Ce cycle peut être répété plusieurs fois.

Le *co-apprentissage* part du postulat qu'il existe 2 (ou plus) projections indépendantes d'un même espace de données. Ainsi, il va apprendre un classifieur sur chacune de ces projections et réaliser des prédictions indépendantes. Si la prédiction des classificateurs pour une même donnée est identique⁴, alors la donnée est discriminée par la classe prédite.

Par exemple, supposons que nous voulons discriminer un individu selon son sexe. La taille, le poids et la pilosité sont des critères discriminants. Ainsi, trois projections selon ces variables permettent la création de trois classificateurs qui évalueront un individu selon ces trois critères. Dans les faits, la taille et les poids ne sont pas complètement indépendants⁵. Néanmoins, on peut faire l'hypothèse qu'ils le sont.

2.2.4 Apprentissage par transfert

L'apprentissage automatique est une tâche coûteuse en ressources et en temps. De même, elle nécessite un important volume de données. Ces contraintes sont déterminantes pour le développement et l'exploitation de l'intelligence artificielle. En effet, il peut être difficile d'avoir des données associées à une tâche précise, de même qu'il n'est pas toujours possible d'avoir à disposition, une forte capacité de calculs (ou de temps).

C'est dans cet environnement que l'apprentissage par transfert (*Transfer Learning*) prend toute son importance. En effet, ce paradigme repose sur l'idée que

⁴Au moins, à la majorité...

⁵Le poids et la taille sont corrélés entre eux.

les connaissances acquises lors d'un apprentissage dans un environnement particulier peuvent être utilisées pour améliorer l'apprentissage dans un autre environnement. Ainsi, il n'est plus nécessaire de supposer un contexte strictement inconnu mais au contraire, en supposer une partie déjà définie. Cette approche permet donc un gain de temps d'apprentissage et une optimisation des données considérables, ce qui est critique pour l'industrie moderne.

Il ne s'agit pas d'une méthode d'apprentissage à proprement parler (comme l'apprentissage supervisé ou non supervisé) mais plutôt d'une optimisation de ces dernières. Elles sont donc complémentaires. Par exemple, dans le cadre de l'apprentissage supervisé, l'apprentissage par transfert implique la possibilité de réutiliser la connaissance de la structure de dépendance entre les caractéristiques et les étiquettes apprises dans un contexte afin d'améliorer l'inférence de la structure de dépendance dans un autre contexte.

Supposons un phénomène particulier (ou domaine) défini sur un espace \mathcal{X} par une distribution \mathcal{P} . Nous avons donc un domaine \mathcal{D} tel que $\mathcal{D} = (\mathcal{X}, \mathcal{P})$. Nous souhaitons réaliser une tâche \mathcal{T} tel que $\mathcal{T} = (\mathcal{Y}, f)$ avec f , fonction-cible définie telle que $f : \mathcal{X} \rightarrow \mathcal{Y}$.

Supposons un cas où nous disposons de deux domaines distincts avec une tâche définie par domaine. Le premier est le domaine-source \mathcal{D}_S et possède une tâche \mathcal{T}_S . Le second est le domaine-cible et est défini par \mathcal{D}_C et \mathcal{T}_C . L'objectif de l'apprentissage par transfert est d'améliorer l'apprentissage de $f_{\mathcal{T}_C}$ en utilisant la connaissance issue de \mathcal{T}_S , \mathcal{D}_S en plus de \mathcal{D}_C , \mathcal{T}_C avec des conditions d'inégalité entre les domaines et les tâches. Par exemple, $\mathcal{D}_S \neq \mathcal{D}_C$ et $\mathcal{T}_S \neq \mathcal{T}_C$.

Il existe trois grandes approches d'apprentissage par transfert (voir Figure 2) :

1. **Transductive:** Les deux domaines sont différents⁶ mais la tâche est la même. De plus, nous ne possédons pas de label pour les données du domaine-cible. Nous avons donc: $\mathcal{T}_S = \mathcal{T}_C$, $\mathcal{D}_S \neq \mathcal{D}_C$ et $\mathcal{D}_{C|\mathcal{Y}} = \emptyset$. Le cas particulier où $\mathcal{D}_S = \mathcal{D}_C$ est néanmoins possible et exploite d'autres méthodes pour être exploitées.
2. **Inductive:** Les deux domaines sont identiques mais les tâches diffèrent. Nous possédons les labels pour les deux domaines ou uniquement pour le domaine cible. Nous avons donc: $\mathcal{T}_S \neq \mathcal{T}_C$, $\mathcal{D}_S = \mathcal{D}_C$
3. **Non-supervisée:** Les deux domaines sont différents et les tâches diffèrent aussi. Nous ne possédons pas les labels pour les deux domaines. Nous avons donc: $\mathcal{T}_S \neq \mathcal{T}_C$, $\mathcal{D}_S = \mathcal{D}_C$ et $\mathcal{D}_{S|\mathcal{Y}} \neq \mathcal{D}_{C|\mathcal{Y}} = \emptyset$.

⁶Différents mais liés par un lien quelconque permettant le transfert !

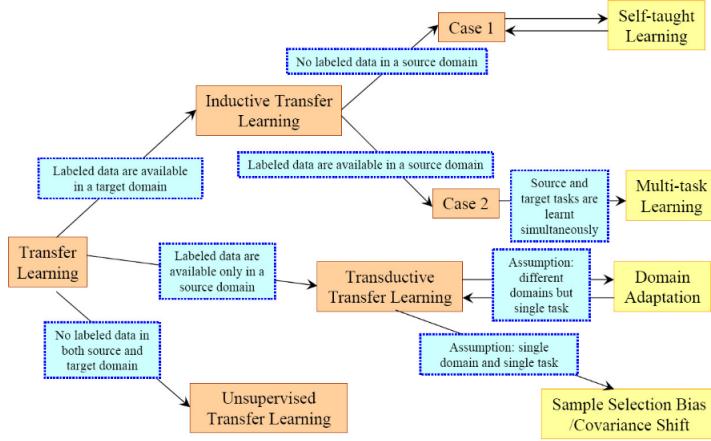


Figure 2: Les différentes approches d'apprentissage par transfert

De même, deux spécificités sont associées aux espaces des domaines. Ainsi, nous parlons d'apprentissage par transfert **homogène** si $\mathcal{X}_S = \mathcal{X}_C$. La différence se situe donc au niveau de la loi de probabilité associée à chacun des domaines. Si $\mathcal{X}_S \neq \mathcal{X}_C$, nous parlerons d'apprentissage par transfert **hétérogène**.

Quatre configurations de ces approches sont proposées: **Instance Transfer**, **Feature Representation Transfer**, **Parameter Transfer** et **Relational Knowledge Transfer**[82].

2.2.4.1 Instance Transfer

L'approche *Instance Transfer* repose sur l'idée que des données issues du domaine source peuvent être utilisées dans un autre domaine (lié) à travers différentes méthodes de *sampling* et de pondération.

2.2.4.2 Feature Representation Transfer

L'approche *Feature Representation Transfer* cherche à projeter la donnée issue du domaine-source dans un espace spécifique afin de faciliter la discrimination de f_C . Ainsi, nous définissons une fonction de projection $\phi : \mathcal{X} \rightarrow \mathcal{Z}$. De même, nous redéfinissons h_C tel que $h_C = f(\phi(x); \theta_C)$.

Cette approche est exploitée pour modifier la donnée-source afin de faciliter l'apprentissage du modèle. Par exemple, si la donnée-source possède une dimension trop élevée, il peut être utile de la représenter à travers une dimension plus faible, ce qui permet de faciliter son apprentissage. Néanmoins, il est important de considérer la perte d'informations associée à une projection dans un autre espace.

2.2.4.3 Parameter Transfer

L'approche *Parameter Transfer* est utilisée lorsque la tâche-source \mathcal{T}_S et la tâche-cible \mathcal{T}_C partagent des paramètres, i.e pour les hypothèses source et cible $h_S, h_C \in \mathcal{H}$, $h_S = f(x; \theta_S)$, $h_C = f(x; \theta_C)$, il existe des similarités entre θ_S et θ_C . Cette approche est exploitée dans le cadre du *Inductive Transfer Learning*.

Dans le cadre du Deep Learning, il s'agit, sans doute, de la méthode la plus employée. En effet, la problématique est souvent de pouvoir réexploiter un réseau ayant appris une tâche particulière pour réaliser une autre tâche à partir d'un même phénomène. Par exemple, supposons un modèle ayant appris à discriminer des images de chats. Il est judicieux de penser qu'il peut être utilisé comme base pour apprendre à discriminer des chiens. Dans ce contexte, les deux domaines sont identiques (images quelconques définies par N pixels et prises dans un **même contexte**⁷) mais les 2 tâches sont distinctes car associées à la discrimination de chats ou de chiens. Pour illustrer, nous pouvons aussi citer la discrimination de texte selon un auteur donné. Les domaines sont identiques (textes dans une langue donnée⁸) mais la tâche, différente car nous voulons discriminer des auteurs différents.

2.2.4.4 Relational Knowledge Transfer

Cette approche est l'application du Transfer Learning dans le cadre de domaines où les données ne sont pas i.i.d⁹ et dont la représentation à travers des *relations* est possible (comme les graphes, les réseaux sociaux...). *Relational Knowledge Transfer* va donc supposer que les domaines ne sont pas indépendants et i.i.d. Avec cette hypothèse, elle va donc chercher à transférer les caractéristiques relationnelles d'un domaine-source vers un domaine-cible.

2.2.4.5 Approches et configurations

Plusieurs approches et configurations de Transfer Learning ont été proposées. Néanmoins, chaque configuration ne peut être employée avec chaque approche. En effet, certaines configurations nécessitent des connaissances a-priori sur le domaine-source et/ou cible qui peuvent ne pas être satisfaites. Le tableau 3 récapitule les relations possibles ou non.

Nous pouvons observer que la configuration *Feature Representation* est réalisable pour chaque approche. En effet, cette configuration nécessite que des données (labelisées ou non) indépendamment de toute autre connaissance. Cette

⁷Cette condition est importante car deux images de même dimension peuvent ne pas appartenir au même domaine. Par exemple, une image standardisée sur fond blanc et une image prise en conditions réelles.

⁸Ceci est vrai si la langue est la même car sinon, les domaines sont distincts.

⁹Cette hypothèse sur les données est au coeur de beaucoup de preuves mathématiques du Machine Learning. Dans les faits, elle est rarement vérifiée voire même complètement fausse dans le cas de données fortement dépendantes comme les graphes.

	Inductive	Transductive	Unsupervised
Instance	✓	✓	
Feature Representation	✓	✓	✓
Parameter	✓		
Relational Knowledge	✓		

Figure 3: Configurations possibles des différentes approches de Transfer Learning

grande tolérance la rend facilement exploitable. Au contraire de *Parameter* et *Relational Knowledge* qui, à travers leurs grandes dépendances aux données-sources **et** cible, sont exploitables uniquement par l'approche la plus tolérante soit l'approche *Inductive*.

Pour approfondir les différentes problématiques du *Transfer Learning*, il est vivement recommandé de consulter l'article [82] qui propose un état de l'art accessible et complet.

2.2.5 Apprentissage par renforcement

L'apprentissage par renforcement est une méthode d'apprentissage qui se base sur la *récompense* (positive ou négative) qu'offre l'environnement d'évolution de l'automate suite à une action réalisée par ce dernier. L'automate apprend ainsi une politique de décision (ou stratégie) afin de maximiser quantitativement ses récompenses au cours du temps. Cette apprentissage est réalisée à travers une succession d'expériences itératives qui permet à l'automate de discriminer une fonction-hypothèse de comportement (la politique) basée sur l'état courant¹⁰ et l'action à réaliser.

Cette méthode d'apprentissage s'approche de l'apprentissage humain basé sur *l'essai*. L'automate n'a pas de repères ou d'exemples de réussite pour réaliser son apprentissage. Il apprend, par lui-même, à isoler et reproduire un comportement à suivre pour réaliser sa tâche. Ce type d'apprentissage s'affranchit ainsi de toute donnée d'apprentissage en dehors de la capacité d'évaluer la récompense issue de l'interaction avec l'environnement. Pour illustrer, nous pouvons prendre l'exemple d'un enfant qui apprend à marcher. Au début, il ne sait pas tenir debout. Il essaye une action "aléatoire" et tombe. La chute est la récompense (négative) de l'environnement. Il va donc réessayer en exploitant une autre approche. Au fil des essais, l'enfant sera capable de définir le bon comportement à suivre pour marcher. Il ne savait pas comment exploiter son corps pour réaliser cette tâche mais au fil des essais, il a pu discriminer les actions permettant de réaliser cette tâche. L'apprentissage par renforcement apprend ainsi à déterminer une méthode de décision uniquement basée sur son interac-

¹⁰L'état courant correspond aux spécificités de l'automate et de l'environnement observé à l'instant considéré

tion avec l'environnement, ce qui l'émancipe de tout entité extérieure (oracle) pour orienter son apprentissage.

Ce type d'apprentissage est encore peu exploité par l'industrie mais constitue un milieu très actif de la Recherche. Les bases théoriques de ce type d'apprentissage sont approfondies dans la section 14.

2.3 Formalisation d'un problème d'Induction

Un problème d'induction est défini par plusieurs composantes:

1. **L'environnement:** L'environnement permet de générer des entités x_i obtenues indépendamment et de manière identiquement distribuées (échantillon i.i.d) selon une distribution D_x sur un espace \mathbb{X} . On supposera l'environnement **stationnaire** pour des raisons de simplification. Néanmoins, il est possible de considérer l'environnement comme évolutif au cours du temps. Cette particularité est souvent considérée dans le cadre de l'*apprentissage en ligne*.
2. **L'oracle:** L'oracle retourne une réponse à une entité donné (souvent un label dans le cas d'une approche supervisée) selon une distribution de probabilité $F(u_i|x_i)$ inconnues avec $u_i \in \mathbb{U}$, ensemble des réponses possibles de l'oracle.
3. **L'apprenant:** L'apprenant applique une fonction (probabiliste ou déterministe) appartenant à un espace de fonctions \mathbb{H} . La sortie de l'apprenant est ainsi $y_i = h(x_i)$ avec $h \in \mathbb{H}$.

La **tâche d'apprentissage** consiste à permettre à l'apprenant de trouver une fonction h , nommée **fonction hypothèse**, dans l'espace \mathbb{H} (espaces des fonctions hypothèse possibles) permettant d'approximer (au mieux) la réponse exprimée par l'oracle. Dans le cadre de l'induction, on calcule la proximité entre la fonction hypothèse et la réponse de l'oracle à travers l'*espérance de perte* sur l'ensemble des cas possibles, i.e $\mathbb{X} \times \mathbb{U}$.

Pour une entité x_i et la réponse u_i de l'oracle, on définit une **perte** $l(u_i, h(x_i))$ qui évalue le coût d'avoir prédit $h(x_i)$ lorsque la réponse était u_i . On définit alors le **risque réel** par:

$$R_{real}(f) = \int_{\mathbb{X} \times \mathbb{U}} l(u_i, h(x_i)) dF(x, u)$$

Il s'agit d'une mesure statistique sur l'ensemble des événements réalisables où $F(x,u)$ peut être une distribution de probabilité ou déterministe. La densité de probabilité de $F(x,u)$ est inconnue. Il s'agit donc de trouver la fonction hypothèse h la plus proche de F selon la fonction de perte considérée et ce, pour les régions de l'espace \mathbb{X} les plus fréquentes. On ne possède pas d'information

*a priori*¹¹ sur ces régions. L'ensemble d'apprentissage est donc utilisé pour les déterminer (en mal ou en bien !). L'objectif est donc de chercher à **minimiser le risque réel inconnu** à partir d'un échantillon d'apprentissage \mathbb{S} avec $\mathbb{S} \in \mathbb{X}$.

Le **principe inductif** permet d'expliciter ce que doit vérifier la fonction h recherchée. Pour cela, il définit la fonction de proximité décrite précédemment et se base sur l'échantillon d'apprentissage $\mathbb{S} = (x_1, u_1), \dots, (x_n, u_n)$ afin de minimiser le risque réel.

En d'autres mots, le principe inductif formalise ce que doit vérifier la fonction hypothèse selon la fonction de perte, l'échantillon d'apprentissage et possiblement d'autres paramètres selon le principe choisi. Il s'agit d'un idéal théorique à ne pas confondre avec la méthode d'apprentissage qui correspond à une application effective du principe inductif (via un algorithme). En effet, pour un principe donné, plusieurs méthodes d'apprentissage peuvent exister. Les contraintes computationnelles ou algorithmiques sont indépendantes du principe inductif. Par exemple, dans le cadre d'une optimisation par une méthode de gradient, la méthode d'apprentissage est associée à la méthode exploitée i.e la méthode de gradient et l'optimum souhaité, l'objectif défini par le principe inductif mais cet optimum peut être obtenu par une autre approche que la méthode de gradient.

2.4 Quelques principes d'induction

Le principe inductif permet de définir les critères qu'une hypothèse doit respecter afin de déceler celle qui optimise la minimisation du risque réel à partir d'un ensemble fini d'exemples appelé ensemble d'apprentissage.

Il existe plusieurs principes inductifs et aujourd'hui encore, il s'agit d'un champ de recherche théorique important car la théorie actuelle est jugée imparfaite par le milieu scientifique¹².

Deux principes inductifs sont très fréquents en Machine Learning: le principe ERM et le principe de décision bayésienne. Il en existe de nombreux autres mais pour limiter d'alourdir inutilement cette partie théorique, nous les négligerons.

2.4.1 Le principe ERM

Le principe ERM (*Empirical Risk Minimization*) repose sur l'hypothèse qu'il faut minimiser le **risque empirique**. Le risque empirique correspond à la perte moyenne (ou encore l'erreur) mesurée sur l'échantillon d'apprentissage \mathbb{S} . Il est défini par:

$$R_{emp}(h) = \frac{1}{n} \sum_{i=1}^n l(u_i, h(x_i))$$

¹¹Se remémorer les bases de la théorie bayésienne !

¹²D'où la grande attention et la rigueur à avoir lors de l'utilisation de l'IA dans sa globalité.

Le principe ERM repose sur une hypothèse forte. Il suppose que la fonction hypothèse qui s'accorde le mieux aux données d'apprentissage est une fonction capable de correctement décrire le phénomène général observé. Elle repose sur l'axiome qu'une caractéristique observée sur les données connues est aussi vérifiées par les autres données générées par le phénomène considéré. Ainsi:

$$h_{ERM} = \underset{h \in \mathcal{H}}{\operatorname{ArgMin}} R_{emp}(h)$$

Ce critère inductif est le plus populaire et exploité par l'Intelligence Artificielle moderne, notamment par le Deep Learning. Le Deep Learning cherche ainsi une fonction hypothèse qui tend à avoir un risque empirique nul¹³.

2.4.2 Le principe de décision bayésienne

Le principe de décision bayésienne repose sur une approche probabiliste, i.e choisir l'hypothèse la plus probable selon le jeu de données d'apprentissage. Comme son nom l'indique, elle exploite l'approche bayésienne.

L'hypothèse faite est qu'il est possible de définir une distribution de probabilité sur l'espace des fonctions hypothèses \mathbb{H} . La connaissance préalable du phénomène peut s'exprimer sous la forme d'une distribution *a priori* de probabilités. Le jeu d'apprentissage influencera alors cette distribution initiale. Il est possible de choisir l'hypothèse la plus probable ou de définir une hypothèse composite définie par la moyenne de l'ensemble des hypothèses pondérées par leur probabilité *a posteriori*¹⁴.

Le choix de l'hypothèse la plus probable repose sur deux principes: *Maximum a posteriori* (MAP) et *Maximum de Vraisemblance*.

Si chaque erreur a une pondération équivalente, alors, la règle de décision bayésienne de risque minimal devient similaire à MAP:

$$h^* = \underset{h \in F}{\operatorname{ArgMax}} p_{\mathcal{F}}(f)p_{\mathcal{X}}(x|f)$$

$p_{\mathcal{F}}(f)$ dénote la probabilité *a priori* que le monde soit dans l'état f , \mathcal{F} , ensemble des états possibles.

Si chaque hypothèse ont la même probabilité *a priori*, alors MAP est équivalent au Maximum de Vraisemblance (Maximum Likelihood) et devient:

$$h^* = \underset{h \in F}{\operatorname{ArgMax}} p_{\mathcal{X}}(x|f)$$

¹³Dans les faits, atteindre cette valeur est souvent synonyme de mauvais apprentissage du fait du phénomène appelé *sur-apprentissage* !

¹⁴Il s'agit de la vraie approche bayésienne mais elle peut être difficile à déterminer.

Pour rappel, la relation liant probabilité *a priori* et *a posteriori* est définie par:

$$P_{a \text{ posteriori}} = \frac{P_{\text{vraisemblance}} \times P_{a \text{ priori}}}{P_{\text{evidence}}} \leftrightarrow P(y|x) = \frac{P(x|y) \times P(y)}{P(x)}$$

La connaissance initiale du phénomène permet de définir $P(y)$. Les données du jeu d'apprentissage modifie le comportement *a priori* à travers $P(x|y)$. Ainsi, supposons un phénomène y rare. Il aura une probabilité *a priori* faible. Cependant, si ce phénomène est très présent dans le jeu d'apprentissage alors $P(y|x)$ sera bien plus importante d'où la nécessité de s'assurer que le jeu d'apprentissage soit représentatif du phénomène considéré. $P(y)$ peut être initialisée de manière *neutre*. Dans ce cas, seule $P(x|y)$ sera considérée. Cette approche est utilisée lorsque aucune connaissance du phénomène est possible.

Dans les faits, $P(x)$ peut être difficile à évaluer. Cependant, cette valeur est constante et ne présente pas d'intérêt car il s'agit d'un facteur constant récurrent. Il peut donc être ignoré. Il se comporte comme un facteur de normalisation, ce qui est peu utile en comparaison de la complexité de son calcul.

2.5 Evaluation de l'apprentissage

Dans la partie précédente, nous avons proposé deux approches pour *apprendre*. Il est, cependant, fondamental de pouvoir analyser l'apprentissage et de pouvoir déterminer son comportement intrasèque. Pour cela, il est nécessaire d'exploiter des hypothèses supplémentaires liées à ce qu'on cherche évaluer du système d'apprentissage.

Un problème d'apprentissage est dépendant de l'environnement, de l'oracle et de la fonction de perte choisie. Lorsqu'on évalue la performance espérée d'un apprenant, il est nécessaire de déterminer le cadre liant l'environnement et l'oracle.

Trois cas sont possibles:

1. **L'analyse dans le pire des cas:** Cette approche suppose qu'on ne connaît rien *a priori* sur l'environnement ni sur son comportement avec l'oracle. On cherche donc à quantifier la performance de l'apprenant dans la pire des situations. Cette approche est indépendante de l'oracle et de l'environnement (et même de la fonction-cible). Il s'agit donc d'un cadre d'étude très généraliste sans hypothèse préalable forte sur les constituantes du phénomène et de l'apprentissage. Néanmoins, les conditions identifiées seront très fortes et souvent loin de la réalité expérimentale et applicative.
2. **L'analyse en cas moyen:** Cette approche cherche à mesurer une espérance de performance. Néanmoins, il est nécessaire de faire une hypothèse sur la distribution $D_{\mathbb{X}}$ (environnement) et $D_{\mathbb{F}}$ (fonction-cible possible). Théoriquement, cette méthode permet une mesure de performance plus fine que l'approche précédente mais dans les faits, il est difficile de

l'exploiter véritablement à cause de la difficulté à estimer les probabilités *a priori* et à l'obligation d'utiliser des méthodes d'approximation qui nuisent à la précision du résultat.

3. **L'analyse dans le meilleur des cas:** Cette approche est caractérisée par l'analyse de l'apprentissage dans le meilleur des cas, i.e lorsque l'oracle et l'environnement favorisent l'apprentissage en aidant l'apprenant. Cependant, la notion d'aide est difficile à évaluer, notamment la différence entre la bienveillance (professeur) et la collusion (l'oracle devient un complice). Ce type d'analyse est aujourd'hui, peu exploitée et peu établi théoriquement parlant.

2.6 Théorie de l'apprentissage

L'objectif de la *Théorie de l'apprentissage* est de développer un socle théorique permettant le développement et l'analyse d'algorithmes dit d'apprentissage. Elle cherche à démontrer la faisabilité de l'apprentissage d'un *concept* (aussi appelé *règle de prédiction*), le volume de données nécessaire à son apprentissage et les garanties théoriques associées (borner l'erreur et la complexité). Dans un cas plus général, elle étudie aussi l'efficacité (ou non) d'algorithmes selon des conditions spécifiques.

Important: Bien que plusieurs cadres théoriques aient déjà été proposés, aucun ne fait consensus auprès du milieu scientifique. Il s'agit donc d'un domaine de recherche très actif et crucial étant donné l'engouement pour l'intelligence artificielle.

2.6.1 PAC Learning (Probably Approximately Correct)

L'une des théories principales est l'**Analyse PAC** (PAC Learning - *Probably Approximately Correct*)[112].

L'hypothèse de PAC est de considérer que l'apprentissage d'un concept inconnu est *réalisable* si il est possible d'obtenir une *hypothèse* capable de l'**approximer** (Approximately) et ce, avec une **forte probabilité** (Probably).

2.6.1.1 Qu'est-ce qu'un problème d'apprentissage ?

Supposons:

- $\mathcal{C} : \mathcal{X} \rightarrow \mathcal{Y}$, une classe de concepts. On suppose \mathcal{C} comme connu et \mathcal{Y} , espace discret.
- $h \in \mathcal{C}$, le concept cible à apprendre.
- \mathcal{D} , distribution sur \mathcal{X} .

- $S = \{(x_i, h(x_i))\}_{i=1}^m$, un ensemble d'apprentissage de dimension m tel que $x_i \in \mathcal{X}$ en accord avec la distribution \mathcal{D} et i.i.d. On a $h(x_i) \in \mathcal{Y}$.
- \mathcal{A} , un algorithme d'apprentissage.
- \mathcal{H} , espace d'hypothèses apprenables par \mathcal{A} . On supposera $\mathcal{H} = \mathcal{C}$ dans un premier temps

Après l'observation de S , \mathcal{A} choisit une hypothèse $\hat{h} \in \mathcal{H}$ par discrimination. Un problème d'apprentissage est défini par la recherche d'une hypothèse \hat{h} avec une faible **erreur de généralisation** définie par:

$$P_{x \sim D}(\hat{h}_S(x) \neq h(x))$$

La relation entre l'erreur de généralisation et m^{15} , nombre de données d'apprentissage, définit la *difficulté* d'apprendre le concept h . Plus le concept est difficile, plus m sera grand avant que \hat{h}_S ait une erreur de généralisation faible.

ATTENTION:

Une interprétation naïve de ce problème serait de définir qu'une classe de concept \mathcal{C} est *apprenable* selon la définition:

- \mathcal{C} est apprenable si, pour tout $h \in \mathcal{C}$, $P_{x \sim D}(\hat{h}_S(x) \neq h(x)) \leq \epsilon$ avec $m = |\mathcal{S}|$ polynomial par $1/\epsilon$.

Cette définition définit une classe de concept comme apprenable si il est possible de borner l'erreur de généralisation avec un nombre *raisonnable* de données d'apprentissage.

Bien que séduisante, cette définition *déterministe* est incomplète car S est une **variable aléatoire**¹⁶. De ce fait, il est nécessaire de considérer le cas où S n'est pas représentatif du concept observé.

Illustrons cette problématique. Supposons le jeu Pierre-Feuille-Ciseau non biaisé. Étant donné que ce jeu est un jeu de hasard et qu'il n'est pas biaisé, le concept-cible est donc associé à une fonction aléatoire qui sélectionne un des trois choix selon une probabilité $1/3$.

Supposons que pour apprendre, notre algorithme possède des exemples de parties où x_i est toujours "Feuille". De ce fait, d'après les exemples d'apprentissage, l'algorithme maximise les gains en proposant, *à chaque fois*, le signe "Ciseau" (Le "Ciseau" gagne sur la "Feuille"). Or, ce choix d'hypothèse est mauvais vis-à-vis de l'erreur de généralisation. Bien que très peu probable, il est nécessaire de considérer la possibilité d'un ensemble S non représentatif issu de \mathcal{D} . La

¹⁵La valeur de m est liée à l'erreur de généralisation car S est paramétré par m .

¹⁶Car dépendant de x_i issu de \mathcal{D}

notion d'apprentissage PAC (**Probably Approximately Correct**) est donc proposée pour compléter cette définition.

2.6.1.2 Formalisation du PAC Learning

Considérons les notations de la section précédente. Afin de formaliser le PAC Learning, il est nécessaire de définir la notion de *consistence* et de *sample complexity* (les termes anglophones sont conservés car les traductions françaises portent à confusion du fait de leurs multiples sens).

Supposons un apprenant qui produit une hypothèse $\hat{h}_S \in \mathcal{H}$ à partir d'un ensemble d'apprentissage de taille m . Un algorithme est dit *consistent* si, pour tout ϵ et δ , il existe un nombre fini de données d'apprentissage m pour toute distribution \mathcal{D} tel que:

$$P(P_{x \sim D}(\hat{h}_S(x) \neq h(x)) > \epsilon) < \delta$$

Or $P_{x \sim D}(\hat{h}_S(x) \neq h(x)) = |R_{reel}(\hat{h}_S) - R_{reel}(h)|$ d'où:

$$P(|R_{reel}(\hat{h}_S) - R_{reel}(h)| > \epsilon) < \delta$$

En d'autres mots, un apprenant est *consistent* si:

$$R_{reel}(\hat{h}_S) \xrightarrow[m \rightarrow \infty]{} R_{reel}(h)$$

$$R_{emp}(\hat{h}_S) \xrightarrow[m \rightarrow \infty]{} R_{emp}(h)$$

La *sample complexity* est la valeur minimum de m pour qu'un algorithme *consistent* respecte la condition de *consistence*. Si m est fini alors \mathcal{H} est dit *apprenable*. Si N est une fonction polynomiale en $1/\epsilon$ et $1/\delta$, alors \mathcal{H} est **PAC-apprenable**.

THÉORÈME: PAC Learning

Soit $\mathcal{X}, \mathcal{Y}, \mathcal{C}, \mathcal{D}$ et \mathcal{S} . On définit $h \in \mathcal{C}$ alors un algorithme \mathcal{A} est un **PAC-apprenant** pour le concept h si, sous la distribution \mathcal{D} et pour tout $\epsilon, \delta \in (0, 1/2)$, \mathcal{A} s'exécute en temps polynomial de paramètres $(1/\epsilon, 1/\delta)$ et demande un volume de données d'apprentissage polynomial de paramètres $(1/\epsilon, 1/\delta)$ pour discriminer une hypothèse \hat{h}_S dans \mathcal{H} tel que:

$$P(|R_{reel}(\hat{h}_S) - R_{reel}(h)| \leq \epsilon) \geq 1 - \delta$$

Le cadre théorique du PAC Learning suppose des hypothèses fortes parfois *irréelles*. En effet, la condition que \mathcal{S} est un ensemble i.i.d issu de \mathcal{D} peut être non atteignable. La condition i.i.d sous-entend que chaque donnée d'apprentissage est décorrélées d'une autre. Or, supposons un ensemble d'apprentissage reposant sur des extraits d'une vidéo à différents instants. Il est évident que

chaque donnée extraite sera corrélée avec les précédentes et suivantes. La condition i.i.d est une hypothèse théorique au coeur du Machine Learning actuel. Bien que souvent non respectée, de nombreuses approches existent pour limiter la corrélation interne des données très préjudiciables à une bonne généralisation.

De même, pour qu'un algorithme soit PAC-apprenant, il est nécessaire qu'il soit *efficace* sur l'ensemble des distribution \mathcal{D} possibles. Ainsi, les démonstrations d'apprenant non-PAC sont souvent restreintes à la création d'une distribution peu favorable à l'algorithme étudié. Cette distribution peut être fortement improbable dans un contexte réel et de ce fait, cette condition est souvent considérée comme trop stricte. C'est pourquoi PAC Learning est une analyse dite **dans le pire des cas** car elle évalue l'algorithme selon son pire comportement possible.

2.6.1.3 PAC Learning et espace d'hypothèses

Dans la forme initiale du PAC Learning dite *correct*, $\mathcal{H} = \mathcal{C}$. Sa forme *incorrecte* assouplit cette condition en proposant $\mathcal{H} \neq \mathcal{C}$. L'ensemble des autres conditions sont respectées.

Cette caractéristique est utile dans le cas où un apprenant n'est pas capable d'approximer des hypothèses sur \mathcal{C} .

2.6.1.4 Agnostic PAC Learning

Dans la version initiale du PAC Learning, nous supposons qu'il existe un concept-cible h tel que $h \in \mathcal{C}$ (et $\mathcal{C} = \mathcal{H}$). Il existe donc une hypothèse-cible qui labellise parfaitement les observations.

Admettons maintenant que \mathcal{D} est une distribution sur $\mathcal{X} \times \mathcal{Y}$. Ainsi, \mathcal{D} devient aussi une variable aléatoire selon \mathcal{Y} . De ce fait, les données d'apprentissage utilisés ne sont plus de la forme $(x_i, h(x_i))$ mais (x_i, y_i) . Il est donc possible que pour un x_i donné, plusieurs y_i soient possibles. En d'autres mots, l'oracle n'est plus déterministe mais probabiliste.

Bien que cela puisse sembler contre-intuitif, cette situation est la plus réelle. En effet, cette condition est réalisée en présence de bruits sur les données d'apprentissage, d'erreur de labellisation de l'oracle (qui peut ne pas être infaillible dans la réalité) ou encore si $h \notin \mathcal{C}$ ¹⁷ par exemple.

Cette situation est dit *non-réalisable* car il n'existe pas de concept-cible sur \mathcal{H} capable de parfaitement labelliser les données (opposée à la situation *réalisable* du PAC Learning présenté initialement). L'apprenant \mathcal{A} ne doit donc pas trouver une hypothèse *idéale* mais une hypothèse qui minimise au mieux les erreurs.

¹⁷Une distribution sur $\mathcal{X} \times \mathcal{Y}$ ne peut être approximée par aucune fonction sur $\mathcal{X} \rightarrow \mathcal{Y}$.

Ce cadre théorique s'appelle **Agnostic PAC Learning**[75].

A partir de ce postulat, nous pouvons mettre à jour la notion de risque réel qui de:

$$R(\hat{h}_S) = P_{x \sim \mathcal{D}}[\hat{h}_S(x) \neq h(x)]$$

Devient:

$$R(\hat{h}_S) = P_{(x,y) \sim \mathcal{D}}[\hat{h}_S(x) \neq y]$$

Cette écriture est une généralisation de la notion de risque réel. En effet, il est possible de définir que \mathcal{Y} est une distribution déterministe qui, pour $x \in \mathcal{X}$, est égal à $h(x)$ (il s'agit donc, dans cette situation, d'une distribution conditionnée par \mathcal{X}).

Dans le cadre *agnostic*, le concept-cible peut ne pas être approximée par une fonction hypothèse sur \mathcal{H} . Il n'est alors possible que de discriminer une fonction hypothèse sur \mathcal{H} aussi performante que la meilleure hypothèse sur \mathcal{H} .

Nous définissons la qualité de l'approximation optimale de \mathcal{D} par \mathcal{H} selon:

$$E(\mathcal{D}, \mathcal{C}) := \inf_{h \in \mathcal{H}} P_{(x,y) \sim \mathcal{D}}[\hat{h}_S(x) \neq y]$$

Une nouvelle contrainte apparaît. Dans le contexte *no-agnostic*, il n'existe plus nécessairement d'hypothèse $\hat{h} \in \mathcal{H}$ capable de parfaitement approximer les prédictions de l'oracle sur les données d'apprentissage. Il est donc nécessaire d'assouplir la condition de discrimination d'une hypothèse en considérant que l'hypothèse doit être la plus performante possible. Pour cela, nous utiliserons la notion de risque empirique tel que pour $S = \{(x_i, y_i)\}_{i=1}^m$:

$$R_{emp}(\hat{h}_S) = \frac{1}{m} |\{i : \hat{h}_S(x_i) \neq y_i\}|$$

THÉORÈME: Loi forte des grands nombres

Soit $(X_n)_{n \in \mathbb{N}}$, suite de variables aléatoires indépendantes d'une même loi de probabilité et $Y_n = \frac{1}{n} \sum_{i=1}^n X_i$, nous avons:

$$P\left(\lim_{n \rightarrow \infty} Y_n = E(X)\right) = 1$$

En d'autres termes, la moyenne d'une variable aléatoire (risque empirique) converge vers son espérance (risque réel) lorsque le nombre d'observations augmente.

Mais comment définir la condition du PAC Learning si il n'existe pas d'hypothèse parfaite ?

Supposons un algorithme \mathcal{A} qui, pour un ensemble d'apprentissage \mathcal{S} issu de $\mathcal{D}_{(\mathcal{X}, \mathcal{Y})}$ propose une hypothèse $\hat{h}_{\mathcal{S}} \in \mathcal{H}$ avec une erreur empirique minimale. Si il existe plusieurs hypothèses qui respectent cette condition, \mathcal{A} en sélectionne une arbitrairement. Nous voulons donc, pour reprendre l'idée du PAC Learning, déterminer une borne inférieure sur m qui, pour tout $\epsilon, \delta \in (0, 1/2)$, vérifie:

$$P \left(R_{reel}(\hat{h}_{\mathcal{S}}) - \min_{h \in \mathcal{H}} R_{reel}(h) > \epsilon \right) < \delta$$

Supposons la condition suivante vraie:

$$\forall h \in \mathcal{H}, |R_{reel}(h) - R_{emp}(h)| \leq \frac{\epsilon}{2}$$

Alors, pour tout $h \in \mathcal{H}$:

$$\begin{aligned} R_{reel}(\hat{h}_{\mathcal{S}}) &\leq R_{emp}(\hat{h}_{\mathcal{S}}) + \frac{\epsilon}{2} \\ &\leq R_{emp}(h) + \frac{\epsilon}{2} \\ &\leq (R_{reel}(h) + \frac{\epsilon}{2}) + \frac{\epsilon}{2} = R_{reel}(h) + \epsilon \end{aligned}$$

Les inégalités précédentes sont pour tout $h \in \mathcal{H}$ et vraies d'après la *Loi forte des grands nombres*. Or, nous avons une condition de minimisation de l'erreur empirique pour le choix de \hat{h} . La condition est donc remplie si:

$$R_{reel}(\hat{h}_{\mathcal{S}}) \leq \min_{h \in \mathcal{H}} R_{reel}(h) + \epsilon$$

THÉORÈME: Agnostic PAC Learning

Soit \mathcal{X} , \mathcal{Y} , \mathcal{C} , \mathcal{D} et \mathcal{S} . On définit $h \in \mathcal{C}$ et meilleur approximation du concept-cible $c \notin \mathcal{C}$ alors un algorithme \mathcal{A} est un **agnostic PAC-apprenant** pour le concept c si, sous la distribution \mathcal{D} et pour tout $\epsilon, \delta \in (0, 1/2)$, \mathcal{A} s'exécute en temps polynomial de paramètres $(1/\epsilon, 1/\delta)$ et demande un volume de données d'apprentissage polynomial de paramètres $(1/\epsilon, 1/\delta)$ pour discriminer une hypothèse $\hat{h}_{\mathcal{S}}$ dans \mathcal{H} tel que:

$$P(R_{reel}(\hat{h}_{\mathcal{S}}) \leq \min_{h \in \mathcal{H}} R_{reel}(h) + \epsilon) \geq 1 - \delta$$

Une variante dite *incorrecte* de *agnostic PAC Learning* est possible en autorisant la condition $\mathcal{H} \neq \mathcal{C}$.

De même, une variante relaxée de degrés α nommée α -*agnostic PAC Learning* permet d'assouplir la condition de performance en modifiant la condition telle que:

$$R_{reel}(\hat{h}_{\mathcal{S}}) \leq \alpha E(\mathcal{D}, \mathcal{C}) + \epsilon$$

2.6.1.5 Généralisation aux espaces de concepts continus

PAC Learning est une théorie générale qui s'applique arbitrairement sur tout \mathcal{X} et \mathcal{Y} . Néanmoins, supposons un espace de concepts \mathcal{C} tel que $\mathcal{C} : \mathbb{R}^n \rightarrow \mathbb{R}$. Il s'agit donc d'un concept d'un espace discret à un espace continu.

Dans cette configuration, pour $h \in \mathcal{C}$, la condition de calcul de l'erreur réelle définie par:

$$R_{reel}(\hat{h}_S) = P_{(x,y) \sim \mathcal{D}}[\hat{h}_S(x) \neq y]$$

n'est pas efficace car trop stricte¹⁸.

Une généralisation du risque réel est donc nécessaire. Pour cela, nous proposons:

$$R_{reel,gen}(\hat{h}_S) = \mathbb{E}_{(x,y) \sim \mathcal{D}}[L(\hat{h}(x), y)]$$

L est nommée **fonction de perte**. Elle est caractérisée par $L : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$. Son objectif est de mesurer la similarité entre la prédiction de l'apprenant \mathcal{A} et la prédiction de l'oracle. Dans le cas d'un espace de concepts défini par un espace continu, il est commun d'exploiter la notion de *distance*. Nous pouvons présenter L_1 *Loss* définie par $L(y_1, y_2) = \|y_1 - y_2\|$ ou encore L_2 *Loss* par $L(y_1, y_2) = \|y_1 - y_2\|_2^2$.

Ainsi, R_{reel} est un cas particulier de $R_{reel,gen}$ tel que:

$$R_{reel}(\hat{h}_S) = \mathbb{E}_{(x,y) \sim \mathcal{D}}[L(\hat{h}_S(x), y)]$$

$$L(x, y) = \begin{cases} 1 & \text{si } x \neq y \\ 0 & \text{sinon} \end{cases}$$

2.6.2 Exemple d'une étude par PAC Learning - Le principe ERM dans le cas fini

Supposons que $|\mathcal{H}|$ est fini, i.e il existe un nombre fini d'hypothèses possibles et que la tâche est *réalisable*. Ainsi il existe $\hat{h} \in \mathcal{H}$ telle que $R_{emp}(\hat{h}) = 0$. De même, nous utiliserons la fonction de perte L tel que:

$$L(x, y) = \begin{cases} 1 & \text{si } x \neq y \\ 0 & \text{sinon} \end{cases}$$

Avec cette fonction de perte, le risque réel est égale à la probabilité qu'un exemple se situe dans la zone d'erreur entre l'hypothèse-cible et l'hypothèse apprise. En d'autres mots, il s'agit de la différence symétriques entre les deux ensembles que nous noterons $\hat{h} \Delta h$. Une illustration de cette zone est présentée sur la Figure 4.

¹⁸Les conditions d'égalité sont souvent exploitées dans le cadre de problème de classification (\mathcal{Y} discret) mais pas pour un problème de régression (\mathcal{Y} continu).

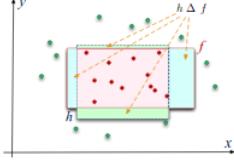


Figure 4: Erreur entre l'hypothèse-cible et l'hypothèse apprise

Soit \hat{h} tel que $R_{reel}(\hat{h}_S) > \epsilon$. La probabilité qu'après une observation $x \in D$, l'hypothèse apprise ne fasse pas d'erreur est donc inférieure ou égale à $1 - \epsilon$.

$$P_D(\hat{h}) = P_{(x,y) \sim D}(\hat{h}(x) = y)$$

Supposons m observations:

$$\begin{aligned} P_D^m(\hat{h}) &= (P_{(x,y) \sim D}(\hat{h}(x) = y))^m \\ &\leq (1 - \epsilon)^m \end{aligned}$$

Or, $1 + x \leq e^x$:

$$P_D^m(\hat{h}) \leq e^{-\epsilon m}$$

Ainsi, pour une hypothèse choisie, la probabilité de "survie" est de $e^{-\epsilon m}$. Or, il est possible qu'il existe plusieurs hypothèses respectant le critère ERM mais ayant un risque réel supérieur à ϵ . Ainsi, en considérant l'ensemble des hypothèses possibles \mathcal{H}_ϵ , la probabilité qu'une hypothèse "mauvaise" soit choisie est bornée par $\sum_{h \in \mathcal{H}_\epsilon} P_D^m(\hat{h})$

$$\begin{aligned} P_D^m(\forall h \in \mathcal{H}_\epsilon \hat{h}) &= \sum_{h \in \mathcal{H}_\epsilon} P_D^m(\hat{h}) \\ &\leq |\mathcal{H}_\epsilon| e^{-\epsilon m} \\ &\leq |\mathcal{H}| e^{-\epsilon m} \end{aligned}$$

Le PAC-Learning est caractérisé par:

$$P(P_{x \sim D}(R_{reel}(\hat{h}_S)) \leq \epsilon) > 1 - \delta$$

La condition est donc respectée si:

$$m \geq \frac{1}{\epsilon} \ln \frac{|\mathcal{H}|}{\delta}$$

Important: En réécrivant l'équation du PAC-Learning dans ce cas-ci, nous observons:

$$P(P_{x \sim D}(R_{reel}(\hat{h}_S)) \leq \epsilon) > 1 - \delta$$

$$P(P_{x \sim D}(R_{reel}(\hat{h}_S)) \leq R_{emp}(\hat{h}) + \frac{\log|\mathcal{H}| + \log\frac{1}{\delta}}{m}) > 1 - \delta$$

$R_{emp}(\hat{h}) = 0$ par définition car nous sommes dans le cas réalisable. Nous observons donc que m est directement dépendant de $|\mathcal{H}|$ mais surtout, la borne d'erreur est directement dépendante de la cardinalité de \mathcal{H} . De ce fait, il est **fondamentale** de limiter l'espace des hypothèses possibles afin de favoriser l'induction. En d'autres mots, il est nécessaire de *contraindre* l'espace des hypothèses avec des conditions supplémentaires. L'application de contraintes s'appelle la **Régularisation**. La **Régularisation** est très employée en Machine Learning et le Deep Learning ne fait pas exception. Elle est employée pour favoriser l'apprentissage et lutter contre le risque de sur-apprentissage.

Remarque: Cette étude de cas est relativement triviale notamment du fait qu'on a étudié le cas fini avec un espace d'hypothèse fini. Dans les cas plus généraux, l'étude de modèle est une tâche ardue et complexe qui nécessite une connaissance mathématique poussée.

2.7 Théorie de la régularisation

2.7.1 Contexte et environnement

L'induction se présente comme un problème dit *mal posé*. Un problème *bien posé* doit répondre à 3 critères:

- **Existence:** pour toute fonction-cible h , il existe une fonction $\hat{h} \in \mathcal{H}$ en tant que solution du problème.
- **Unicité:** \hat{h} est unique.
- **Continuité:** \hat{h} dépend *continûment* de h .

Or, dans le cadre de l'induction, la condition d'unicité est souvent non respectée. De même qu'il est possible qu'il n'existe pas d'hypothèse dans \mathcal{H} qui puisse parfaitement s'associer aux données, notamment si les données sont bruitées ou non représentables par l'espace \mathcal{H} .

La théorie de l'apprentissage modifie le problème *mal posé* que représente le problème d'induction en ajoutant des contraintes additionnelles. Ainsi, le problème à résoudre est la recherche de \mathcal{H} respectant la condition du critère inductif *et* une contrainte $\Phi(\mathcal{H}) \leq \alpha$. Cette contrainte représente une caractéristique connue a priori sur la solution optimale recherchée. Elle permet de compléter la discrimination des hypothèses valides et ainsi, de diminuer le nombre d'hypothèses recevables par le modèle, i.e la cardinalité de \mathcal{H} .

Intuitivement, nous voulons supprimer les classes d'hypothèses *trop* complexes, i.e capable de s'accorder à tout échantillon de dimension m . Ce type d'hypothèses

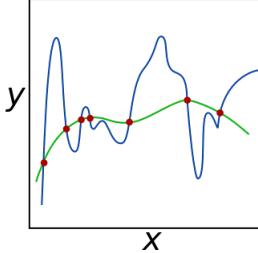


Figure 5: Illustration d'une régularisation non paramétrique

est souvent associé à des hypothèses polynomiales de haute dimension. En effet, plus la dimension est élevée, plus la fonction peut expliquer une information précise du comportement observé. Or, la capacité explicative peut être telle que toute fonction de la classe peut expliquer parfaitement le comportement "grossier" issu des m exemples sans pour autant être pertinente sur l'ensemble du phénomène observé. Nous sommes donc en présence de sur-apprentissage car la fonction est trop spécialisée sur l'échantillon d'apprentissage au point de ne pas conserver une capacité de généralisation.

Il existe deux approches pour contraindre un espace d'hypothèses:

- **Approche paramétrique:** On constraint le nombre de paramètre de l'hypothèse, i.e sa dimension. Ainsi, il est possible de limiter le nombre de connexion dans le réseau de neurones.
- **Approche non paramétrique:** Cette approche ne considère pas le nombre de paramètres mais la *régularité* (smoothness) de la fonction. Ainsi, ce critère favorise les hypothèses représentées par une courbe "lisse". Une illustration est visible sur la Figure 5. Une régularisation non paramétrique va favoriser l'hypothèse représentée par la courbe verte. Par exemple, cette régularisation limitera le degrés polynomial de l'hypothèse choisie dans le cadre d'une régression car une fonction polynomiale "lisse" tend à être une fonction polynomiale de faible degrés.

2.7.2 Définition de la régularisation

La fonctionnelle (ou fonction) de pénalisation, que nous noterons Φ , définit la connaissance a priori qui caractérise les spécificités que doivent respecter les hypothèses jugées valides par le modèle. On cherche ainsi à minimiser le **risque pénalisé**:

$$R_{reg}(h)R_{emp}(h) + \lambda\Phi(h)$$

λ est un paramètre qui détermine le compromis entre la fidélité aux données d'apprentissage et les caractéristiques que doit avoir l'hypothèse. Il est difficile

de connaître les caractéristiques que doit avoir l'hypothèse. Afin d'assouplir la pénalisation en cas d'informations a priori "mauvaises", le critère de pénalisation est souvent ajusté. Ainsi, on fixe la fonctionnelle mais on ajuste sa pondération selon le phénomène observé de manière empirique à partir des données d'apprentissage. En faisant varier λ , on modifie la classe d'hypothèses valorisée par le modèle. Néanmoins, comme les exemples sont issus d'un sous-ensemble de la distribution de données, les hypothèses sont nécessairement *optimistes* car il est plus facile de satisfaire les conditions d'un ensemble ensemble de plus faible dimension. C'est pourquoi il est apprécié de réaliser l'apprentissage d'une fonction hypothèse pour différentes classes d'hypothèse (en faisant varier λ) et d'évaluer les différentes hypothèses sur un ensemble d'exemples indépendants de l'apprentissage (et du test). Nous parlons alors d'*ensemble de validation*.

2.7.3 Les formes de la régularisation

Il existe trois formes principales de régularisation. Ces formes sont complémentaires et peuvent être (éventuellement) cumulées selon le cas. Ainsi, nous avons:

- **Régularisation des poids:** Cette approche consiste à limiter l'importance des paramètres du modèle. En effet, un paramètre à valeur élevée favorise un sur-optimisme pour une caractéristique donnée, ce qui nuit à la capacité d'adaptation globale en sur-spécialisant l'hypothèse sur un type d'exemples non représentatif de l'ensemble du phénomène.
- **Early Stopping:** Cette approche provoque un arrêt prématuré de l'apprentissage. En effet, plus le modèle converge vers un optimum local, plus l'hypothèse devient complexe. En forçant un arrêt précoce, on limite la complexité de l'hypothèse et ainsi, le risque de perte de généralisation.
- **Échantillon bruité:** Cette approche repose sur un apprentissage à partir de données *bruitées*. Les données *bruitées* limite le risque que le modèle se spécialise sur l'échantillon d'apprentissage en accentuant les irrégularités locales. Ainsi, l'hypothèse choisie aura tendance à favoriser les régularités dites *profondes* et négligera les *détails*, accentués par le bruit imposé aux données.

2.8 No Free-lunch Theorem, un fondamental théorique (et grande désillusion ?)

No Free Lunch Theorem[119] est un théorème théorique **fondamental** de l'apprentissage automatique. Ce théorème démontre une limitation forte qui modère grandement l'engouement que l'on peut avoir pour un modèle d'apprentissage ou une classe d'algorithme.

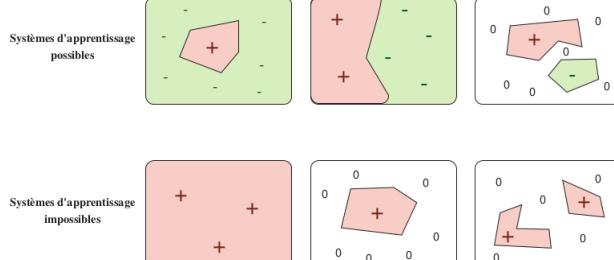


Figure 6: La fin du rêve: No Free Lunch Theorem

Supposons deux algorithmes d'apprentissage M_1 et M_2 quelconques et m , nombre de données d'apprentissage. Les propositions suivantes sont vraies:

1. En moyenne uniforme sur toutes les fonctions cible f dans \mathcal{F} :

$$E_{M_1}[R_{reel}|f, m] - E_{M_2}[R_{reel}|f, m] = 0$$
2. Pour tout échantillon d'apprentissage S , en moyenne uniforme sur toutes les fonctions cible de f dans \mathcal{F} :

$$E_{M_1}[R_{reel}|f, S] - E_{M_2}[R_{reel}|f, S] = 0$$
3. En moyenne uniforme sur toutes les distributions possibles $P(f)$:

$$E_{M_1}[R_{reel}|m] - E_{M_2}[R_{reel}|m] = 0$$
4. Pour tout échantillon d'apprentissage S , en moyenne uniforme sur toute les distribution $P(f)$:

$$E_{M_1}[R_{reel}|S] - E_{M_2}[R_{reel}|S] = 0$$

En résumé, ces propositions démontrent **qu'aucun algorithme d'optimisation numérique n'est plus efficace qu'un autre sur l'ensemble d'un espace des problèmes**. En d'autres mots, chaque algorithme d'optimisation (aussi extraordinaire soit-il selon son créateur) est aussi performant que l'algorithme de sélection aléatoire sur l'ensemble des problématiques. Celui qui sera performant sur une classe particulière de problèmes sera mauvais sur une autre, etc... Ainsi, pas de recette miracle ! Chaque problème demandera une étude de cas poussée et un algorithme sur mesure pour le résoudre. Pour illustrer, nous pouvons voir des performances possibles et impossibles de modèles d'apprentissage sur l'ensemble des fonction-cible possible. Les signes + et - correspondent à une performance respectivement supérieure et inférieure à la sélection aléatoire. 0 indique une performance équivalente au hasard.

Pour être capable de discriminer, un modèle repose sur des hypothèses. Ces hypothèses sont des **biais prédictifs**. Sans biais, impossible d'apprendre. Néanmoins, ces hypothèses peuvent être contre-productives pour un contexte donné et utiles dans d'autres. Il est donc nécessaire de faire des hypothèses au cas par cas, ce qui met fin au rêve d'un modèle parfait supérieur à tous !

Remarque: Cette restriction est néanmoins nuancée. En effet, parmi l'ensemble des problématiques de l'espace des problèmes, nombreuses sont celles qui ne correspondent pas à un phénomène réel pertinent ou "naturellement réalisable". De ce fait, il est possible de voir un modèle considéré comme "supérieur" du fait de sa grande performance sur des problématiques répandues dans notre société actuelle. Il ne faut pas oublier que cet avantage se paie dans une partie de l'espace, certes, non pertinente mais bien réelle.

De même, ce théorème s'applique à un modèle précis et non une classe de modèle. Dans une même classe, il existe une multitude d'architectures plus ou moins performantes selon la problématique. Dans le cadre du Deep Learning, les architectures sont particulièrement diversifiées, ce qui permet au Deep Learning d'être performant dans l'ensemble des problématiques actuelles du fait de cette richesse d'architecture. Mais il est fondamental de retenir qu'un modèle neuronal spécifique n'est pas supérieur à un autre, qu'une classe d'architecture neuronale n'est pas supérieure à une autre mais surtout, **le Deep Learning n'est pas plus performant que les architectures de Machine Learning classiques** (quoi qu'il puisse être vu, lu ou entendu...) !

3 Les Fondamentaux

3.1 Deep Learning, le fer de lance du Machine Learning

Le deep Learning est une catégorie d’algorithmes basée sur un assemblage spécifique d’entités nommées **neurones**¹⁹. Ces entités sont assemblées sous forme de couches plus ou moins nombreuses et profondes, à l’architecture variable selon la complexité du modèle recherché et du problème à résoudre.

Le Deep Learning modélise les données avec une haute capacité d’abstraction où chaque couche extrait et transforme les données issues de la couche précédente²⁰ avec une approche majoritairement **non linéaire**. Un réseau possède donc un niveau d’abstraction dépendant de son architecture. Ainsi, les couches les plus basses discrimineront des phénomènes simples et généralistes²¹ alors que les couches supérieures discrimineront des phénomènes plus complexes et caractéristiques du phénomène observé. La Figure 7 illustre un exemple d’analyse d’une image. Nous pouvons observer que la première couche discrimine des structures géométriques simples telles que des lignes ou des courbes alors que les couches suivantes discriminent des structures plus complexes. Il n’y a pas de méthodologie reconnue pour déterminer l’architecture *idéale* pour une problématique donnée. Seule l’approche empirique est employée à ce jour.

Aujourd’hui, le Deep Learning est très populaire pour ses résultats qui sont, dans de nombreux domaines, les meilleurs de l’état de l’art. Néanmoins, sa grande qualité repose sur sa capacité à s’émanciper du travail de pré-traitement des données nécessaires aux méthodes plus classiques. Par exemple, une image, pour être exploitée par des modèles standards, doit être pré-traitée pour extraire son contenu utile. Ces méthodes d’extraction sont généralistes, lourdes à employer et pas toujours efficaces. Les modèles de Deep Learning (notamment convolutifs²²) ont la capacité d’apprendre dynamiquement l’extraction du contenu utile d’une image. Ainsi, le réseau se spécialise - sans intervention humaine - sur le type d’image qu’il traite et bien souvent, dépasse les performances de méthodes de pré-traitement standards. Cette capacité d’auto-apprentissage rend le Deep Learning modulable, simple²³ et facilement déployable. En effet, en simplifiant, il suffit de donner la donnée brute et le réseau s’adaptera seul. Cette capacité se généralise à différents types de données: signal 1D (texte, signal sonore), signal 2D (image), séries temporelles... Cependant, il est nécessaire d’adapter l’architecture du modèle au type de données et au problème ciblé.

Cette souplesse d’utilisation a un coût non négligeable: **le volume de données**. Pour être fonctionnel, un réseau de Deep Learning doit exploiter un volume très

¹⁹Une analogie peut être réalisée avec le cerveau humain

²⁰Selon le type d’architecture, il peut y avoir des phénomènes de rétro-action

²¹Non spécifiques au phénomène observé.

²²Nous l’étudierons dans cette introduction

²³La simplicité est relative...

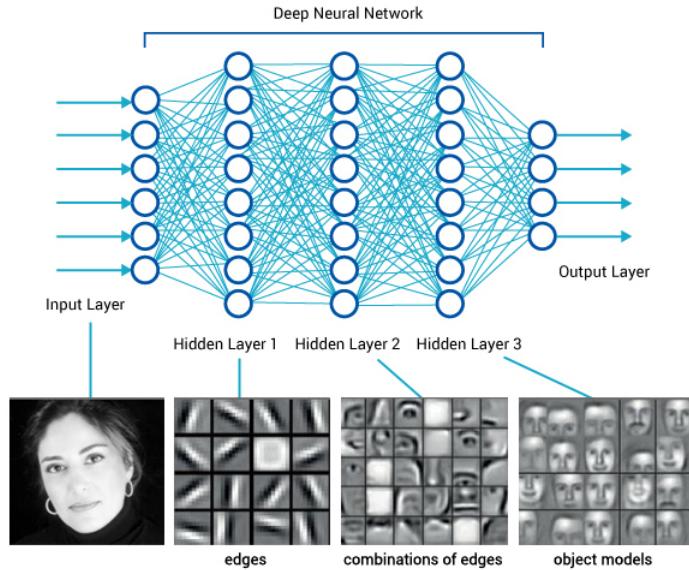


Figure 7: Relation entre profondeur du réseau et capacité d'abstraction

important de données durant son apprentissage²⁴. Aujourd’hui, peu d’acteurs du milieu public ou privé ont un volume suffisant à disposition. Le Deep Learning ne constitue donc pas un remplaçant absolu aux approches de Machine Learning traditionnelles mais une alternative lorsque l’apprentissage peut être réalisé dans de bonnes conditions. De même, les architectures de Deep Learning sont exigeantes en temps machine et en ressources. Il est parfois pertinent d’exploiter des modèles plus légers et accessibles lorsque la tâches à accomplir ne nécessite pas un modèle à forte complexité.

De ce fait, aujourd’hui, pour les tâches d’*optimisation*, le Machine Learning traditionnel garde une place dominante alors que sur des thématiques complexes ou créatives (génération automatique, traduction...), le Deep Learning devient nécessaire.

Important: Le Deep Learning profite d’une médiatisation importante qui tend à extrapoler ses qualités. Il est crucial de comprendre en profondeur les qualités et les défauts de ce type de modèle afin de ne pas alimenter les risques d’un développement non maîtrisé et biaisé.

²⁴Le volume dépend de la tâche à accomplir et de la nature des données

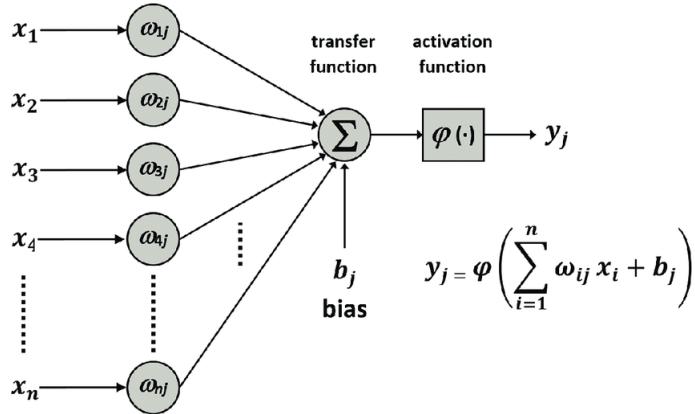


Figure 8: La structure d'un neurone (perceptron)

3.2 Un neurone: le perceptron

Le modèle le plus simple correspond au modèle unitaire: un seul neurone. Ce modèle est appelé **perceptron** et a été inventé en 1957 par Frank Rosenblatt. La Figure 8 représente un neurone.

Le perceptron (neurone) **selon Rosenblatt** est défini par différents éléments. Ainsi, un neurone j est décrit par:

- Des données d'entrée: x_1, x_2, \dots, x_n
- Des poids (qui pondèrent les données d'entrée): $\omega_1, \dots, \omega_n$
- Un biais (qui régule l'activation du neurone): b_j
- Une fonction d'activation (qui modélise la réponse du neurone en fonction des entrées pondérées): $\varphi()$ avec $\varphi = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x > 0 \end{cases}$ (fonction heavy-side²⁵)
- Une réponse (ou sortie): y_j

Le fonctionnement d'un neurone est simple.

Supposons une entrée $[x_1, \dots, x_n]$ (de dimension n). Chaque entrée i est pondérée par un poids ω_i associé²⁶. Ainsi: $[x_1, \dots, x_n] \xrightarrow{\omega} [x_1 * w_1, \dots, x_n * w_n]$

Une somme est réalisée de l'ensemble des données d'entrée pondérées et le biais b_j . On obtient donc: $\text{transfer_function} = \sum_{i=1}^n (x_i * w_i) + b_j$

Le biais est souvent représenté comme une valeur d'entrée constante (supposons

²⁵Nous verrons par la suite que c'est un très mauvais choix de fonction !

²⁶il y a donc n poids

1 par exemple) et pondérée par un poids comme le serait une autre entrée. Ainsi, nous obtenons: $transfer_function = \sum_{i=1}^n (x_i * w_i) + w_0 = \sum_{i=0}^n (x_i * w_i)$ avec $x_0 = 1$

La somme obtenue est transformée par une fonction d'activation $\varphi()$ et définira la sortie y du neurone: $y = \varphi(transfer_function)$

3.3 Le biais

Le biais est utilisé pour éviter que l'hyperplan réalisé par le neurone ait l'obligation de passer par l'origine. L'impact, la détermination et la représentation du biais sont encore un sujet d'étude et de recherche. Afin de ne pas complexifier cette introduction avec des approches mathématiques *secondaires*, nous n'approfondirons pas cette problématique. Cependant, il est important d'avoir une sensibilité sur son utilité. Un exemple simple permet de la comprendre.

Supposons une image complètement noire (pixels à 0) et une sortie souhaitée égale à λ . Cette problématique est insoluble car, avec les fonctions d'activation standards, la valeur de sortie par rapport à une stimulation nulle est invariable et correspond à la valeur de la fonction d'activation en 0. Supposons la fonction heavyside, la sortie serait donc de 0.5 et constante.

Néanmoins, dans le cadre d'un réseau profond, le biais tend à voir son impact diminuer avec la complexité du modèle où le comportement d'activation est partiellement réalisé par des neurones de la couche précédente. Ainsi, dans le contexte du deep learning où les modèles se complexifient énormément, la gestion du biais peut être simplifiée voir "délaissee".

Dans les réseaux profonds, le biais est aussi utilisé pour se protéger des défauts liées à l'initialisation des neurones, notamment le phénomène *Dead ReLu*²⁷.

3.4 Le Perceptron Multicouche

Un neurone unique possède des qualités limitées²⁸ et ne peut résoudre que des problèmes simples²⁹. Afin de résoudre des problèmes plus complexes, il est nécessaire d'en assembler plusieurs afin de créer une architecture plus performante. La première architecture que nous verrons est le modèle *FeedForward*.

Le modèle FeedForward est caractérisé par des couches de neurones où chaque neurone est connecté à l'intégralité des neurones de la couche suivante, permettant une *propagation en avant* de l'information. Le réseau le plus populaire se reposant sur ce modèle est le Perceptron Multicouche. La Figure 9 en présente

²⁷Voir Section 3.7, partie fonction d'activation, ReLu

²⁸Par exemple, il ne peut traiter que des problèmes linéairement séparables

²⁹Ils sont souvent employés pour simuler des portes logiques par exemple

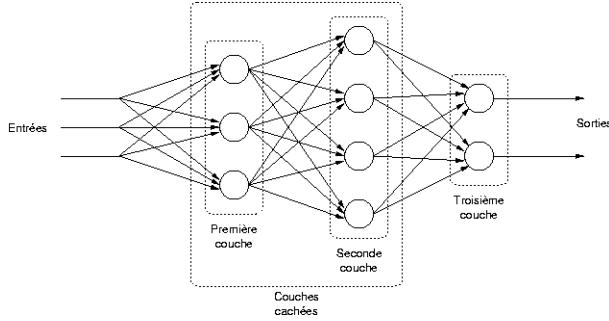


Figure 9: Modèle FeedForward: Le Perceptron multicouche

un exemple.

Ainsi, les données d'entrées sont réparties sur la couche d'entrée composée de 1 ou plusieurs neurones où chaque donnée est envoyée à l'intégralité des neurones. Les sorties de ces neurones sont connectées à l'intégralité des neurones de la couche suivante, appelée couche cachée. Il peut y avoir plusieurs couches cachées. Les données d'entrées des couches cachées sont donc les sorties des neurones de la couche précédente. La couche de sortie correspond à la dernière couche du réseau et la sortie des neurones de cette couche correspond à la prédiction réalisée par le réseau. Ainsi, par exemple, dans un problème de classification à N classes³⁰, la couche de sortie sera composée de N neurones où les N sorties correspondent à la prédiction associée à chacune des classes (souvent sous forme probabiliste).

Le nombre de neurones par couche et le nombre de couche nécessaire pour résoudre un problème ne peuvent être déterminés à l'avance. Il n'existe pas de méthode reconnue pour obtenir le réseau "idéal" pour résoudre un problème. Néanmoins, plus le nombre de neurones et/ou de couches est élevé, plus le modèle possède un haut pouvoir d'abstraction. Avoir une haute capacité d'abstraction est nécessaire sur des problématiques complexes mais augmenter la taille du réseau ne garantit pas un meilleur résultat sur l'ensemble des problèmes et présente des risques non négligeables que nous approfondirons dans la suite de cette introduction.

En théorie, un perceptron multicouche à une couche cachée est capable d'approximer toute fonction d'un sous-ensemble compact de R^n avec un nombre fini de neurones³¹. Il est donc considéré comme un **approximateur universel** de fonctions. Ce résultat est théorique et ne considère pas les difficultés d'apprentissage

³⁰Voir Section 3.10 pour plus de détails

³¹Malheureusement, la méthode d'apprentissage à base de descente de gradient limite grandement les capacités d'apprentissage d'un réseau. Il n'existe pas d'alternative significativement meilleure actuellement.

et de stabilité pour des problèmes complexes en très grande dimension caractéristique du Deep Learning.

3.5 L'apprentissage du neurone

Un neurone est défini par ses poids (et son biais). Lors de la création du neurone, les poids sont générés arbitrairement. Il est donc nécessaire de lui permettre *d'apprendre*, i.e mettre à jour ses poids, afin qu'il puisse limiter ses erreurs de prédiction, i.e minimiser la valeur de la fonction de coût. Cet objectif est réalisé en deux étapes: l'étape *Forward* et l'étape *Backward*.

L'étape *Forward* consiste à déterminer une prédiction réalisée par le neurone et à calculer l'erreur (potentiellement cumulée) réalisée par rapport à une valeur de référence indiquée par les données d'apprentissage. L'étape *Backward* réalise la mise à jour des poids du neurone en fonction de l'erreur réalisée grâce à l'algorithme de **Rétropropagation du gradient**.

3.5.1 Etape Forward: Fonction de coût

Afin de pouvoir mettre à jour les poids, il est nécessaire de savoir lorsque le neurone s'est trompé et quelle est l'ampleur de son erreur. La fonction de coût est utilisée pour quantifier cette erreur. Il existe différentes fonctions de coût selon la problématique à traiter. Les principales sont définies par:

Soit \hat{y} , prédiction du neurone, y , valeur de référence de la donnée d'apprentissage et n , nombre de prédictions faites:

- Problème de **régession**:

- **Mean Squared Error:** $\mathcal{L} = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2 = \frac{1}{n} \sum_{i=1}^n (\|\hat{y}^{(i)} - y^{(i)}\|_2^2)$

Cette fonction est la fonction la plus répandue et standard. Sa forme quadratique justifie qu'elle est convexe et ainsi, favorise l'apprentissage du réseau de neurone qui s'apparente à un problème d'optimisation.

Cette fonction, du fait de la mise au carré, maximise l'importance des *grandes erreurs*. C'est une spécificité importante à considérer car elle rendra l'apprentissage sensible aux valeurs extrêmes. De plus, cette fonction favorise une vitesse de convergence relativement lente du réseau de neurone.

- **Mean Squared Logarithmic Error:** $\mathcal{L} = \frac{1}{n} \sum_{i=1}^n (\log(y^{(i)} + 1) - \log(\hat{y}^{(i)} + 1))^2$

Cette fonction est une variante de Mean Squared Error à la différence qu'elle est moins sensible aux *grandes erreurs* de prédiction. Dans le

cas de données non normalisées aux échelles différentes, cette fonction limite la pénalisation des grands écarts de prédiction qui peuvent être liés, dans ce contexte, à l'échelle des données et non une véritable erreur de prédiction.

- **Mean Absolute Error:** $\mathcal{L} = \frac{1}{n} \sum_{i=1}^n (|\hat{y}_i - y_i|) = \frac{1}{n} \sum_{i=1}^n (\|\hat{y}_i - y_i\|_1)$

Alors que *Mean Squared Error* est empiriquement plus précise et performante dans le cadre d'une optimisation, *Mean Absolute Error* produit des solutions plus *éparses*³², ce qui est utile dans le cadre d'extraction d'attribut pour les problèmes à haute dimension. De même, cette fonction de perte est plus résistante aux valeurs aberrantes (*outliers*) et ignore les détails spécifiques, ce qui peut être utile pour lutter contre le sur-apprentissage. Néanmoins, cet aspect est responsable d'une perte d'informations, ce qui tend à la rendre empiriquement moins performante que son homologue basé sur la distance L_2 bien que plus stable.

Dans les faits, cette fonction n'est pas dérivable en 0, ce qui est problématique dans le cadre de la rétropropagation du gradient³³. Pour corriger ce défaut, une variante a été proposée nommée *Smooth L₁*:

$$|d|_{\text{smooth}} = \begin{cases} 0.5d^2, & \text{if } |d| \leq 1 \\ |d| - 0.5, & \text{otherwise} \end{cases}$$

- Problème de **classification**:

- **Cross Entropy:** $\mathcal{L} = -\frac{1}{n} \sum_{i=1}^n [y^{(i)} \log(\hat{y}^{(i)}) + (1-y^{(i)}) \log(1-\hat{y}^{(i)})]$

Cette fonction est la fonction standard dans le cadre de la classification binaire. Contrairement aux fonctions quadratiques, elle possède une vitesse de convergence plus rapide et tend à favoriser la convergence vers le minimum global (mais rien ne garantit cela !)

- **Negative Logarithmic Likelihood:** $\mathcal{L} = -\frac{1}{n} \sum_{i=1}^n \log(\hat{y}^{(i)})$

Cette fonction de coût est employée dans le cadre de problème de classification multi-classe. Elle peut être considérée comme une généralisation de la Cross Entropy.

Il existe de nombreuses autres fonctions de coût aux spécificités particulières. Le choix de la fonction de coût demande de l'intuition et une certaine sensibilité

³²Isole les attributs fortement discriminants en minorant les attributs faiblement informatifs
³³Cet aspect sera détaillé par la suite.

mathématique. Cependant, son choix est crucial pour un fonctionnement optimal du réseau de neurones. Le travail d'analyse des différentes fonctions est un étape indispensable pour l'étude de problèmes complexes.

3.5.2 Etape Backward: Rétropropagation du gradient

La fonction de coût détermine l'erreur du neurone durant son apprentissage. L'étape suivante est de permettre au neurone d'apprendre, d'évoluer en fonction de l'erreur réalisée et de faire varier chaque poids selon son impact dans l'erreur produite. Pour cela, la *descente de gradient*³⁴ est utilisée *localement* et son application récursive sur les différents neurones du réseau forme l'algorithme de **Rétropropagation du gradient**.

La descente du gradient est un algorithme d'optimisation exploitée dans le cadre de la minimisation (ou de la maximisation) d'une fonction objectif $f(x)$ paramétrée par x . Dans le cadre du *Machine Learning*, nous chercherons à minimiser la fonction objectif, i.e diminuer l'erreur d'apprentissage. La fonction objectif est de dimension R^d où d , nombre de paramètres du modèle.

3.5.2.1 Les méthodes de descente

La méthode de *Descente du gradient* est une méthode de descente. Avant d'expliquer les spécificités de la descente du gradient, il est nécessaire d'expliciter la notion de méthode de descente.

A partir d'un point x_0 , une méthode de descente va générer une suite $(x_k)_{k \in \mathbb{N}}$ telle que:

$$\begin{aligned} x_{k+1} &= x_k + \alpha_k d_k \\ \forall k \in \mathbb{N}, f(x_{k+1}) &\leq f(x_k) \end{aligned}$$

Deux paramètres sont donc à déterminer: la direction de descente d_k et la valeur du pas α_k . La méthode de détermination de la direction de descente est utilisée pour nommer l'algorithme. La recherche de la valeur du pas est nommée *recherche linéaire*.

Important: Cette partie a pour vocation d'introduire le concept d'optimisation différentiable nécessaire pour comprendre l'algorithme de Rétropropagation du gradient. De ce fait, elle est très restreinte et incomplète pour une compréhension approfondie. La section 3.6 approfondit spécifiquement les spécificités de ces algorithmes dans le cadre du Deep Learning, notamment le **calcul du pas** et les particularités liées au jeu d'apprentissage.

³⁴Il s'agit d'une méthode d'optimisation simple mais efficace. Il est possible d'en exploiter d'autres mais à ce jour, cette approche fait consensus dans la communauté scientifique.

3.5.2.2 La direction de descente

Soit une fonction objectif $f \in \mathbb{R}^n$. Une direction de descente est définie par un vecteur $d \in \mathbb{R}^n$ si pour un point $x \in \mathbb{R}^n$, $t \rightarrow f(x + td)$ est décroissante pour $t = 0$. En d'autres mots, s'il existe $\epsilon > 0$ tel que:

$$\forall t \in]0, \epsilon], f(x + td) < f(x) \quad (1)$$

Dans le cadre classique des méthodes de descente, la fonction à minimiser est *differentiable*³⁵. De ce fait, nous pouvons dire que d est une direction de descente de f en $x \in \mathbb{R}$ ssi:

$$f'(x; d) = f'(x).d = \langle \nabla f(x), d \rangle = \nabla f(x)^T d < 0$$

Ceci garantie une décroissance de la fonction f selon la direction de d car:

$$\forall \beta < 1, \exists \epsilon > 0, \forall t \in [0, \epsilon], f(x + td) < f(x) + t\beta \nabla f(x)^T d < f(x)$$

3.5.2.3 Algorithme des méthodes à direction de descente

L'objectif de cet algorithme est de déterminer $\min_{x \in \mathbb{R}^n} f(x)$.

Supposons $f : \mathbb{R}^n \rightarrow \mathbb{R}$ differentiable³⁶ et $|\epsilon| \in [0, \nu]$ avec ν , petit. Au début de l'itération k , nous disposons de $x_k \in \mathbb{R}^n$. Ainsi:

1. *Critère d'arrêt*: si $\nabla f(x_k) \leq \epsilon$, arrêt de l'algorithme
2. Sélection d'une direction de descente $d_k \in \{d \in \mathbb{R}^n : \langle \nabla f(x), d \rangle < 0\}$
3. *Recherche linéaire*: calcul du pas α_k selon une approche spécifique
4. $x_{k+1} = x_k + \alpha_k d_k$

Tant que le critère d'arrêt n'est pas atteint, l'algorithme continue à itérer. Il s'agit donc d'une famille d'algorithme **itératif**.

3.5.2.4 Détermination de la direction de descente

L'une des difficultés de cette classe d'algorithme d'optimisation est la détermination de la direction de descente. C'est un sujet de recherche toujours actif. Dans le cadre de cette section, nous nous limiterons à l'étude des méthodes de descente en optimisation différentiable sans contrainte.

Aujourd'hui, deux stratégies sont exploitées:

³⁵Dans les faits, nous souhaitons qu'elle soit une fois ou deux fois differentiable selon la méthode de descente choisie.

³⁶De degrés nécessaire selon les méthodes internes choisies

- **L'approche de Cauchy:** L'approche de Cauchy repose sur l'exploitation du gradient, i.e:

$$d_k = -\nabla f(x_k)$$

Elle est à l'origine des méthodes dites *algorithmes du gradient* (ou de la plus profonde descente). Cette approche est la **référence** dans le cadre des algorithmes de Deep Learning car les algorithmes associés sont de faible complexité algorithmique et de ce fait, véloces. Néanmoins, ses performances théoriques sont limitées bien qu'empiriquement satisfaisantes. Pour pouvoir exploiter cette approche, la fonction à optimiser doit être **au moins** une fois différentiable.

Le pas optimal t_k est défini par:

$$\nabla f(x_k + t_k d_k) = 0$$

En d'autres mots, l'algorithme converge vers un *point stationnaire*, plus spécifiquement un **minimum local**. Le fait qu'il soit local est une caractéristique très importante car à l'origine d'une faiblesse théorique de performances.

- **Approche du gradient conjugué:** L'algorithme du gradient conjugué est une variante des algorithmes du gradient. L'idée est de construire itérativement des directions d_k mutuellement *conjuguées*. Ainsi, chaque direction d_k est obtenue par combinaison linéaire du gradient en x_k et de la direction précédente d_{k-1} , elle-même dépendante de d_{k-2} etc... Ainsi, nous avons:

$$d_k = \begin{cases} -\nabla f(x_k) & \text{si } k = 0 \\ -\nabla f(x_k) + \beta_k d_{k-1} & \text{si } k > 0 \end{cases}$$

Cette approche a inspiré les méthodes dites *de moment* dans le cadre des optimizers³⁷ utilisés pour le Deep Learning.

- **Stratégie de Newton:** L'approche de Newton repose sur l'exploitation du hessien, i.e

$$d_k = -H[f](x_k)^{-1} \nabla f(x_k) = -\nabla^2 f(x_k)^{-1} \nabla f(x_k)$$

Elle est à l'origine des méthodes dites *algorithmes de Newton*. Comparée à l'approche par gradient, cette stratégie est bien plus performante et théoriquement robuste. Néanmoins, calculer le hessien, dans le cadre du Deep Learning, est techniquement irréalisable. En effet, l'algorithme est de complexité $O(n^3)$, ce qui le rend trop gourmand en capacités computationnelles³⁸. Du fait de l'utilisation du hessien, la fonction à optimiser doit être **au moins** deux fois différentiable et le hessien, inversible.

³⁷La fonction d'optimizer sera détaillée en détail dans la suite de cette introduction

³⁸Cette méthode pose aussi problème au niveau du coût mémoire nécessaire à stocker le hessien. La matrice hessienne est de dimension $n \times n$ pour une fonction à optimiser sur \mathbb{R}^n , ce qui peut être très lourd dans le cadre d'un modèle neuronal très profond avec des millions de paramètres.

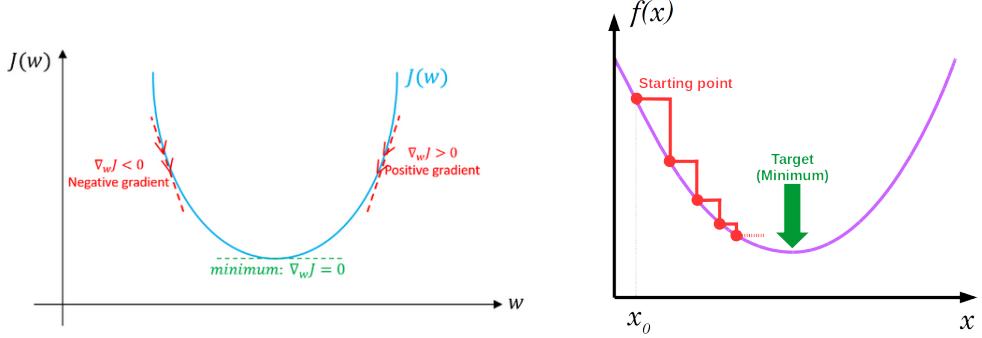


Figure 10: Illustration de la descente de gradient

Bien que non fonctionnelle en l'état, cette approche a inspiré des alternatives dans le cadre du Deep Learning. C'est notamment le cas de l'optimizer *Adadelta*.

Il existe de nombreuses autres stratégies. Nous ne les approfondirons pas car les méthodes du gradient sont les seules véritablement exploitées dans le cadre du Deep Learning³⁹.

Le Figure 10 illustre le comportement d'une optimisation par descente de gradient. La descente de gradient ne garantit pas de trouver le minimum **global** ! Cette particularité est très importante car cela signifie qu'un modèle neuronal ne converge pas vers un état idéal mais uniquement vers la meilleure tendance locale ! Deux modèles ayant appris sur des mêmes données peuvent donc être différents. Ceci est très problématique et responsable d'une faiblesse théorique de ce type de modèle et plus globalement, du Machine Learning actuel.

3.5.2.5 Rétropropagation du gradient - A FAIRE

3.5.2.6 Fonction d'activation

Bien que le perceptron de Rosenblatt utilise la fonction Heavyside comme fonction d'activation, il est possible d'en utiliser d'autres. Comme pour le choix de la fonction de coût, le choix de la fonction d'activation est capitale dans le développement d'un réseau de neurones. Il n'y a pas de méthodes clairement définies dans le choix de cette fonction. Les méthodes sont exclusivement empiriques bien que certaines fonctions semblent "sortir du lot"[52].

Pour être pleinement fonctionnelle, les fonctions d'activations doivent présenter des caractéristiques particulières dépendantes des spécificités de l'algorithme de *Rétropropagation du gradient*:

³⁹Bien que la recherche soit activée sur l'utilisation d'approches plus performantes.

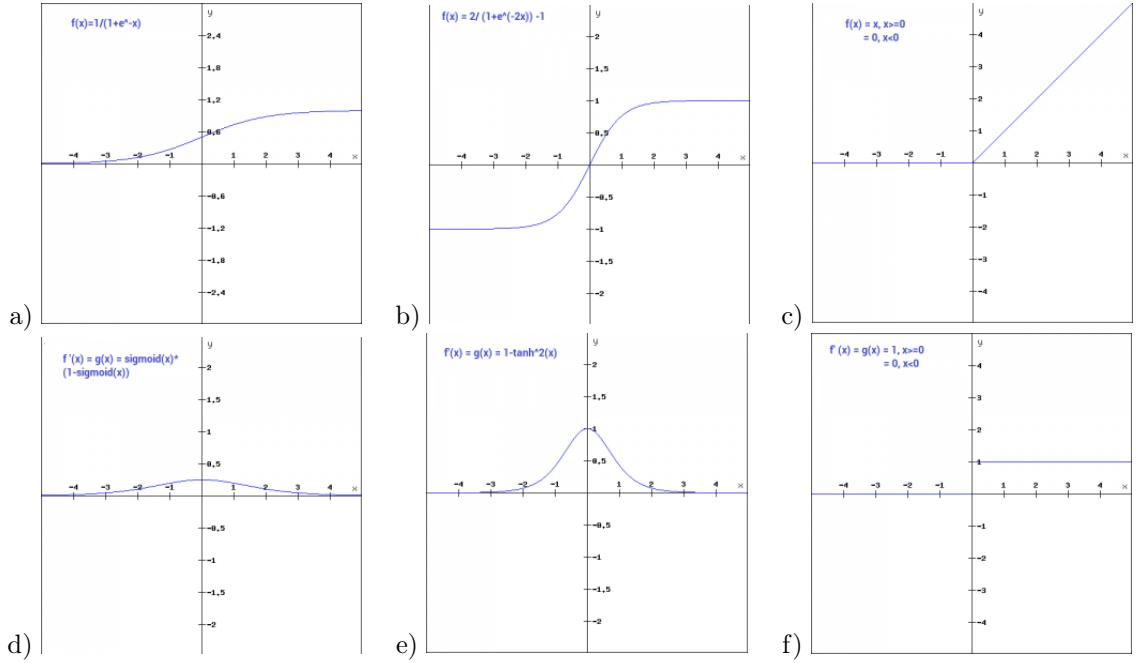


Figure 11: Fonctions d'activation: a) Sigmoid, b) tanh, c) ReLu et leurs dérivées associées

- **Être dérivable en tout point:** Le gradient étant dépendant de la dérivée de la fonction d'activation, une fonction non dérivable est difficilement utilisable.
- **La dérivé doit être non nulle:** Si la dérivée de la fonction d'activation est nulle, le gradient sera nul. De ce fait, l'apprentissage est impossible. Ce problème justifie la non-efficacité de la fonction Heavyside du perceptron standard car il **ne peut pas apprendre** en faisant varier ses poids.
- **Être non linéaire:** Cette condition n'est pas une nécessité absolue mais permet de considérer les problématiques non linéairement séparables.

Les principales fonctions d'activation actuelles sont:

- Fonction Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$

La fonction sigmoïde est une référence historique en tant que fonction d'activation et reste aujourd'hui, très utilisée. Elle est appréciée pour sa dérivée non constante, ce qui permet d'avoir un gradient variable selon la valeur et donc, une convergence plus performante. Néanmoins, le fait qu'elle soit positive la rend peu performante sur certains problèmes. De

plus, cette fonction favorise le phénomène de *vanishing gradient*⁴⁰. Un effet de bord est aussi présent. En effet, au-delà de (-3,3), la valeur de la dérivée est très faible ce qui impliquera un gradient faible. Un gradient très faible limitera grandement les variations des poids et augmentera le temps d'apprentissage du neurone (voire le "freezera").

Cette fonction est souvent utilisée sur la couche de sortie pour représenter une prédiction sous forme de probabilité (car résultat entre 0 et 1). C'est la fonction de référence dans le cadre de la classification binaire probabiliste.

Remarque: Un perceptron avec la sigmoïde comme fonction d'activation est identique à une régression logistique⁴¹.

- Fonction tanh: $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

La fonction tanh présente les mêmes caractéristiques que la fonction sigmoïde. La différence se situe sur le fait qu'elle est symétrique en 0 et prend valeur dans (-1,1), ce qui lui permet de ne pas être limitée par une condition de valeur positive. De plus, on observe un pic plus prononcé sur sa dérivée, ce qui implique un gradient plus important. Ceci est une qualité et un défaut car selon le problème, ça peut accélérer la convergence ou, au contraire, l'empêcher de pleinement converger.

- Fonction ReLu: $f(x) = \begin{cases} 0 & \text{si } x < 0 \\ \max(0, x) & \text{si } x \geq 0 \end{cases}$

La fonction ReLu est, aujourd'hui, la fonction d'activation la plus employée dans les réseaux de neurones au niveau des couches cachées⁴². Cette fonction présente des particularités très importantes dans le cadre des réseaux profonds. Nous l'étudierons en profondeur dans la suite de cette introduction.

- Fonction Softmax: $\delta(z)_j = \frac{e^{z_j}}{\sum_{i=1}^N e^{z_i}}$ avec $z = (z_1, \dots, z_N)$ et $j \in [1, \dots, N]$

La fonction Softmax est une version généralisée de la fonction Sigmoid. Elle est ainsi utilisée dans la couche de sortie pour des problèmes de classifications multi-classes (voir Section 3.10) avec représentation probabiliste. Elle est employée lorsque la couche de sortie possède différents neurones associés aux classes prédictes afin d'extraire la classe dont la probabilité est la plus élevée.

⁴⁰Cette notion sera étudiée par la suite

⁴¹Soyez fier de vous. Nous venons de réinventer la roue !

⁴²Cette fonction présente peu d'intérêt en couche de sortie du fait de son faible pouvoir explicatif

3.6 Descente du gradient et optimiseur

Dans la section 3.5.2, nous avons observé que la correction appliquée sur les poids est dépendante de l'erreur totale du réseau. Il est important de définir comment cette erreur est calculée. Pour cela, il y a trois approches complémentaires: la détermination du nombres de données d'apprentissage analysées pour une même itération, la détermination du pas d'apprentissage optimal et la méthode de calcul du gradient.

3.6.1 Approche par Batch, Minibatch et Stochastique

Le calcul du gradient se présente sous trois formes distinctes dépendant du nombre de données d'apprentissage utilisées pour déterminer une valeur de gradient à rétro-propager.

3.6.1.1 Approche par Batch

L'approche par batch correspond à la version originale du calcul du gradient. Cette approche utilise l'intégralité des données d'apprentissage avant de faire une rétro-propagation (une mise à jour des poids). Ainsi, l'erreur du réseau correspond à la moyenne de l'erreur réalisée par les prédictions de l'intégralité des données d'apprentissage.

Supposons la métrique *Mean Squared Error* et un jeu de données de m entités, nous obtenons donc:

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta)$$
$$J(\theta) = \frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_i)^2$$

Cette approche tend à définir un gradient stable permettant de limiter les "convergences oscillantes" du réseau. Il permet donc, en théorie, de favoriser une convergence lissée. Néanmoins, elle tend à converger vers un minimum local moins performant du fait de la trop grande généralité du gradient⁴³. De plus, il est nécessaire de calculer l'intégralité des erreurs avant de réaliser une mise à jour. Bien que ces calculs puissent être parallélisés, les mises à jour des poids sont lentes et de ce fait, l'apprentissage du réseau aussi. Dans le cas de jeu de données massifs (plusieurs millions voir milliards de données), cette approche est inutilisable en temps humain.

3.6.1.2 Approche Stochastique

L'approche stochastique calcule l'erreur avec une donnée d'apprentissage uniquement. Ainsi, il y a un apprentissage par rétropropagation à chaque prédiction

⁴³Rappelez-vous le compromis biais-variance. Trop de généralité nuit à la spécialisation du modèle !

réalisée durant l'apprentissage.

Supposons la métrique *Mean Squared Error*, nous obtenons donc:

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i)}; y^{(i)})$$

$$J(\theta; x^{(i)}; y^{(i)}) = (\hat{y}^{(i)} - y^{(i)})^2$$

Cette approche permet une évolution rapide du réseau du fait de la mise à jour récurrente des poids à chaque prédiction d'apprentissage. L'aspect stochastique favorise un gradient *bruité* qui limite le risque de convergence précipitée vers un minimum local mais favorise aussi la convergence vers un minimum potentiellement moins performant. De plus, l'aspect bruité du gradient entraîne une forte variance dans la mise à jour des poids et de ce fait, tend à rendre l'évolution du réseau moins *lissée*. Cette approche est très sensible aux données aberrantes et extrêmes. De même, il est préférable de sélectionner aléatoirement la donnée d'apprentissage afin d'éviter un biais associé à une répartition constante des données au fil des *epochs*.

3.6.1.3 Approche par MiniBatch

L'approche par MiniBatch est un compromis entre l'approche par Batch et Stochastique. Avec cette méthode, l'erreur est calculé à partir d'un sous-ensemble du jeu d'apprentissage. Cette méthode est la plus répandue aujourd'hui du fait de ses performances.

Supposons la métrique *Mean Squared Error*, un jeu de données de m entités et un minibatch de n entités. Nous obtenons donc:

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

$$J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

Cette méthode permet une plus grande robustesse et un meilleur contrôle de l'évolution du réseau que l'approche stochastique tout en conservant une mise à jour du réseau "régulière", ce qui permet une évolution du modèle plus rapide mais surtout, son exploitation. Le MiniBatch étant de taille restreinte, la problématique de convergence précipitée de l'approche par Batch est évitée. Cette méthode offre ainsi un bon compromis qui favorise les performances, de même que pour les facilités d'implémentation en limitant les problématiques de stockage des données d'apprentissage en mémoire. Néanmoins, le nombre d'entités dans le MiniBatch est un hyperparamètre important qui ne peut être défini qu'empiriquement. Par défaut, on a tendance à exploiter des minibatches de 32 éléments ou plus afin d'avoir une significativité statistique. De même, il est préférable de sélectionner aléatoirement les données d'apprentissage afin d'éviter un biais associé à une répartition constante des données au fil des *epochs*.

3.6.2 La problématique de la détermination du pas - A FAIRE

khjh

3.6.3 Aperçu des optimizer

Lors de la mise à jour des poids, le pas (facteur multiplicatif du gradient) a une grande importance. Une des grandes difficultés est de déterminer qu'elle est sa valeur optimale et ce, de manière statique ou dynamique. Dans la version initiale de l'algorithme du gradient, le pas est un hyperparamètre constant et fixé par l'utilisateur. Cette méthode s'avère difficile à optimiser et peu efficace expérimentalement.

Pour corriger ce problème, des améliorations (appelées *optimizer*) ont été proposées. Deux facteurs d'optimisation existent: la méthode de détermination du pas d'apprentissage et l'étude du *moment* des gradients. Le moment représente "l'engouement" que l'on a dans la mise à jour du gradient proposé. Supposons une succession de gradients dans la même direction⁴⁴ et de grande intensité, nous pouvons donc supposer être sur une voie de convergence favorable. De ce fait, il faut favoriser la mise à jour associée. Au contraire, si la valeur s'amoindrit ou que la direction varie, il faut limiter l'engouement de la mise à jour afin de ne pas "se perdre" (divergence ou convergence interrompue à cause d'une valeur de pas qui la bloque). Le moment permet ainsi de limiter les oscillations durant la convergence, d'avoir une mémoire sur son évolution et d'accélérer la convergence.

parler de:

- faire section: problematique du pas avec sgd
- snapshot ensemble <https://arxiv.org/pdf/1704.00109.pdf>
- second ordre, problème lgfgs <http://cs231n.github.io/neural-networks-3/#second>

3.6.3.1 SGD Annealing

Par défaut, la descente du gradient exploite un pas fixe. L'invariabilité du pas est une problématique majeure. En effet, un pas important ne permet pas de converger au plus bas du minimum local considéré. Un pas important aura tendance à s'extraire du minimum local ou à limiter grandement la précision de l'optimisation en se limitant à osciller entre deux positions approximatives. Un pas plus faible permet une plus grande sensibilité de l'optimisation en limitant l'impact de la variation, ce qui permet une plus grande précision.

Un pas important permet d'*explorer* l'univers des hypothèses et permet une plus grande robustesse à la convergence vers le minimum local le plus proche.

⁴⁴Imaginez une balle qui glisse le long d'une pente. L'énergie emmagasinée par la balle augmente avec la pente et diminue avec une montée ou un sol plat. Le moment peut être associé à l'énergie cinétique de cette balle

Au contraire, un pas faible permet une convergence plus précise mais est plus limité quant à sa capacité d'exploration. Intuitivement, nous pouvons considérer qu'un pas important est préférable au début d'un apprentissage afin que le réseau puisse explorer l'univers des hypothèses. Au contraire, plus le réseau apprend, plus il doit devenir précis et limiter l'exploration. Ainsi, le modèle doit optimiser la convergence vers le minimum local considéré. De ce fait, le pas doit être plus faible pour favoriser ce comportement.

Cette intuition mène à une amélioration de la détermination du pas. Ce dernier ne doit pas être fixe mais *diminuer au cours du temps* (annealing[81]). Ainsi, périodiquement, le pas va diminuer afin de passer d'un comportement exploratoire à un comportement de convergence.

La méthode de réduction périodique du pas est variable et présente une infinité de possibilités. Néanmoins, trois méthodes classiques sont répandues:

- **Step decay:** Cette approche consiste à retirer une **valeur fixe** à la valeur du pas selon un critère de périodicité fixe ou une condition particulière. Le critère de périodicité de référence est le nombre d'épochs. Ainsi, par exemple, nous pouvons diminuer d'un quart la valeur du pas à chaque époche (ou toutes les deux époches etc...).

Il est aussi possible de se baser sur une condition particulière, notamment la diminution de l'erreur d'apprentissage. Par exemple, au lieu de se baser sur le nombre d'époches, nous pouvons définir que le pas doit être diminué lorsque l'erreur d'apprentissage stagne (qu'on peut associer à un comportement oscillant de la convergence) ou encore, qu'elle augmente.

De même, il est possible de définir un critère sériel. Réduire le pas à chaque variation "néfaste" de l'erreur d'apprentissage possède les défauts d'un comportement stochastique, notamment la sur-exploitation des tendances locales qui peuvent nuire à la convergence. Appliquer la diminution du pas lorsqu'une augmentation de l'erreur d'apprentissage est présente β fois permet de favoriser une bonne interprétation de la tendance globale. β est un hyperparamètre qui peut être difficile à déterminer et relève essentiellement de l'instinct.

- **Exponential decay:** La variation par *Step* se comporte comme une variation *par palier*. Il peut être préférable d'avoir une variation plus lissée mais surtout à l'amplitude variable. Ainsi, retirer une valeur fixe à un pas élevé est moins impactant que retirer cette même valeur à un pas plus faible. Ainsi, il peut être préférable de retirer une valeur dépendant de la valeur actuelle du pas.

Exponential decay se base sur un facteur de variation exponentiel. Ainsi,

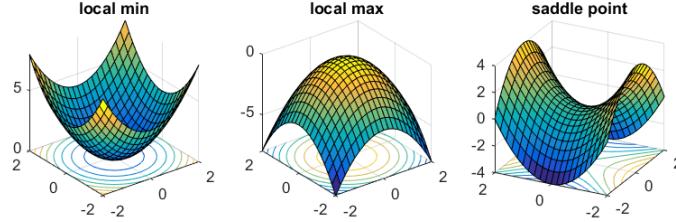


Figure 12: Différence entre minimum global, maximum global et point-selle

nous avons:

$$\alpha = \alpha_0 e^{-kt}$$

Avec α , pas actuel, α_0 , pas initial et k , critère multiplicatif qui détermine l'impact de la variation. Il s'agit d'hyperparamètre à définir par l'utilisateur.

- **1/t decay:** Cette approche suit la même logique que Exponential decay. Elle est définie par:

$$\alpha = \alpha_0 / (1 + kt)$$

Avec α , pas actuel, α_0 , pas initial et k , critère multiplicatif qui détermine l'impact de la variation. Il s'agit d'hyperparamètre à définir par l'utilisateur.

Dans les faits, lorsque ce type de méthode pour la détermination du pas est employée, **Step decay** est préféré car plus facilement interprétable d'un point de vue métier et pour la supervision de l'apprentissage. De plus, bien que plus efficace qu'une approche SGD classique, ces méthodes tendent à devenir obsolètes face à d'autres techniques plus sophistiquées sauf dans des cas d'architectures particulières.

3.6.3.2 Approche par cycle: Cyclical Learning Rate

Une des problématiques majeures de l'optimisation par descente de gradient est la convergence vers un **minimum local**. *SGD annealing*, du fait de la réduction permanente du pas durant l'optimisation, a tendance à être très sensible à ce problème. De même, la fonction de perte d'un réseau de neurones possède de nombreuses dimensions, ce qui favorise l'existence de **point-selle**. La Figure 12 illustre les spécificités de ce type de point. Ces points sont très nocifs à la qualité de la convergence et trompent l'algorithme du gradient. Pour favoriser la convergence vers le minimum global⁴⁵, une amélioration de SGD appelée Cyclical LR[102] a été proposée.

⁴⁵Il n'y a aucune certitude d'y arriver !

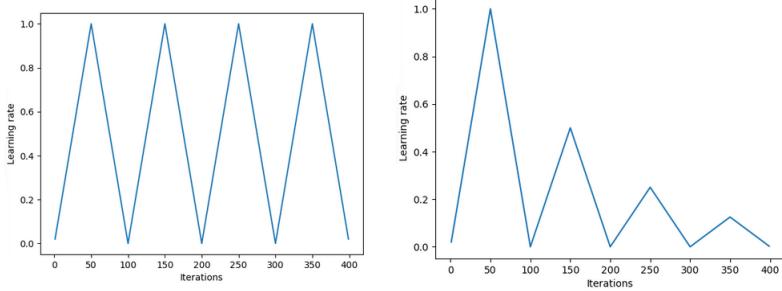


Figure 13: Comportement du pas avec Cyclical LR avec une borne supérieure constante et une borne supérieure dégressive

L'idée de Cyclical LR repose sur le fait d'augmenter périodiquement la valeur du pas afin de favoriser la sortie d'un minimum local instable (ou d'un point-selle). Si le minimum local (qui peut être global) est assez stable, le "pic" de la valeur du pas devrait être supporté et ne pas provoquer un changement de minimum. Le pas suit donc un comportement cyclique qui cherche à réduire sa valeur et à la rehausser périodiquement. Chaque cycle est défini par un nombre d'itérations identique et la valeur du pas est compris entre deux bornes choisies comme hyperparamètres. La variation du pas est linéaire et un cycle s'apparente à un comportement *triangulaire*. Une approche parabolique et sinusoïdale ont été proposées sans amélioration notable de performance.

Afin de limiter le temps d'apprentissage et de "forcer" la convergence vers un minimum local, une limitation sur la borne supérieure est applicable. Elle consiste à modifier la borne supérieure pour le cycle t tel que $\max_{LR_t} = \frac{\max_{LR_0}}{t}$. Ainsi, le pas tend vers 0, ce qui forcera le réseau à avoir un comportement de convergence et non d'exploration. Une alternative est possible en choisissant une minoration par un facteur exponentiel, ce qui favorisera une diminution plus "lissée". Une illustration du comportement du pas est visible sur la Figure 13.

3.6.3.3 Détermination des bornes supérieures et inférieures des valeurs du pas: le LR Range test

Une contrainte de Cyclical LR est la détermination des bornes de l'intervalle des valeurs potentielles du pas. Afin de résoudre ce problème, un test a été créé: le **LR Range test**[102].

Ce test consiste à définir un pas très faible, supposons 10^{-7} et à l'augmenter progressivement à chaque itération en observant l'impact sur le comportement de l'apprentissage. Lorsque le pas est faible, l'apprentissage stagne car la convergence est très lente. En augmentant, la convergence va se réaliser, ce qui

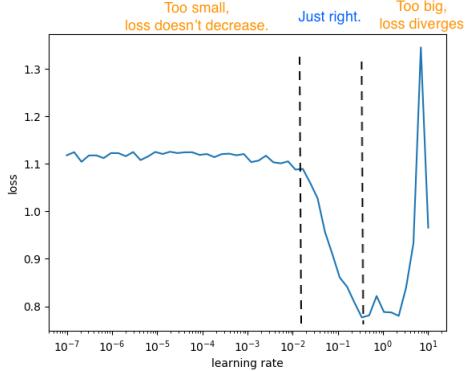


Figure 14: Exemple du LR Range test

permettra à l'erreur de diminuer. Lorsque le pas sera trop important, le modèle divergera et l'erreur augmentera. Nous pouvons ainsi observer trois régions qui permettront de définir un intervalle pertinent pour les valeurs du pas. Une illustration du test est visible sur la Figure 14. Il est important de noter qu'il s'agit d'un test empirique et que ses résultats sont dépendants du jeux de données. L'intervalle obtenu n'est donc pas généralisable.

3.6.3.4 Approche par cycle: Exploring Stochastic Gradient Descent with Warm Restarts (SGDR)

L'idée de SGDR[72] est une variante de Cyclical LR et exploite aussi une comportement cyclique. Contrairement à Cyclical LR qui possède un comportement linéaire dans la variation du pas, SGDR réalise une diminution du pas avant de le réinitialiser à la valeur de la borne supérieure, ce qui entraîne une variation non continue lors de la réinitialisation. Les bornes des valeurs du pas sont obtenues par LR Range test. Une illustration du comportement du pas est visible sur la Figure 15.

SGDR utilise une approche *cosinus annealing*⁴⁶ afin de diminuer la valeur du pas. Ainsi, pour le cycle i, le pas est égal à:

$$\eta_t = \eta_{min}^i + \frac{1}{2}(\eta_{max}^i - \eta_{min}^i)(1 + \cos(\frac{T_{cur}}{T_i}\pi))$$

Avec η_{min}^i , η_{max}^i , valeur minimum et maximum du pas, T_{cur} , le nombre d'itérations réalisées depuis la dernière réinitialisation du cycle et T_i , le nombre d'itérations du cycle i. Nous pouvons ainsi voir que si $t=0$ et $T_{cur} = 0$, $\eta_t = \eta_{max}^i$. Si $T_{cur} = T_i$, alors la composante cosinus sera égale à -1 et ainsi $\eta_t = \eta_{min}^i$. η_{min}^i et η_{max}^i sont des hyperparamètres.

⁴⁶Diminution du pas selon une fonction cosinus

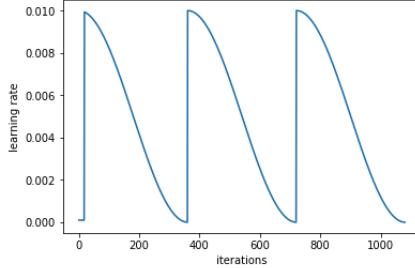


Figure 15: Comportement du pas avec SGDR

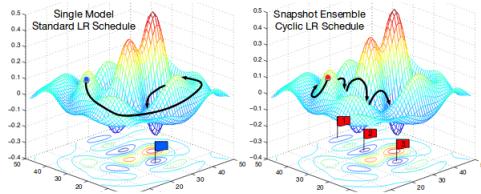


Figure 16: Différence entre SGD et Snapshot Ensembles

Afin d'améliorer les performances, il est possible d'augmenter la durée d'un cycle au fil des cycles. Pour augmenter progressivement le nombre d'itération par cycle, il est conseillé de remplacer $T_i = T_0$ par $T_i = T_0 * T_{mult}$ avec T_{mult} , coefficient multiplicateur incrémenté à chaque cycle selon une valeur choisie. Il est possible de faire varier η_{min}^i et η_{max}^i selon le cycle. Cependant, pour des raisons de simplicité, il est préférable de choisir des valeurs constantes.

Lors de la fin d'un cycle, la valeur du poids considéré n'est pas réinitialisée mais conservée. Ainsi, si x_t est la dernière valeur du poids à la fin du cycle, lors de la première itération du cycle suivant, la mise à jour du poids sera basée sur la valeur x_t . Cette caractéristique permet de conserver une continuité dans le comportement du poids et de ce fait, les valeurs des *momentum*⁴⁷ sont exploitables par exemple.

3.6.3.5 1Cycle Policy et Super-Convergence: A FAIRE

1cycle Policy[103]

3.6.3.6 Snapshot Ensembles

Snapshot Ensembles[40]

⁴⁷Cette notion sera expliquée par la suite.

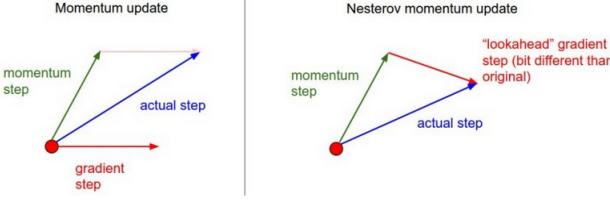


Figure 17: Différence entre Momentum et Nesterov Momentum

3.6.3.7 Calcul du moment: Vanilla Momentum

Cette méthode est simple. Elle est calculée en ajoutant à la valeur du gradient (pondéré par le pas), la valeur du gradient précédent pondérée par une constante qui représente l'importance associé au moment. Cette constante est un hyperparamètre choisi par l'utilisateur et souvent défini à 0.9 ou une valeur similaire⁴⁸.

Ainsi, le gradient est défini par:

$$v_t = \gamma v_{t-1} - \eta \nabla_{\theta} J(\theta)$$

$$\theta = \theta + v_t$$

3.6.3.8 Calcul du moment: Nesterov Accelerated gradient (NAG)

Nesterov Accelerated gradient[78] (NAG) est une variante du Vanilla Momentum. Alors que l'approche Vanilla Momentum n'anticipe pas ses positions futures, l'approche par Nesterov cherche à approximer la valeur future du gradient afin d'orienter son moment. L'estimation du gradient ne se fait donc plus à partir position actuelle mais à la position "prédicté" après l'approximation de la position future. L'amplitude des évolutions est donc mieux maîtrisée et permet une évolution plus précise et rapide. Cette approche possède une preuve théorique plus forte, est plus efficace dans le cadre d'optimisation convexe et semble donner de meilleurs résultats expérimentaux.

Une illustration est visible sur la Figure 17. La mise à jour du gradient par Momentum est réalisée à partir de l'état actuel (point rouge) alors qu'avec l'approche Nesterov, la mise à jour est réalisée à partir de la position prédicté (extrémité de la flèche verte). Pour plus d'explication, consulter le site <http://cs231n.github.io/neural-networks-3/#sgd>.

Ainsi, le gradient est défini par:

$$v_t = \gamma v_{t-1} - \eta \nabla_{\theta} J(\theta + \gamma v_{t-1})$$

$$\theta = \theta + v_t$$

⁴⁸Il faut éviter que sa valeur soit supérieure à 1.

3.6.3.9 Méthode adaptative: Adagrad

Adagrad[18] modifie le pas afin qu'il devienne dynamique et dépendant du paramètre optimisé. Ainsi, un paramètre ayant été peu modifié (en terme de variation de valeur) aura un pas important alors qu'un paramètre régulièrement mis à jour aura un pas minoré. Cette méthode nous émancipe de l'étude d'une valeur pour le pas, si ce n'est la détermination d'une valeur initiale (souvent placé à 0.01). En effet, il a été montré empiriquement qu'il est souvent plus efficace d'avoir un pas variable selon l'état du réseau.

Supposons une mise à jour d'un poids par une approche SGD *classique*. Nous avons:

$$g_{t,i} = \nabla_{\theta} J(\theta_{t,i})$$

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

Dans le cas de *Adagrad*, nous avons alors:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \epsilon}} \cdot g_{t,i}$$

$$G_{t,i} = \sum_{j=1}^t g_{j,i}^2$$

Le pas est ainsi minoré selon la somme au carré de l'ensemble des gradients précédents. L'utilisation de ϵ permet d'éviter le cas interdit de la division par 0. Sa valeur est constante et très faible. 10^{-8} par exemple.

La faiblesse de cette méthode est associée au facteur uniquement dégressif de la valeur du pas. En effet, selon cet optimizer, il ne peut "que" diminuer. Ainsi, au fil des apprentissages, le pas diminuera jusqu'à devenir infinitésimal et figera le réseau. Dans le cas d'apprentissage très profond, cette limitation est très problématique car la convergence n'est pas atteinte avant le blocage du pas.

3.6.3.10 Méthode adaptative: AdaDelta

AdaDelta[130] est une variante de Adagrad qui corrige le problème de la minoration agressive du pas. Ainsi, au lieu de considérer les différents gradients identiquement, cet optimizer se concentre sur une fenêtre des derniers gradients calculés en minorant proportionnellement les gradients selon leur ancienneté. Il propose aussi une méthode pour supprimer l'initialisation du pas, émancipant l'utilisateur d'un hyperparamètre.

Dans le cas de *AdaDelta*, l'approche de normalisation du pas est définie par:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) g_t^2$$

$$\Delta \theta_t = - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t$$

Ainsi, le facteur γ s'applique récursivement sur chaque valeur de gradient selon son ancienneté et permet de maîtriser la valeur du pas⁴⁹. Dans les faits, l'équation précédente est une approche naïve. En effet, elle présente un problème potentiel lié à l'unité de $\Delta\theta_t$. Ce problème est d'ailleurs présent avec les optimizers par moment et Adagrad.

Supposons une mise à jour par SGD classique pour un poids θ donné. Nous avons donc:

$$\Delta\theta = -\eta g$$

Soit $U_{\Delta\theta}$, unité de $\Delta\theta$ et f , fonction de perte. Nous avons donc:

$$U_{\Delta\theta} \propto U_g \propto \frac{\partial f}{\partial \theta} \propto \frac{1}{U_\theta}$$

Remarque: On fait l'hypothèse que la fonction de perte n'a pas d'unité.

On observe que les unités ne sont pas en phase. En effet, d'après l'équation de mise à jour des poids, $U_{\Delta\theta}$ devrait être égal à U_θ .

Pour corriger cette erreur, il a été proposé de s'inspirer d'une méthode du second ordre telle que la méthode de Newton qui repose sur une approximation hessienne. Ainsi, nous avons dorénavant:

$$\Delta\theta = -H^{-1}g$$

H correspond à la dérivée seconde de la fonction de perte. Ainsi, nous obtenons:

$$U_{\Delta\theta} \propto U_{H^{-1}g} \propto \frac{\frac{\partial f}{\partial \theta}}{\frac{\partial^2 f}{\partial^2 \theta}} \propto U_\theta$$

On observe que l'unité est respectée. Cette approche est donc efficace. Néanmoins, il faut définir le paramètre H^{-1} .

$$\begin{aligned} U_{\Delta\theta} &= \frac{\frac{\partial f}{\partial \theta}}{\frac{\partial^2 f}{\partial^2 \theta}} \\ \frac{1}{\frac{\partial^2 f}{\partial^2 \theta}} &= \frac{U_{\Delta\theta}}{\frac{\partial f}{\partial \theta}} \\ H^{-1} &= \frac{U_{\Delta\theta}}{\frac{\partial f}{\partial \theta}} \end{aligned}$$

D'où:

$$\Delta\theta = \frac{U_{\Delta\theta}}{\frac{\partial f}{\partial \theta}} g$$

⁴⁹Il est conseillé d'exploiter une valeur entre 0.5 et 0.9

D'après l'équation obtenue par l'approche naïve, nous avons:

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t}g_t$$

Il reste à modifier le numérateur afin de respecter l'unité. Pour cela, nous pouvons utiliser la même approche que pour le dénominateur soit:

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2$$

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$

Nous avons donc:

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_t}{RMS[g]_t}g_t$$

Malheureusement, la valeur $RMS[\Delta\theta]_t$ est inconnue. Afin de l'approximer, on fait l'hypothèse que la courbe est *localement* lisse. Nous pouvons donc approximer la valeur en exploitant la valeur jusqu'à la dernière mise à jour soit $\Delta\theta_{t-1}$. Ainsi:

$$\begin{aligned}\Delta\theta_t &= -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t}g_t \\ \theta_{t+1} &= \theta_t + \Delta\theta_t\end{aligned}$$

Cette approche est une version "simplifiée" d'une vraie optimisation par méthode de second ordre. En effet, calculer une inversion de matrice est une opération lourde ($O(n^3)$) et ne peut être exploitée. On observera qu'il n'y a plus d'hyperparamètre à définir. On supposera une valeur nulle à $\Delta\theta_0$.

3.6.3.11 Méthode adaptative: RMSprop

Cet optimizer est équivalent à la version naïve d'AdalDelta. Les hyperparamètres sont fixés selon l'équation suivante:

$$\begin{aligned}E[g^2]_t &= 0.9E[g^2]_{t-1} + 0.1g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}}g_t\end{aligned}$$

3.6.3.12 Méthode adaptative: Adam

Adam[56] est une approche adaptative qui exploite la moyenne des gradients au carré (comme Adadelta et RMSprop) mais aussi la moyenne des gradients. Ainsi, il réalise une estimation de la moyenne (1er moment) et de la variance non centrée du gradient (2nd moment). L'estimation respecte une approche dépréciative selon la position du gradient antécédent.

Adam est défini par:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

β_1 et β_2 sont des hyperparamètres et déterminent le pas dépréciatif des gradients antécédents.

Les deux moyennes sont initialisées à 0. De ce fait, leurs estimations sont biaisées (notamment durant les premières itérations). Afin de corriger les estimations, les valeurs non biaisées sont calculées selon:

$$\begin{aligned}\hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t}\end{aligned}$$

La mise à jour du paramètre est appliquée selon:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

ϵ apporte une solution au cas insoluble en $t = 0$ qui impose une division par 0, ce qui n'est pas réalisable. α est un hyperparamètre comparable au pas d'apprentissage (régulé par le comportement des gradients précédents). Les créateurs de cet optimizer estime qu'une bonne initialisation par défaut pour les différents hyperparamètres serait: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ et $\alpha = 0.001$.

Remarque: Il est possible de réécrire l'équation de manière plus efficiente. En effet, il est équivalent de définir l'équation de mise à jour telle que:

$$\begin{aligned}\alpha_t &= \alpha \frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t} \\ \theta_t &= \theta_{t-1} - \frac{\alpha_t m_t}{\sqrt{v_t} + \epsilon}\end{aligned}$$

Lors de la mise à jour des poids, la dernière itération est souvent bruitée à cause des approximations stochastiques. Afin d'obtenir une meilleure généralisation, un *moyennage* est souvent réalisé. Il a été montré que SGD converge mieux avec $\bar{\theta}_t = \frac{1}{t} \sum_{k=1}^n \theta_k$. Dans le cadre de Adam, une alternative basée sur une moyenne dépréciative (nommée *Temporal Averaging*) peut être appliquée telle que:

$$\bar{\theta}_t = \beta_2 \bar{\theta}_{t-1} + (1 - \beta_2) \theta_t$$

Nous appliquons $\bar{\theta}_0 = 0$. Cette initialisation est biaisée. L'estimation non biaisée est définie par: $\hat{\theta}_t = \frac{\bar{\theta}_t}{(1 - \beta_2^t)}$.

Aujourd'hui, cet optimizer fait office de choix par défaut lorsque aucun autre optimizer présente un intérêt particulier ou qu'aucune étude n'a été faite pour définir le meilleur choix à réaliser. Bien expérimentalement rapide à converger, il est souvent critiqué pour être d'une grande inefficacité dans le cadre de certaines thématiques ou d'être trop sensible à la problématique du minimum local.

3.6.3.13 Méthode adaptative: AdaMax

Dans la cadre d'Adam, le facteur v_t est inversement proportionnel à la norme l_2 des gradients précédents (via la valeur v_{t-1}). AdaMax[56] est une variante de Adam qui généralise en utilisant la norme l_p . Nous avons donc:

$$v_t = \beta_2^p v_{t-1} + (1 - \beta_2^p) |g_t|^p$$

Lorsque p augmente, les normes tendent à devenir numériquement instables d'où l'exploitation de la norme l_1 et l_2 . Cependant, la norme l_∞ présente une stabilité exploitable en plus d'une facilité d'implémentation.

Ainsi, nous obtenons:

$$u_t = \lim_{p \rightarrow \infty} (v_t)^{1/p} = \beta_2^\infty v_{t-1} + (1 - \beta_2^\infty) |g_t|^\infty$$

$$u_t = \max(\beta_2 \cdot v_{t-1}, |g_t|)$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{u_t} \hat{m}_t$$

Remarque: l'équation précédente peut être réécrite telle que:

$$\theta_t = \theta_{t-1} - \frac{\alpha}{(1 - \beta_1^t) u_t} m_t$$

De même, le correctif *Temporal Averaging*⁵⁰ est applicable pour cet algorithme. Les créateurs de cet optimizer estime qu'une bonne initialisation par défaut pour les différents hyperparamètres serait: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ et $\alpha = 0.002$.

3.6.3.14 Méthode adaptative: Nadam

Nadam[17] est une variante de Adam qui exploite *Nesterov Accelerated gradient* (NAG) au lieu de *Vanilla Momentum*.

Pour rappel, *Vanilla Momentum* est défini par:

$$g_t = \nabla_{\theta_t} J(\theta_t)$$

$$m_t = \gamma m_{t-1} + \alpha g_t$$

$$\theta_t = \theta_{t-1} - m_t$$

NAG réalise une estimation plus précise en évaluant la valeur du gradient en considérant le moment du gradient et non uniquement θ_t . Ainsi, NAG est défini par:

$$g_t = \nabla_{\theta_{t-1}} J(\theta_{t-1} - \gamma m_{t-1})$$

⁵⁰Voir algorithme Adam.

$$m_t = \gamma m_{t-1} + \alpha g_t$$

$$\theta_t = \theta_{t-1} - m_t$$

On peut observer que le moment est exploité **deux fois**: lors de la mise à jour du gradient et lors du calcul de θ_t . Afin de simplifier l'implémentation de Nadam, une alternative a été proposée. Elle permet de calculer le moment de l'itération $t+1$ à l'itération t:

$$g_t = \nabla_{\theta_{t-1}} J(\theta_{t-1})$$

$$m_t = \gamma m_{t-1} + \alpha g_t$$

$$\theta_t = \theta_{t-1} - (\gamma_{t+1} m_t + \alpha_t g_t)$$

Pour rappel, Adam est défini par:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\theta_t = \theta_{t-1} - \frac{\alpha_t}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

En réécrivant l'approche *Vanilla Momentum* de Adam par remplacement des variables, nous obtenons l'équation suivante (avec application du correctif de biais):

$$\theta_t = \theta_{t-1} - \frac{\alpha_t}{\sqrt{\hat{v}_t} + \epsilon} \left(\frac{\beta_{1,t} m_{t-1}}{1 - \beta_1^t} + \frac{(1 - \beta_{1,t}) g_t}{1 - \beta_1^t} \right)$$

En appliquant l'approche NAG, nous avons dorénavant:

$$\theta_t = \theta_{t-1} - \frac{\alpha_t}{\sqrt{\hat{v}_t} + \epsilon} \left(\frac{\beta_{1,t+1} m_t}{1 - \beta_1^{t+1}} + \frac{(1 - \beta_{1,t}) g_t}{1 - \beta_1^t} \right)$$

Cette approche est applicable aux autres méthodes adaptatives comme Adamax par exemple.

3.6.3.15 Adam et correctifs théoriques - AdamW

Pour favoriser une bonne convergence, différentes méthodes de régulation sont exploitées. La plus répandue est la régularisation par la norme L_2 (Voir la partie 5.3.2 pour plus d'informations). Pour rappel, cette régularisation s'applique sur la fonction de perte et est définie par:

$$\mathcal{L}_{L_2} = \mathcal{L} + \alpha \|\mathcal{L}\|_2^2$$

Une autre forme de régularisation existe et est très employée dans le cadre des optimizers. Elle se nomme *Weight Decay*[31]. Contrairement à l'approche par norme (qui agit sur la fonction de perte), *Weight Decay* agit directement sur la

valeur du paramètre lors de sa mise à jour en imposant une minoration. Ainsi, lors d'une mise à jour, le paramètre θ sera défini par:

$$\theta_{t+1} = (1 - \lambda)\theta_t - \alpha \nabla \mathcal{L}(\theta_t)$$

Dans les faits, ces deux méthodes sont similaires. A tel point qu'elles sont équivalentes dans le cadre de l'optimizer SGD.

Preuve:

Supposons une approche SGD régularisée par la norme L_2 telle que:

$$\mathcal{L}_{L_2} = \mathcal{L} + \frac{\lambda'}{2} \|\theta\|_2^2$$

Lors de la mise à jour d'un poids, nous avons donc:

$$\theta_{t+1} = \theta_t - \alpha \nabla \mathcal{L}_{L_2}(\theta_t) = \theta_t - \alpha \nabla \mathcal{L}(\theta_t) - \alpha \lambda' \theta_t$$

Supposons SGD avec l'approche *Weight Decay*:

$$\theta_{t+1} = (1 - \lambda)\theta_t - \alpha \nabla \mathcal{L}(\theta_t)$$

Les deux approches sont identiques si $\lambda' = \frac{\lambda}{\alpha}$

Du fait de cette équivalence, la régularisation par norme est souvent considérée comme *équivalente* à la régulation par Weight Decay. Or, ce n'est **pas vrai**. Notamment dans le cas des méthodes adaptatives telles que Adam.

Dans de nombreuses implémentations de l'algorithme Adam, appliquer Weight Decay est identique à utiliser la régularisation L_2 . En d'autres mots, lors de l'utilisation de L_2 /Weight Decay sur Adam, la valeur g_t devient égale à⁵¹:

$$g_{t+1, reg} = \nabla \mathcal{L}(\theta_t) + \lambda \theta_t$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Afin de corriger cette erreur, AdamW[73] a été proposé et implémenté correctement Weight Decay. Ainsi, avec AdamW, nous obtenons:

$$g_{t+1} = \nabla \mathcal{L}(\theta_t)$$

$$\theta_{t+1, reg} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t + \lambda \theta_t$$

Définir le facteur de Weight Decay est difficile. Néanmoins, il a été montré qu'une petite taille de batch accentue l'effet du Weight Decay. Afin de limiter

⁵¹Vous référer à la partie sur Adam pour le détail de l'optimizer Adam (et vous remémorer les notations !)

le risque lié à cette dépendance, il est préférable de normaliser λ afin de limiter le risque de biais. Pour cela, il a été proposé:

$$\lambda_{norm} = \lambda * \sqrt{\frac{b}{BT}}$$

Avec b , dimension du batch, B , nombre totale de données d'apprentissage, T , nombre d'epochs. Cette normalisation est instinctive. Il est probable qu'il soit possible de proposer une amélioration notable à cette approche.

Bien que simple, AdamW permet une amélioration notable des performances de l'optimizer Adam. De plus, il est suspecté que SGD soit meilleure que Adam sur certaine problématique du fait de cette erreur théorique de généralisation de la norme L_2 .

3.6.3.16 Adam et correctifs théoriques - AMSGrad

Les algorithmes adaptatifs basées sur une moyenne pondérée présente une faiblesse théorique. Supposons la valeur Γ_{t+1} tel que:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\Gamma_{t+1} = \frac{\sqrt{v_{t+1}}}{\alpha_{t+1}} - \frac{\sqrt{v_t}}{\alpha_t}$$

Afin de limiter le risque de convergence indésirable, il est nécessaire que Γ_t soit défini positif, i.e $\Gamma_t \succeq 0$ pour tout t . Or, ce n'est pas le cas avec Adam⁵².

Cette observation peut laisser penser que Adagrad est une approche, qui, au final, est préférable. Ce n'est pas le cas car la diminution du pas est trop agressive avec cette approche. Il est donc nécessaire de proposer une alternative qui respecte la condition sur Γ_t tout en limitant l'agressivité de la diminution imposée.

AMSGrad[86] est une variante de Adam et respecte la condition sur Γ_t . Au lieu d'exploiter la moyenne pondérée des gradients passés v_t , il utilise le *maximum* des gradients passés, i.e $\hat{v}_t = \max(\hat{v}_{t-1}, v_t)$. De plus, β_1 est variable et décroissant (par exemple $\beta_{1,t} = \beta_1/t$).

Remarque: Dans l'article de référence, l'utilisation de β_1 est floue. Le paramètre est considéré comme fixe dans les démonstrations mais variable dans la définition de l'algorithme. Dans les faits, il semblerait que le comportement du coefficient β_1 soit peu important pour l'efficacité globale de AMSGrad. Néanmoins, une valeur décroissante semble avoir une démonstration théorique confirmée du fait d'articles complémentaires supposant cette condition.

⁵²Consultez l'article [86] pour la démonstration mathématique de ce résultat.

Supposons le cas où $v_{t-1} > g_t^2 > 0$. Adam va provoquer une augmentation agressive du pas d'apprentissage (diminution de v_t). Au contraire, Adagrad va diminuer le pas d'apprentissage (augmentation de v_t) car il exploite une somme non pondérée. Par opposition, AMSGrad ne va pas modifier le pas d'apprentissage, assurant une meilleure stabilité.

Une explication intuitive peut être donnée à ce résultat pouvant sembler contre-intuitif. Il a été observé que des minibatchs porteurs d'informations utiles sont existants mais rares. De ce fait, leur influence est diminuée car écrasée par la moyenne pondérée. L'utilisation d'une moyenne pondérée provoque un phénomène de mémoire à *court terme* qui limite l'impact des minibatchs utiles, ce qui est désastreux dans le cadre de certaines problématiques.

Ainsi, AMSGrad est défini par:

$$\begin{aligned}\beta_1 &= \beta_{1,1} \\ \beta_{1,t} &= \beta_1 \lambda^{t-1} \\ m_t &= \beta_{1,t} m_{t-1} + (1 - \beta_{1,t}) g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \\ \hat{v}_t &= \max(\hat{v}_{t-1}, v_t) \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} m_t\end{aligned}$$

Une autre variante a aussi été proposée pour satisfaire la condition Γ_t . Nommée ADAMNC, elle est strictement identique à Adam si ce n'est que β_2 est **non constant**. Nous avons donc:

$$\begin{aligned}\beta_1 &= \beta_{1,1} \\ \beta_{1,t} &= \beta_1 \lambda^{t-1} \\ \beta_{2,t} &= 1 - 1/t\end{aligned}$$

Cette approche permet de pondérer différemment l'ensemble des gradients passés. Dans le cadre de ADAMNC, la pondération est constante (si β_2 est défini comme précédemment), i.e $v_{t,i} = \sum_{j=1}^t g_{j,i}^2 / t$.

3.6.3.17 Adam et correctifs théoriques - Nostalgic Adam

Nostalgic Adam[43] (NosAdam) propose une autre approche pour exploiter les gradients passés afin de calculer le pas d'apprentissage. La particularité est que les gradients anciens ont un poids plus importants que les gradients récents. Pour cela, il est nécessaire de définir des conditions particulières sur la détermination de $\beta_{2,t}$. NosAdam respecte la condition de semi-positivité de Γ_t . Elle a donc une solidité théorique avérée.

Ainsi, NosAdam est défini par:

$$\begin{aligned} B_t &= \sum_{k=1}^t b_k, \quad b_k > 0, \quad t \geq 1, \quad B_0 = 0 \\ \beta_{2,t} &= B_{t-1}/B_t \\ m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t &= \beta_{2,t} v_{t-1} + (1 - \beta_{2,t}) g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} m_t \end{aligned}$$

Remarque: La détermination de β_1 est imprécise dans l'article de recherche. Il est variable dans le cadre des démonstrations théoriques mais l'écriture de l'algorithme sous-entend qu'il est constant. Une certaine tolérance doit être envisageable sur ce paramètre...

La condition $\beta_{2,t} = B_{t-1}/B_t$ est une écriture générale. Il a été démontré que la condition de semi-positivité est vérifiée si B_t/t n'est pas croissant, i.e b_k non croissant et $b_k > 0$. Elle est notamment avérée lorsque $\beta_{2,t}$ est indépendant des données et uniquement lié à t .

Supposons $\beta_{2,t} = B_{t-1}/B_t$ et $B_t = \sum_{k=1}^t b_k$. De ce fait, nous avons:

$$v_t = \sum_{k=1}^t \frac{b_k}{B_t} g_k^2$$

b_k est le poids relatif de g_k^2 . Or, b_k est non croissant donc les gradients anciens sont majorés par rapport aux récents. Cette approche est l'opposé de la méthode par moyenne pondérée qui majore les gradients récents.

Il est intéressant de noter que les créateurs de NosAdam considère ADAMNC comme un cas particulier de NosAdam. En effet, ADAMNC est défini pour $B_{2,t} = 1 - 1/t$. De ce fait, $v_t = \sum_{k=1}^t \frac{g_k^2}{t}$. On observe qu'il n'y a plus de pondération relative donc que chaque gradient est pondéré identiquement. De même, AMSGrad, à travers l'opérateur $\max(\cdot)$, conserve une mémoire *longue-durée* des gradients passés et peut réaliser une pondération importante d'un gradient passé grâce aux spécificités de $\max(\cdot)$ ⁵³.

NosAdam définit une classe d'optimizers caractérisée par des conditions sur β_2 . Un cas particulier de NosAdam a été proposé: NosAdam-HH. Cette approche exploite une série *hyper-harmonique* pour définir b_k tel que $b_k = k^{-\lambda}$ avec $\lambda \geq 0$.

⁵³Il y a une dépendance vis-à-vis de g_k^2 . En effet, il n'y a pas mémoire du passé dans sa globalité mais que de l'événement principal caractérisé par $\max(\cdot)$. Il s'agit donc d'une mémoire long-terme sélective en plus d'être exclusive.

3.6.3.18 Méthode adaptative cyclique - AdamWR

Les optimizers cycliques présentent une efficacité remarquable grâce à leur capacité à s'extraire des minimums locaux peu performants. AdamWR[73] propose d'unir AdamW avec *Cosinus Annealing and Warm Restarts*.

AdamW modifie Adam tel que:

$$\theta_{t+1,reg} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t + \lambda \theta_t$$

Dans le cadre de AdamWR, nous avons dorénavant:

$$\begin{aligned}\theta_{t+1,reg} &= \theta_t - \alpha_t \left(\frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t + \lambda \theta_t \right) \\ \alpha_t &= \alpha_{min}^{(i)} + 0.5(\alpha_{max}^{(i)} - \alpha_{min}^{(i)})(1 + \cos(\frac{\pi T_{curr}}{T_i}))\end{aligned}$$

Avec $\alpha_{min}^{(i)}$ et $\alpha_{max}^{(i)}$, valeur minimale et maximale de α durant l'i-ième restart. Pour limiter le nombre d'hyperparamètre, il est commun de considérer ces valeurs comme fixes mais il a été montré que proposer une ajustation à chaque restart pourrait potentiellement améliorer les performances. T_{curr} correspond au nombre d'itérations depuis le dernier restart et T_i , le nombre total d'itérations durant le i-ième restart.

Afin d'améliorer cet algorithme, il est préférable d'utiliser une valeur T_i variable selon le nombre de restarts réalisés. Intuitivement, il est préférable d'avoir des restarts réguliers au début de l'apprentissage mais plus rare au fil du temps afin de favoriser une convergence. De ce fait, il est préférable que T_i soit croissant au fil des itérations. Cette pratique est standard dans le cadre des optimizers cycliques. De même, il est nécessaire d'utiliser une valeur *normalisée* du facteur de Weight Decay⁵⁴ (dans le cadre du Warm Restarts, nous considérerons T comme nombre d'itération dans le restart actuel).

AdamWR présente une efficacité globalement similaire à AdamW mais sa vitesse de convergence est significativement meilleure. AdamWR rivalise avec SGDWR en terme de performance, ce qui illustre l'hypothèse que les mauvaises performances de Adam sur certains problèmes pourraient être liées à la confusion entre L_2 /Weight Decay.

3.6.3.19 Discriminative Fine-Tuning

Discriminative Fine-Tuning[37] est une approche complémentaire aux méthodes vues précédemment qui évalue le pas d'apprentissage selon le comportement vis-a-vis des données⁵⁵.

⁵⁴Voir AdamW pour plus d'informations sur cette normalisation.

⁵⁵Le gradient est directement lié aux données d'apprentissage.

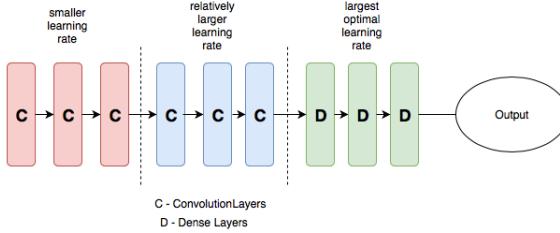


Figure 18: Discriminative Fine-Tuning sur un réseau CNN standard

Les méthodes précédentes considèrent que chaque couche exploite un même pas d'apprentissage pour mettre à jour ses poids. Or, ce postulat est contestable. En effet, les couches les plus basses (les premières couches du réseau) discriminent des phénomènes généralistes⁵⁶. De ce fait, ces couches sont faiblement spécialisées car non liées à la tâche à discriminer. Par exemple, supposons le cas de l'analyse d'image. Peu importe le type d'objet à discriminer sur les images, son analyse exploitera toujours des structures géométriques simples et récurrentes car chaque phénomène aussi complexe soit-il repose sur des structures simples communes. Ainsi, le comportement discriminant est réalisé essentiellement par les couches les plus hautes (les dernières couches du réseau) qui analysent des structures plus complexes et donc, spécialisées.

Cette observation conditionne l'idée que seule les dernières couches possèdent un vrai pouvoir discriminant. De ce fait, il est nécessaire que les modifications induites par l'exploitation du gradient soit plus importantes pour les couches hautes que les couches basses. En effet, le gradient étant lié aux données et donc au phénomènes observés, il a donc un lien plus fort avec les couches hautes. Ainsi, Discriminative Fine-Tuning propose de modifier le pas d'apprentissage selon la position de la couche mise-à-jour. Les couches les plus hautes auront un pas majoré et les couches les plus basses, un pas minoré. Une illustration est visible sur la Figure⁵⁷ 18.

Plus formellement, supposons un réseau avec L couches. Nous définissons $\theta = \{\theta_0, \dots, \theta_l, \dots, \theta_L\}$, les paramètres du réseau avec θ_l , paramètres de la l -ième couche. Alors:

$$\theta_t^l = \theta_{t-1}^l - \eta^l \nabla_{\theta^l} J(\theta)$$

Avec $\eta^{l-1} = \eta^l / 2.6$ avec $\eta^L = \eta_{max}$ tel que η_{max} est la valeur du pas obtenu par l'optimizer choisi (par exemple, Adam). Le facteur 2.6 est un résultat obtenu empiriquement. Il est donc utile de considérer une étude approfondie du meilleur

⁵⁶Par exemple, dans le cadre de l'analyse d'une image, les premières couches discrimineront des structures géométriques simples comme des droites, courbes, cercles etc...

⁵⁷Les spécificités d'une architecture CNN seront développées dans la suite de cette introduction.

facteur possible selon le type de tâche à accomplir.

Remarque: Dans les faits, cette méthode n'est pas utilisée lorsque l'apprentissage est *from-scratch*⁵⁸. En effet, dans cette situation, les poids des couches basses sont aléatoires. De ce fait, les couches basses sont incapables de discriminer un quelconque phénomène. Il est donc nécessaire de les entraîner comme les couches plus hautes.

Au contraire, l'idée de pondérer le pas d'apprentissage est très exploitée lors d'un *apprentissage par transfert*. En effet, dans cette configuration, le réseau initial n'est pas *from scratch* mais déjà entraîné sur une tâche. Le comportement des couches basses n'est donc plus aléatoire mais apte à une discrimination. Ce cas particulier sera étudié dans la suite de cette introduction.

3.6.3.20 Et les méthodes d'optimisation du 2nd ordre ? - A FAIRE

3.6.3.21 SGD, calcul distribué et parallélisation - A FAIRE

3.6.3.22 Inférence de l'optimizer - Neural Optimization Search

PowerSign and AddSign [7]

3.6.3.23 Inférence de l'optimizer - LSTM approach

LSTM-Learning[2]

3.6.3.24 Quel optimizer choisir ?

C'est toujours une question **sans réponse !** La liste présentée est non-exhaustive et l'étude des optimizer est encore un sujet de recherche actuel. Il n'existe pas de hiérarchie de performance clairement établie. Seule l'approche empirique est exploitée aujourd'hui. Néanmoins, les méthodes avec un *learning rate* adaptatif semblent être les plus répandues, notamment Adam malgré des performances parfois très mauvaises. L'état de l'art tend à exploiter des approches cycliques bien qu'elles ne soient pas encore pleinement démocratisées. Néanmoins, il n'est pas rare d'observer l'emploi de SGD (avec Momentum) dans des publications actuelles, notamment pour des tâches de traduction (analyse de texte).

Important: A l'heure actuelle, la recherche est encore très manuelle et intuitive, i.e l'homme propose et expérimente lui-même des concepts qu'il invente. Néanmoins, des travaux récents exploitent l'intelligence artificielle pour apprendre à découvrir les concepts performants. Un aperçu a été proposé avec l'inférence de l'optimizer mais cette pratique se généralise avec l'auto-apprentissage de l'architecture d'un réseau dans sa globalité. Cette approche est encore expérimentale mais promet d'être une semi-révolution dans l'exploitation du Deep

⁵⁸Un apprentissage from-scratch est un apprentissage à partir de rien, i.e une initialisation aléatoire des poids du réseau.

Learning et de sa Recherche de part la facilité d'innovation et d'exploitation (pour l'industrie) qui en découle.

3.6.4 Gradient noise

Afin de favoriser la robustesse du réseau face à des stimulations ambiguës ou inhabituelles, il est intéressant de *bruiter*[4] la valeur du gradient durant l'apprentissage afin de renforcer le réseau. Cette approche consiste à ajouter un bruit gaussien (distribution gaussienne centrée en 0 et écart-type variable) à la valeur du gradient calculé à chaque rétropropagation. Ainsi, supposons $g_{i,t}$, le gradient obtenu pour le poids i à l'instant t, alors $g_{i,t,gauss} = g_{i,t} + \mathcal{N}(0, \sigma^2)$ avec $\sigma_t^2 = \frac{\eta}{(1+t)^\gamma}$ et $\eta \in [0.01, 0.3, 1.0]$.

Ainsi, le bruit sera plus important en début d'apprentissage pour forcer le réseau à ne pas converger trop rapidement vers un minimum local⁵⁹. Cette méthode serait performante pour les modèles très profonds, limiterait l'impact d'une mauvaise initialisation des poids du réseau et favoriserait "l'échappement" des minimum locaux (qui sont de plus en plus nombreux alors que le modèle s'approfondit). Cette méthode, bien qu'élégante, n'est que peu employée par la recherche et son efficacité incertaine.

3.6.5 Gradient Clipping

Afin de limiter le risque d'explosion du gradient (phénomène nommé *exploding gradient*⁶⁰), il est pertinent de borner la valeur du gradient en normalisant sa valeur absolue lorsqu'elle dépasse un seuil défini. Cette méthode est appelée *Gradient Clipping*[85]. Il existe plusieurs manières de réaliser cette normalisation bien que la majorité des implémentations se limitent à l'utilisation d'une valeur seuil, i.e $|g_{i,t}| > \lambda \Rightarrow |g_{i,t}| = \lambda$. Cette approche est reconnue comme efficace au fil des publications de recherche.

3.7 ReLu et les dangers du gradient

Le gradient est l'élément central pour l'apprentissage d'un réseau de neurones. Cependant, trois dangers majeurs sont à considérer pour garantir l'intégrité des gradients: **Exploding Gradient**, **Vanishing Gradient** et **Dead ReLu**.

Un réseau de neurone peut avoir des centaines (ou plus) couches associées à des milliers (ou plus) neurones. De ce fait, le modèle peut être très profond. Nous avons vu que l'apprentissage se base sur la rétropropagation du gradient. Cette rétropropagation est un produit de facteur. Ainsi, plus le poids à mettre à jour est éloignée de la sortie du réseau (couches basses du réseau), plus le degrés du produit est élevé. Cette particularité peut poser problème selon

⁵⁹Favoriser l'exploration des solutions possibles

⁶⁰Voir Section 3.7.2

l'architecture du réseau (notamment la profondeur) ou les fonctions de transfert dont la dérivée est comprise dans $[0, 1[$ ou strictement supérieure à 1 sur un intervalle quelconque.

3.7.1 Vanishing Gradient

Dans le cas où la valeur des dérivées partielles des couches intermédiaires est comprise dans $[0, 1[, il y a un risque de *vanishing gradient*. En effet, pour n dans $[0, 1[, \prod_{k=1}^{\infty} \alpha_n \xrightarrow{\infty} 0$. Cette particularité peut figer le réseau et annuler la mise à jour de neurones éloignés car la mise à jour du poids est infinitésimale.$

3.7.2 Exploding Gradient

Au contraire, pour des valeurs comprises dans $]1, \infty[$, il y a un risque d'*exploding gradient*. Pour n dans $]1, \infty[$, $\prod_{k=1}^{\infty} \alpha_n \xrightarrow{\infty}$. Ainsi, la valeur du gradient "explose" et peut empêcher une bonne convergence du modèle (évolution du poids trop importante) voire annuler l'apprentissage du réseau avec des valeurs dites NaN (Not A Number) car dépassant les limites matérielles de la représentation des nombres par ordinateur.

3.7.3 Fonction ReLU et Dead ReLU

3.7.3.1 Les avantages et le danger de ReLU

Pour corriger cette difficulté, une fonction d'activation est souvent utilisée pour les couches cachées des réseaux: la fonction ReLU.

$$\text{La fonction ReLU est définie par: } f(x) = \begin{cases} 0 & \text{si } x < 0 \\ \max(0, x) & \text{si } x \geq 0 \end{cases}$$

Sa dérivée est nulle sur $]\infty, 0]$ et égale à 1 sur $]0, \infty[$. De part les valeurs de sa dérivée, le risque d'*exploding gradient* est annulé. Néanmoins, la présence d'une dérivée nulle peut laisser penser que le risque de *vanishing gradient* est présent. Bien que vrai, ce défaut est compensé par la capacité de ReLU à rendre le réseau éparsé.

Le risque de sur-apprentissage⁶¹ est présent lorsque les neurones deviennent trop inter-dépendants, que ce soit durant la prédiction ou l'apprentissage. L'idée principale, soutenue par l'analogie biologique du cerveau, est que le réseau doit être localement stimulé pour répondre à une entrée. Ainsi, selon l'entrée, l'ensemble du réseau ne doit pas être activé mais seulement une sous-partie capable d'interpréter cette stimulation. La fonction ReLU, nulle pour tout x négatif, permet de simuler ce comportement. De plus, la dérivée de ReLU, semblable à une fonction *Porte*, permet de réaliser des mises à jour éparses du réseau. La présence d'une dérivé nulle (pour x négatif) permet de limiter localement la mise à jour des poids, rendant les évolutions du système localisées et

⁶¹Voir Section 5.1

moins inter-dépendantes. En cas de stimulation positive, la dérivée étant de 1, l'influence sur la valeur du gradient est nulle. Cette fonction d'activation permet ainsi d'éteindre des neurones durant la phase de prédiction et force le réseau à avoir une stimulation localisée pour réaliser la prédiction. Durant la rétro-propagation, son comportement orchestre les régions du réseau qui apprennent et celles qui dorment. Ainsi, cette fonction permet d'agir sur l'architecture du réseau pour limiter les problèmes de corrélation néfastes entre les neurones qui favorisent le sur-apprentissage. De plus, cette limitation des neurones employés permet de limiter le coût de calcul et ainsi, d'augmenter la vitesse du réseau (en prédiction et apprentissage).

Ces spécificités ont popularisé la fonction ReLu qui est, aujourd'hui, la fonction de référence pour les couches cachées des réseaux neuronaux. Elle est peu employée pour la couche de sortie car elle ne présente que peu de pouvoir explicatif. Les fonction sigmoïde/softmax sont préférées pour leur représentation probabiliste ou encore la fonction linéaire standard pour une représentation sans norme spécifique.

Néanmoins, cette fonction est sensible au phénomène appelé *Dead ReLu*. En effet, la dérivée (localement) nulle de la fonction ReLu présente des caractéristiques dangereuses. En effet, dans le cas d'initialisation du réseau avec des poids mal calibrés, il est possible que la stimulation soit fortement négative (peu importe la données d'apprentissage), provocant une sortie nulle permanente du neurone. La dérivée étant nulle, le neurone n'apprend pas (gradient nul) et de ce fait, la sortie ne pourra jamais être autre que nulle durant tout l'apprentissage. Ce comportement est associé à un "neurone mort": le neurone n'apprend pas et retourne une sortie nulle en tout temps. Cette particularité peut entraîner l'inactivité d'une partie plus ou moins importante du réseau et nuire à son efficacité voire le condamner. Afin de lutter contre ce problème, l'utilisation de *Régularisation*⁶² en plus d'une bonne initialisation⁶³) sont employées.

Afin de lutter contre ce phénomène, des améliorations ont été proposées pour la fonction ReLu (un graphique récapitulatif est visible sur la Figure 19):

3.7.3.2 Leaky ReLU (LReLU)

LReLU[123] est définie par: $f(x) = \begin{cases} \alpha x & \text{si } x < 0 \\ \max(0, x) & \text{si } x \geq 0 \end{cases}$ avec α petit (souvent initié à 0.01).

Cette variante permet de conserver la capacité d'apprentissage du neurone en supprimant la possibilité de gradient nul. Néanmoins, la valeur de la dérivée reste très faible (égale à α), ce qui peut demander un temps d'apprentissage

⁶²Voir Section 5.3

⁶³Voir Section 3.8.1

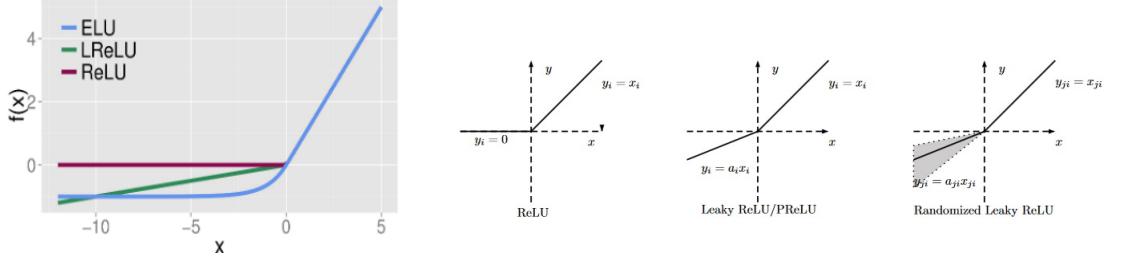


Figure 19: Fonction ReLU et ses Variantes

important pour ré-activer le neurone. De plus, elle demande la détermination d'un nouveau hyperparamètre.

3.7.3.3 Randomized ReLU (RReLU)

RReLU[123] est définie par: $f(x) = \begin{cases} \alpha^{(\mathcal{U}_i)} x & \text{si } x < 0 \\ \max(0, x) & \text{si } x \geq 0 \end{cases}$ avec $\alpha^{(\mathcal{U}_i)}$, valeur aléatoire issue d'une distribution uniforme $\mathcal{U}(m, n)$ avec $m < n$ et $[m, n] \in [0, 1[$.

Lors de l'apprentissage, $\alpha^{(\mathcal{U}_i)}$ varie à chaque itérations. Lors du l'évaluation (test) du modèle ou de prédiction, la valeur de $\alpha^{(\mathcal{U}_i)}$ est fixé. L'objectif de l'aléatoire est de diminuer le risque de sur-apprentissage du réseau.

3.7.3.4 Parameterized ReLU (PReLU)

PReLU[35][123] est semblable à Leaky ReLU mais le coefficient α est calculé dynamiquement par backpropagation. Ainsi, α n'est plus un hyperparamètre mais demande un coût de calcul supérieur (très négligeable). PReLU est définie par:

$$f(x_i) = \begin{cases} \alpha_i x_i & \text{si } x_i < 0 \\ \max(0, x_i) & \text{si } x_i \geq 0 \end{cases}$$

Le paramètre α_i peut être appris selon deux approches:

- **Local:** L'approche locale exploite un coefficient α_i pour chaque channel de la couche correspondante. Il y aura donc autant de paramètres appris que de *feature map* en sortie de la couche.
- **Global:** L'approche globale exploite un coefficient α unique pour l'ensemble des channels de la couche, ce qui diminue le nombre de paramètres à apprendre durant l'apprentissage.

Expérimentalement, les deux approches ont des résultats similaires même si l'approche *locale* est légèrement plus performante. Pour les deux approches,

le coût computationnel est négligeable car $nbr_{poids} \gg nbr_{\alpha_i}$. Néanmoins, il serait intéressant d'étudier l'impact des deux configurations vis-à-vis du sur-apprentissage et de la capacité de généralisation du modèle.

Le paramètre α_i est entraîné par backpropagation. Ainsi, la mise à jour de sa valeur est dépendante de son gradient. Dans le cadre de l'approche locale, nous avons:

$$\frac{\partial \varepsilon}{\partial \alpha_i} = \sum_{x_i} \frac{\partial \varepsilon}{\partial f(x_i)} \frac{\partial f(x_i)}{\partial \alpha_i}$$

$$\frac{\partial f(x_i)}{\partial \alpha_i} = \begin{cases} x_i & \text{si } x_i < 0 \\ 0 & \text{si } x_i \geq 0 \end{cases}$$

Avec ε représentant la fonction de coût. La somme \sum_{x_i} est appliquée car on exploite l'erreur selon l'ensemble des positions de la *feature map*⁶⁴.

Dans le cadre de l'approche globale, nous obtenons:

$$\frac{\partial \varepsilon}{\partial \alpha} = \sum_i \sum_{x_i} \frac{\partial \varepsilon}{\partial f(x_i)} \frac{\partial f(x_i)}{\partial \alpha}$$

L'erreur est ainsi généralisée sur l'ensemble des channels et sommées pour en extraire sa valeur.

La mise à jour du paramètre suit une approche *momentum* et est caractérisée par:

$$\alpha_i^{k+1} = \beta \alpha_i^k + \epsilon \frac{\partial \varepsilon}{\partial \alpha_i^k}$$

Avec β , coefficient du *momentum* et ϵ , pas d'apprentissage.

Il est intéressant de noter qu'aucune régularisation n'a été appliquée (notamment l_2) car elle a tendance à forcer α_i à tendre vers 0, ce qui rend PReLU comparable à Relu. De plus, sa valeur est non bornée. Néanmoins, expérimentalement, il semble qu'il n'y ait pas de phénomène d'*explosion* de la valeur du coefficient⁶⁵. Par défaut, l'initialisation de α_i est à 0.25.

3.7.3.5 Exponential ReLU (ELU)

ELU[13] est définie par: $f(x) = \begin{cases} \alpha(\exp(x) - 1) & \text{si } x < 0 \\ \max(0, x) & \text{si } x \geq 0 \end{cases}$

Cette fonction permet de borner la valeur d'activation pour $x < 0$, augmentant ainsi sa résistance au bruit. Le facteur α est un hyperparamètre fixé.

⁶⁴Ne pas oublier le comportement de fenêtre glissante dans les réseaux convolutifs. Ainsi, le même neurone est appliqué à chaque position de la feature map considérée, ce qui nécessite de sommer les résultats pour apprendre sur l'erreur globale de l'analyse de la feature map.

⁶⁵La valeur dépasse rarement 1 d'après les expérimentations des créateurs de PReLU.

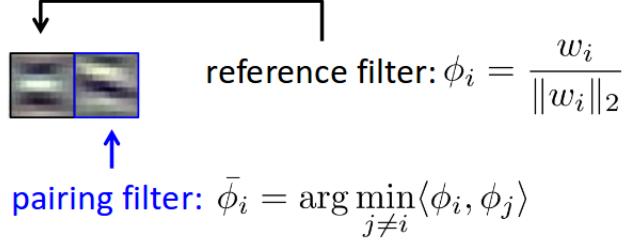


Figure 20: Exemple de deux filtres de phase opposée

3.7.3.6 Concatenated ReLU (CReLU)

Expérimentalement, il a été montré que les filtres des premières couches d'un réseau tendent à être redondants afin d'extraire l'information issue des *phases* positives et négatives d'un signal donné. Cette faiblesse oblige le réseau à exploiter d'autres filtres pour exploiter l'information perdue par l'application d'une fonction d'activation ReLU (perte de l'information de la phase négative). Par exemple, sur la Figure 20, nous pouvons observer un couple de filtres. Le comportement des filtres est similaire mais opposé par la phase. Afin de limiter la redondance de filtre, Concatenated ReLU propose d'extraire l'information intégralement sans la perte imposée par ReLU en considérant aussi les valeurs négatives selon la même approche que ReLU possède avec les valeurs positives.

CReLU[98] est ainsi définie par: $f(x) = (\max(0, x), \max(0, -x))$.

Elle est très similaire à Absolute Value Rectification (AVR) mais au lieu d'additionner les deux sorties intermédiaires, CReLU les concatène. De ce fait, la sortie de cette fonction d'activation est de profondeur 2. Cette fonction d'activation permet ainsi de limiter le nombre de filtres nécessaires en optimisant l'extraction d'informations d'un signal donné.

3.7.3.7 Scaled Exponential Linear Units (SeLU)

:

La fonction est définie par: $f(x) = \lambda \begin{cases} \alpha(\exp(x) - 1) & \text{si } x < 0 \\ \max(0, x) & \text{si } x \geq 0 \end{cases}$

Attention: Cette partie nécessite la connaissance des fondamentaux d'architecture d'un réseau profond.

Cette fonction est une des composantes utilisée dans le cadre des Self-Normalizing Neural Networks[57] (SNN). Cette architecture propose une autre approche afin de résoudre la problématique de la normalisation des données dans les réseaux très profonds. Au lieu d'employer des méthodes de normalisation ex-

ternes⁶⁶ appliquées à la sortie d'une fonction d'activation, la sortie de la fonction d'activation fournit des valeurs déjà normalisées. Pour que cette spécificité soit réalisée, il est nécessaire d'employer des fonctions SELU initialisées selon la distribution $W \sim \mathcal{N}(0, \frac{1}{n_{in}})$. Cette distribution est comparable aux distributions standards (MSRA/Xavier par exemple) mais la variance n'exploite pas le facteur 2 qui neutralise les effets de la fonction d'activation. Pour plus de détails, notamment mathématiques, veuillez vous référer à l'article associé [57]⁶⁷.

3.8 Initialisation des hyperparamètres - A FAIRE

3.8.1 Initialisation des poids

L'initialisation des poids d'un réseau est un critère important à considérer. Une mauvaise initialisation peut provoquer la divergence de réseau, notamment dans le cas de réseau profond. Il est donc important de réaliser une initialisation limitant ce danger. L'objectif de l'initialisation est de faire en sorte que les sorties des neurones aient approximativement la même variance, de même que pour les valeurs des gradients obtenus par rétropropagation. Plusieurs approches ont été popularisées ces dernières années:

Basé sur la variance des sorties de neurones uniquement:

- **Calibration de la variance:** L'initialisation avec calibration de la variance revient à choisir aléatoirement une valeur dans une distribution normale définie par: $W \sim \mathcal{N}(0, \frac{1}{n_{in}})$ avec n_{in} , nombre d'entrées du neurone (ou nombre de neurones sur la couche précédente).
- **ReLU Calibration:** La fonction ReLu est la fonction d'activation la plus populaire et utilisée actuellement (pour les couches cachées). Son étude est encore un sujet de recherche important et dynamique. Une publication récente [48] préconise une initialisation telle que: $W \sim \mathcal{N}(0, \frac{2}{n_{in}})$ ou via une distribution uniforme: $W \sim U[-\sqrt{\frac{6}{n_{in}}}, \sqrt{\frac{6}{n_{in}}}]$.

La distribution normale est la distribution standard pour initier les poids associées à la fonction ReLu.

Basé sur un compromis entre la variance des sorties de neurones et des gradients:

- **Xavier initialization:** D'autres approches préconisent un compromis entre la variance des sorties des neurones et des gradients calculés par rétropropagation. La méthode préconisée devient ainsi une dépendante du nombre d'entrées et de sorties d'un neurone⁶⁸. La première approche

⁶⁶Comme le Batch Normalisation par exemple

⁶⁷C'est un article très mathématique et lourd à la lecture !

⁶⁸Lors de la rétropropagation, ce sont les sorties des neurones qui sont exploitées

correspond à une distribution normale: $W \sim \mathcal{N}\left(0, \frac{2}{n_{in}+n_{out}}\right)$ et la seconde, une distribution uniforme: $\mathbb{W} \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right]$.

Important: Par défaut, il est **important** d'éviter les initialisations aléatoires sur un intervalle élevé, de même qu'initialiser à 0 l'ensemble des poids (sauf spécificité d'une méthode).

3.8.2 Initialisation et détermination des hyperparamètres - A FAIRE

3.9 Jeu d'apprentissage et spécificités

Une des plus grandes contraintes du Deep Learning est la création d'un jeu de données d'apprentissage⁶⁹ de *qualité*. La notion de *qualité* repose sur différents critères:

- **Spécialisation:** Un jeu de données doit se focaliser sur le phénomène qu'il représente. Par exemple, si l'on souhaite détecter des chats sur une image, il est évident que le jeu de données d'apprentissage devra contenir des chats...
- **Représentativité:** Un jeu de données doit être capable de représenter le phénomène sous toutes ses formes (tout du moins un maximum) afin de rendre la discrimination représentative de ses différentes hypothèses possibles. Supposons un jeu de données pour discriminer tout type de chats. Il est donc intéressant d'avoir des données variables selon:
 - **Spécificité du phénomène:** La première condition est de représenter un maximum de spécificité que peut posséder le phénomène. Par exemple, dans le cas d'un chat, il est intéressant de mêler des chats de différentes races, tailles, couleurs, posture et mouvement etc...
 - **Spécificité du contexte:** Le phénomène ne sera pas toujours représenté de manière centrée sur l'image tout en occupant la majorité de sa surface. C'est ainsi nécessaire de considérer des cas où le phénomène est localisé sur une sous-partie de l'image. Il est donc important d'avoir des images de l'environnement où se développe le contexte. Par exemple, en supposant le cas des chats, il est utile d'avoir des cas où la présence du chat est plus (ou moins) importante⁷⁰, une diversité d'environnement (un chat dans un appartement, dans la rue,...)
 - **Spécificité du capteur:** Selon le capteur, la nature de l'image peut varier indépendamment du phénomène et de son environnement. Un appareil haute définition ne donnera pas la même qualité d'image qu'une caméra standard. Il est donc nécessaire de considérer cette spécificité en exploitant des images issues de sources différentes.

⁶⁹On utilise souvent l'anglicisme *dataset*

⁷⁰La localisation des zones devrait être uniformément (idéalement) distribuée

- **Spécificité de détérioration:** Il est possible qu'une image soit détériorée (capteur défectueux, présence de bruit, etc...). Exploiter des données bruitées permet donc de renforcer la robustesse du modèle.
- **Biais et Indépendance:** Cet aspect est un sujet de recherche intense et un des piliers de l'*Ethique* de l'Intelligence Artificielle. Afin d'apprendre, un modèle (tout comme un être humain) devra posséder des biais de décision. Comprendre parfaitement et prévenir des dérives prédictives (comme les *prédictions auto-génératrices*) nécessitera la maîtrise et la détection de ces biais. Il est ainsi pertinent d'étudier les biais que possède notre jeu d'apprentissage (en considérant les particularités des méthodes d'apprentissage de l'algorithme utilisé) en les détectant dans un premier cas et si possible, les supprimer (ou modifier) si ils représentent un danger. Réaliser cette analyse est encore un travail immature et non abouti mais **capital** à moyen-terme.

3.10 Prédiction multi-label et multi-classe

3.10.1 Généralités

La caractérisation *multi-classe* / *multi-label* est généralement réalisée dans le cadre d'une tâche de classification.

Une classification est caractérisée comme *mono-classe* lorsqu'elle ne discrimine qu'une classe. Il s'agit donc d'une classification binaire (Est / N'est pas). Au contraire, dans le cas d'une classification *multi-classe*, le réseau doit discriminer plusieurs classes afin de prédire les caractéristiques de l'entité inférée.

L'architecture du réseau est dépendante du type de classification souhaitée. En effet, dans le cadre d'une classification binaire, un seul neurone de sortie est nécessaire. Ce neurone aura pour fonction de calculer $P(Y_{i,\text{classe}}|X_i, \theta)$, i.e la probabilité que l'entité i soit de la classe Y_{classe} sachant ses caractéristiques X et l'architecture du réseau définie par θ .

Au contraire, dans le cadre d'une tâche multi-classe, le réseau doit prédire n probabilités associées aux n classes définies. De ce fait, le réseau doit être capable de prédire $P(Y_{i,\text{classe}_j}|X_i, \theta)$,
 $j \in [1, n]$

Un neurone est capable de prédire la probabilité associée à une classe. Pour prédire n probabilités associées à n classes, il est donc nécessaire d'avoir n neurones sur la couche de sortie.

On parle de classification *multi-classe* lorsque le modèle discrimine plusieurs classes mais qu'une entité ne peut être associée qu'à **une** classe uniquement, i.e les classes sont **mutuellement exclusives**. Par exemple, supposons un

modèle de classification multi-classe capable de discriminer les chats et les chiens, les entités inférées par ce modèle seront classées comme chat **ou** comme chien uniquement. Or, le modèle théorique présenté précédemment calcule $P(Y_{i,\text{classe}_j}|X_i, \theta)$. Pour respecter le cadre de la classification multi-tâche, le réseau doit donc être modifié pour calculer $\underset{j \in [1,n]}{\operatorname{argmax}}(P(Y_{i,\text{classe}_j}|X_i, \theta))$.

Au contraire, on parle de classification multi-label lorsqu'une entité peut être définie par **plusieurs classes**. Elles ne sont donc pas exclusives. Par exemple, un homme à vélo peut être catégorisé comme *homme* et *cycliste*. De ce fait, la classification multi-label, au contraire de la classification multi-classe, n'impose pas l'utilisation de *argmax*.

En résumé, la classification *multi-classe* prédit positivement une unique classe parmi un ensemble de classes alors que la classification *multi-label* en prédit positivement aucune, une (ou plusieurs) parmi un ensemble de classes.

La différence d'architecture entre ces deux approches repose essentiellement sur la fonction d'activation de la couche de sortie. Ainsi, dans le cadre d'une classification *multi-classe*, la couche de sortie sera associée à une activation par la fonction *softmax* qui permet d'extraire la probabilité la plus élevée en plus de respecter la condition d'exclusion. Néanmoins, cette fonction impose $\sum_i P(Y_i|X) = 1$. Cette hypothèse est plus *forte* que la condition d'exclusion. En effet, elle impose que l'*univers* corresponde à l'ensemble des classes du modèle, ce qui est une contrainte forte.

Au contraire, dans le cadre d'une classification multi-label, chaque neurone de la couche de sortie sera associé à la fonction *sigmoïde* qui permet de calculer la probabilité d'appartenance à une classe indépendamment pour chaque classe. Chaque classe étant indépendante des autres, il n'y a pas d'impératif de somme égale à 1.

Remarque: La fonction *softmax* extrait la probabilité la plus forte proportionnellement aux valeurs de l'ensemble de probabilités obtenues⁷¹. De ce fait, une classe est nécessairement prédite *positivement* bien qu'il soit possible que l'entité analysée ne corresponde pas à une des classes du réseau. Par exemple, supposons un réseau capable de discriminer les cyclistes et les chats. Si on présente un chien, le réseau déterminera des probabilités faibles pour les classes cyclistes et chats. Néanmoins, la fonction *softmax* prédira nécessairement des probabilités indépendamment de la valeur "absolue" de ces prédictions. La classe prédite sera probablement chat car un chat "ressemble plus" à un chien qu'à un cycliste. Comme un chien ressemble bien plus à un chat qu'à un cycliste, *softmax* prédira une probabilité élevée pour la classe *chat*. Cette particularité

⁷¹L'utilisation de *softmax* impose l'hypothèse que chaque entité puisse être caractérisée par une classe du modèle.

impose de créer une classe *neutre* qui correspond à une entité non caractérisée par les autres classes. Ainsi, si un modèle discrimine n classes, dans les faits, il devra en discriminer $n+1$ pour considérer la possibilité d'une image non associée à une de ces classes.

Au contraire, la classification *multi-label* repose sur la valeur absolue des probabilités et non la valeur relative entre ces probabilités. Chaque probabilité étant indépendante, il est tout à fait possible qu'une prédiction de ce type de réseau soit négative pour chacune des classes. Ainsi, il n'y a pas d'impératif à la création d'une classe *neutre*.

Bien que l'approche *multi-label* soit plus souple et modulable, elle est plus dure à exploiter dans le cadre de l'apprentissage. L'approche *multi-classe* est souvent préférée lorsque le problème traité le permet.

3.10.2 Prédiction et distribution de données

La difficulté principale de la prédiction *multi-classe* (ou *multi-label*) est associée à la distribution des données d'apprentissage. En effet, il est possible qu'elle soit très déséquilibrée avec, par exemple, une classe fortement majoritaire et d'autres minoritaires. Cette particularité du jeu d'apprentissage induira un biais dans l'apprentissage qui peut être responsable d'un échec critique de l'apprentissage. Ce type de problématique est très répandu, notamment dans les tâches de détection d'entités rares qui sont très présentes dans le domaine médicale par exemple.

Supposons une tâche qui consiste à détecter les tweets *toxiques* et à les classifier selon différentes catégories (acharnement, contenu sexuel, discrimination raciale). Nous supposerons qu'un tweet peut appartenir à plusieurs catégories. Nous sommes donc dans le cadre d'une prédiction *multi-label*.

Notre jeu d'apprentissage correspond à un ensemble de tweets obtenus sur un intervalle continu et dont les tweets sont labellisés sans considération de leurs particularités (sain ou toxique). Il est évident que la majorité des tweets sont "sains" et de ce fait, non classifiés dans les sous catégories de toxicité. Ainsi, la majorité des tweets d'apprentissage sont négatifs pour chacune de ces classes, i.e présentent un vecteur de label égal à $[0, 0, 0]$. Nous supposerons les sous-catégories comme uniformément distribuées et la répartition sain/toxique équivalente à 95%-5%.

Le risque principal induit par ce type de données déséquilibrées est la (possible) convergence vers un minimum local sans pouvoir explicatif. En effet, nous avons 95% des données classées comme sain. Ainsi, si le modèle considère que toute donnée est saine, alors il aura 95% de bonne prédiction sur son jeu d'apprentissage. Les données d'apprentissage associées à un tweet toxique étant rares, leurs impacts sur la fonction de perte sont *noyés* dans l'ensemble des "bonnes prédictions", ce qui diminuera grandement leur pouvoir

d'apprentissage. Ce phénomène est classique et particulièrement critique dans le cadre de l'apprentissage machine en général⁷².

3.10.3 Méthodes d'apprentissage

Afin de permettre un meilleur apprentissage, différentes approches sont envisageables. Elles sont appliquées au niveau des données d'apprentissage ou au niveau du modèle entraîné. Ces méthodes favorisent le phénomène de sur-apprentissage (voir Section 5.1). Elles doivent donc être utilisées avec grande attention.

3.10.3.1 Au niveau des données

Les principales approches au niveau des données reposent sur des méthodes d'échantillonnage. Elles sont divisées en deux groupes:

- **Undersampling:** Afin d'uniformiser les distributions, ce type d'approche propose de supprimer aléatoirement des données de la classe majoritaire. Bien que simple d'utilisation, cette méthode est *destructrice* et peut conduire à la perte de données au pouvoir explicatif important.

Afin de lutter contre ce risque, des améliorations ont été proposées afin de permettre la sélection de données au faible pouvoir explicatif. Elles analysent les données pour détecter les caractéristiques redondantes et se focaliser sur la suppression des *doublons*. Des approches de *Data Decontamination* sont envisageables aussi.

- **Oversampling:** Cette méthode *augmente* les données dont la classe est sous-représentées. L'approche standard consiste à répliquer aléatoirement des données dont la classe est minoritaire afin d'atteindre une distribution uniforme des classes. Néanmoins, un risque élevé de sur-apprentissage est à considérer.

Pour améliorer le pouvoir explicatif des données créées, des améliorations ont été faites notamment via l'utilisation de données artificiellement créées. Pour cela, l'interpolation d'entités *voisines* est exploitée. Les méthodes standards reposent sur l'utilisation du clustering qui permet de considérer l'équilibre intra/inter-classe. De même, exploiter le *Boosting* permet d'isoler les exemples *difficiles* et de cibler l'augmentation des données sur des exemples à problèmes⁷³. Une autre approche (propre au Deep-Learning) consiste à créer des *minibatch* dont la distribution des classes est garantie uniforme par la sélection aléatoire d'exemples issus de chacune des classes.

⁷²Ce problème illustre la nécessité d'une étude préalable des données afin de détecter ce phénomène

⁷³Il y a un risque important de sur-apprentissage avec cette approche.

3.10.3.2 Au niveau du modèle

D'autres méthodes s'appliquent au niveau de l'architecture du modèle, de sa méthode d'apprentissage et de son rapport avec les données utilisées.

- **Calibration:** La *Calibration* est une méthode pour ajuster la prédiction réalisée par le modèle afin de limiter le biais induit par la distribution des données uniformisée par les méthodes ci-dessus. L'approche standard consiste à compenser le biais prédictif par la considération de la probabilité *a priori*⁷⁴ de la classe prédite.

Il a été montré qu'un réseau neuronal estime la probabilité à posteriori d'une entité. Par conséquent, un réseau neuronal estime:

$$y_{classe}(x) = p(\text{classe}|x) = \frac{p(\text{classe}) * p(x|\text{classe})}{p(x)}$$

$p(\text{classe})$ est biaisé par la méthode d'échantillonnage qui ajuste les données d'apprentissage. Afin de corriger ce biais, il est nécessaire de considérer la distribution initiale des données. Ainsi nous obtenons:

$$y_{classe}(x_i)^{\text{calibrated}} = y_{classe}(x_i) * \frac{\text{Card}(x_{\text{classe}})}{\text{Card}(x)}$$

- **Pondération des données d'apprentissage:** Afin de considérer le déséquilibre des distributions, il est possible de modifier l'importance associée à une donnée selon sa classe. L'approche traditionnelle repose sur une pondération en fonction de la fréquence de classe. Ainsi, une données appartenant à une classe fréquente aura une influence plus faible afin de compenser la présence élevée de ce type de données dans le minibatch. Au contraire, une donnée appartenant à une classe rare aura une erreur majorée afin de compenser la faible représentation de cette classe. Ainsi, l'erreur associée à une donnée est définie par:

$$Er_i^{\text{ponderated}} = Er_i * \frac{\text{Card}(x)}{\text{Card}(x_{\text{classe}})}$$

Au lieu de modifier la méthode de calcul de l'erreur, il est possible d'influencer la valeur du pas d'apprentissage en fonction de la données d'apprentissage. Ainsi, dans le cadre d'un apprentissage par gradient stochastique, la valeur du pas sera pondérée selon la classe de la donnée traitée. Le pas serait minoré si la donnée appartient à une classe très représentée et majoré dans le cas contraire. Il est possible de généraliser cette solution pour la rendre exploitable avec un minibatch de données.

Cette calibration est rudimentaire et simpliste. Une approche reposant sur une régression logistique[54] permet une prédiction des poids plus approfondie.

⁷⁴Des bases sur l'inférence Bayésienne sont nécessaires.

Si vous souhaitez approfondir vos connaissances sur cette problématique, veuillez vous référer à l'article [8] qui résume les principales méthodes modernes tout en proposant un recueil bibliographique important pour la poursuite de vos recherches. De même, l'article [90] propose une approche de pondération des données basée sur le comportement du gradient durant l'apprentissage. Cette approche est généralisable à tout type d'architecture tout en présentant une efficacité notable. Il s'agit sans doute d'une des approches à l'état de l'art pour ce type de problème.

3.11 Calcul matriciel et neurones

Un réseau de neurones est une structure qui nécessite beaucoup de ressources. Optimiser le temps de calculs est une nécessité absolue. Pour cela, on exploite les capacités du calcul matriciel qui permet l'exploitation de données à grande échelle grâce à sa capacité de parallélisation⁷⁵. Nous supposerons comme acquis les fondamentaux du calcul matriciel, notamment la multiplication.

Nous avons vu qu'un neurone est défini par une somme de ses entrées pondérées par ses poids (additionnée par la suite avec le biais). Cet ensemble forme un *logit* et ce dernier va être utilisé par la fonction d'activation du neurone pour déterminer la sortie du neurone. La problématique du calcul se situe donc majoritairement dans le calcul du *logit*. Pour cela, une représentation matricielle est possible.

Supposons un vecteur de données dans R^{784} . Il peut être représenté par la matrice:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{784} \end{bmatrix}$$

Supposons un vecteur de poids d'un neurone $W = [w_1, w_1, w_2, \dots, w_{784}]$. Nous souhaitons multiplier terme à terme l'entrée avec son poids associé. On observe donc que ce comportement est réalisable par une multiplication matricielle en considérant W^T .⁷⁶ De plus, dans un neurone, le nombre d'entrée coïncide avec le nombre de poids. La concordance des dimensions est donc respectée. On obtient donc:

$$W = \begin{bmatrix} w_{1,1} \\ w_{2,1} \\ \vdots \\ w_{784,1} \end{bmatrix} \longrightarrow W^T = [w_{1,1} \quad w_{1,2} \quad \dots \quad w_{1,784}]$$

⁷⁵Calculs simultanés sur GPU

⁷⁶On exploite la transposé de W pour avoir un vecteur colonne et non ligne afin de permettre le produit matriciel.

Ainsi, le calcul du logit d'un neurone est égal à:

$$\text{logit} = [w_{1,1} \quad w_{1,2} \quad \dots \quad w_{1,784}] * \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{784} \end{bmatrix}$$

Nous avons vu comment représenter le fonctionnement d'un neurone mais un réseau de neurones possède (l'écrasante majorité du temps), plus d'un neurone par couche. Afin de représenter cette spécificité, l'astuce consiste à représenter l'ensemble des poids de l'ensemble des neurones de la couche à travers une même matrice. Ainsi, pour une couche de 10 neurones avec 784 entrées, nous considérerons une matrice de dimension $784 * 10$, i.e, une colonne représente les poids d'un même neurones. De même, supposons un minibatch de 5 données, nous aurons donc une matrice de dimension $784*5$. Soit X , la matrice représentant le minibatch et W , matrice de poids des différents neurones:

$$X = \begin{bmatrix} x_{1,1} & \cdots & x_{1,5} \\ x_{2,1} & \cdots & x_{2,5} \\ \vdots & \ddots & \vdots \\ x_{784,1} & \cdots & x_{784,5} \end{bmatrix}, \quad W = \begin{bmatrix} w_{1,1} & \cdots & w_{1,10} \\ w_{2,1} & \cdots & w_{2,10} \\ \vdots & \ddots & \vdots \\ w_{784,1} & \cdots & w_{784,10} \end{bmatrix}$$

Afin de calculer les logits, nous calculons $W^T X$ soit le produit matriciel d'une matrice $10*784$ et $784*5$. Nous obtenons donc une matrice de sortie de dimensions $10*5$. Une colonne de la matrice de sortie représente ainsi les sorties des différents neurones pour une même données et une ligne, les sorties d'un même neurone pour différentes données.

Cette méthode est appliquée pour tout réseau et tout type de donnée en entrée. C'est pourquoi il est nécessaire de considérer exclusivement des données numériques. L'exploitation de données non numériques telles que du texte ou des variables catégorielles demandent une étape de pré-traitement pour les rendre exploitables. C'est notamment l'objectif des *Words Embedding* qui propose une projection vectorielle d'un mot, transformant une donnée textuelle en une donnée vectorisée exploitable par un réseau. Dans le cas de données catégorielles, une méthode standard est le *One Hot Encoding* où une variable catégorielle à n catégories est remplacée par un vecteur binaire dans R^n où l'ensemble des dimensions est à 0 exceptée la dimension associée à la catégorie initiale de la donnée⁷⁷.

L'écriture matricielle d'un neurone est souvent exploitée dans la littérature. Ainsi, par exemple, supposons une couche de neurones activée par la fonction *softmax*. Une écriture standard de ce neurone serait:

$$Y = \text{softmax}(W^T X + b)$$

⁷⁷Le vecteur aurait la forme $[0, 0, \dots, 1, 0]$

Ou encore:

$$Y_j = \frac{\exp(W_j^T X + b_j)}{\sum_i \exp(W_i^T X + b_i)}$$

L'écriture mathématique a tendance à "alourdir" la lecture des papiers de recherche. Bien que les notations puissent impressionner, l'intuition et la compréhension des idées développées sont souvent accessibles à un lecteur sans formation mathématique avancée.

Mois	Bénéfices de l'entreprise	Bénéfices d'un employé X
Janvier	$2 * 10^7$	$4 * 10^2$
Février	$8 * 10^7$	$2 * 10^2$
Mars	$1 * 10^6$	$1 * 10^3$
Avril	$5 * 10^8$	$3 * 10^1$
...

Figure 21: Exemple d'un jeu de données (toute ressemblance avec des données réelles est fortuite)

4 Normalisation des données

Les données utilisées par un réseau de neurones peuvent être très variées et de nature différente. Ainsi, par exemple, dans le cas d'image, les données sont très similaires: matrice Hauteur*Largeur à valeurs dans $[0, 255]$ (la résolution de l'image - nombre de pixels de l'image - est supposée constante). Il est souvent préférable de standardiser ses données afin de permettre un apprentissage de qualité.

Supposons maintenant des données numériques et financières: le bénéfice d'une entreprise et le bénéfice d'un employé lambda. Les données sont visibles sur la Figure 21. Les valeurs des données de l'entreprise sont importantes (de l'ordre de grandeur $[10^5, 10^7]$) et les données de l'employé plus faible (de l'ordre de grandeur $[10^1, 10^3]$). Cette différence provoquera donc une valeur moyenne (exprimée par la moyenne d'un point de vue statistique) significativement différente entre ces deux sous-ensembles de données.

De même, la variation entre les données (exprimée par la variance d'un point de vue statistique) sera d'un autre ordre de grandeur. Par exemple, entre Mars et Avril, la variation du bénéfice de l'entreprise se compte en millions alors que la variation du bénéfice d'un employé se compte en centaines.

Revenons au contexte du réseau de neurones et supposons une problématique qui reposera sur notre jeu de données. Nous avons vu qu'un réseau apprend en minimisant une fonction de coût et qu'un poids d'un neurone est corrigé en dépendance avec sa valeur d'entrée. Il est donc évident que des valeurs à des échelles différentes vont sur(sous)-stimuler le neurone. Ceci est très problématique car cette différence va provoquer une pondération des données d'entrées. Dans notre exemple, les données de l'entreprise seront sur-pondérées par rapport aux données de l'employé. Il est donc nécessaire de **normaliser** les données afin d'éviter ce genre de problème. Pour cela, différentes (parfois complémentaires) existent.

4.1 Centrer les données

Afin de limiter le problème d'échelle des données, il est possible de les *centrer*. Centrer les données modifient les données afin que la moyenne de ces données soit 0. Les nouvelles données sont obtenues selon la relation suivante: $data_{i,centre} = data_{i,raw} - \mu$ où μ est la moyenne de cet ensemble de données.

Important: La moyenne n'est pas réalisée sur l'ensemble du jeu de données mais sur les données associées à un même attribut. Dans notre exemple, il y aurait une moyenne pour les données de l'entreprise et une moyenne pour les données de l'employé.

Cette modification est réalisée par la quasi-totalité des pré-traitements de données et limite grandement le risque de biais dans l'apprentissage.

4.2 Réduire les données

Les données d'un jeu d'apprentissage peuvent avoir une variance importante, i.e des valeurs éloignées de la moyenne associée. Les conséquences sont similaires à celles provoquées par une échelle différente de la moyenne. L'idée est donc d'utiliser une échelle commune à l'ensemble du jeu de données en imposant un écart-type de 1. Ainsi, les données seront représentées par une valeur contenue dans $[-1, 1]$. Les nouvelles données sont obtenues selon la relation suivante: $data_{i,reduit} = \frac{data_{i,raw} - \mu}{\sigma}$ où σ est l'écart-type de cet ensemble de données.

Important: L'écart-type n'est pas réalisé sur l'ensemble du jeu de données mais sur les données associées à un même attribut. Dans notre exemple, il y aurait un écart-type pour les données de l'entreprise et un écart-type pour les données de l'employé.

Dans les faits, cette modification est peu employée de manière isolée.

4.3 Centrer-Réduire les données

Centrer-Réduire les données revient à centrer et réduire les données, i.e réaliser les deux modifications explicitées précédemment. Ainsi, les nouvelles données sont définies telles que: $data_{i,reduit} = \frac{data_{i,raw} - \mu}{\sigma}$ avec μ , la moyenne et σ , l'écart-type de ce jeu de données.

Cette normalisation est la normalisation la plus utilisée dans le pré-traitement des données et efficace dans le traitement de la majorité des jeux de données. Elle est souvent appliquée en **traitement par défaut**. Néanmoins, bien que centrer et réduire soit très recommandé dans la majorité des cas, réduire les données dépend du cas à traiter. Par exemple, dans le cas de différents jeux d'images, l'écart-type relatif tend à être similaire entre eux. *Réduire l'image*

tend donc à être peu utile.

Important: Il est **capital** de définir la moyenne et l'écart-type sur le jeu d'apprentissage uniquement. Par exemple, supposons deux jeux de données *train* et *test*. Nous apprendrons notre modèle avec *train* et nous l'évaluerons avec *test*. Afin de normaliser les données, il est nécessaire de pouvoir déterminer la moyenne et l'écart-type. Pour cela, il est indispensable de ne considérer que le jeu de données d'apprentissage. Ainsi, les paramètres μ et σ seront déterminés avec *train* et, lors de l'évaluation du modèle, la normalisation utilisera les paramètres μ et σ calculés avec *train*. Une erreur classique est de déterminer ces paramètres sur l'intégralité des données (*train* et *test*). Ceci est une **erreur grave** qui peut grandement nuire à l'évaluation du modèle.

4.4 Analyse en Composante Principale

Il est possible que les données à étudier soient de très grande dimension (plusieurs milliers d'attribut⁷⁸ voir plus). Analyser des données volumineuses impose une contrainte de temps (pour l'apprentissage et la prédiction - ce qui est problématique dans le cas du temps-réel). Il est souvent préférable de diminuer le nombre de dimension des données afin de limiter ces problèmes. Pour cela, une approche simple est efficace: *l'Analyse en composante principale* (ACP).

Dans un jeu de données, chaque attribut représente une dimension. Ainsi, un jeu de données avec n attributs sera représenté par un vecteur à n dimensions. L'ACP permet de considérer les corrélations entre les attributs d'un jeu de données afin de créer de nouvelles dimensions. Une dimension créée par ACP permet donc d'expliquer l'information utile expliquée par plusieurs attributs et, de ce fait, de diminuer le nombre de dimensions⁷⁹. Cette approche est mathématique. Nous ne détaillerons pas son fonctionnement dans cette introduction. Il faut juste noter l'importance de cette approche pour l'analyse de données très volumineuses et la capacité de projeter des données de dimension N dans une dimension M choisie par l'utilisateur (souvent 2 ou 3 pour la visualisation à une centaine pour la conversion de données volumineuses).

Important: Cette approche est destructrice. En effet, la réduction de dimension impose une perte de données qui peut être importante ou faible selon la nature des données et l'importance de la réduction. Une analyse approfondie de l'impact de la transformation est **capitale** pour limiter la perte d'information utile. Il est **intéressant** de noter que l'ACP est utilisable pour supprimer le bruit des données. En effet, un jeu de données peut présenter des valeurs aberrantes ou douteuses qui provoqueront un bruit dans les données et nueront à l'apprentissage du modèle. Diminuer le nombre de dimension permet donc de perdre l'information la moins représentative souvent caractéristique du bruit et

⁷⁸Attribut correspond à une catégorie du jeu de données, pas à une donnée brute

⁷⁹Plus les données sont corrélées, plus le nombre de dimension peut être faible sans perte d'information majeure

de ce fait, de conserver des données généralisées⁸⁰. Néanmoins, la détermination de présence de bruit ou non est souvent "tricky" et relève plus d'une sensibilité personnelle et d'un pari sur les données que d'une action mathématiquement démontrable. D'un point de vue métier, il est standard de conserver au moins 80% de l'information utile. Au-delà, le risque de destruction est trop important et souvent néfaste.

⁸⁰Et potentiellement de meilleure qualité selon le cas

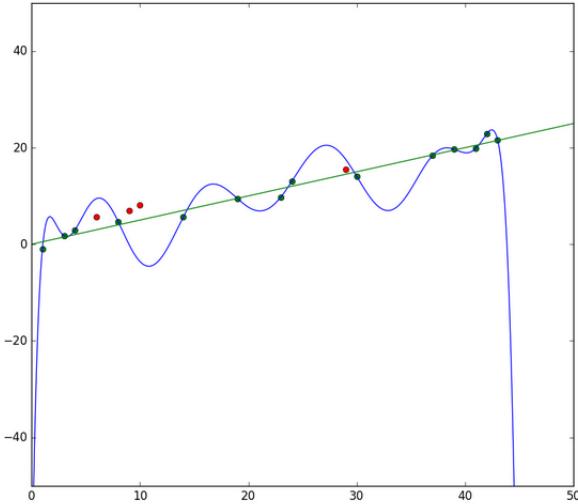


Figure 22: Exemple de sur-apprentissage (bleu) et d'apprentissage optimal (vert)

5 Régularisation et sur-apprentissage

5.1 Le sur-apprentissage

Le **danger principal** de tout algorithme d'apprentissage automatique est le *sur-apprentissage*. Comme vu dans l'introduction de cette partie, un algorithme d'apprentissage automatique cherche à approximer une fonction capable d'expliquer les données **et** capable de généralisation. Mais que signifie "généralisation" ?

Pour expliquer cette problématique, observons la Figure 22. Nous pouvons observer des points verts représentant les données du jeu d'apprentissage et des points rouges, les données inconnues qui doivent être prédites par le modèle. La courbe bleue et la courbe verte présentent deux fonctions apprises par deux modèles. On peut apercevoir que la courbe bleue passe par l'ensemble des points verts alors que la courbe verte possède une moins bonne performance. On peut donc naïvement dire que la courbe bleue a mieux appris. C'est vrai dans l'absolu: la courbe bleue a mieux appris le jeu de données d'apprentissage mais en plus d'apprendre le comportement général des données, elle a appris son bruit, ce qui lui donne son comportement oscillant et "imprévisible". Le fait d'apprendre le bruit est appelé *sur-apprentissage* et est désastreux pour le modèle qui perd ainsi sa capacité de généralisation. Au contraire, la courbe verte n'apprend pas le bruit des données. Elle apprend moins les données d'apprentissage mais conserve sa capacité de généralisation. L'idéal est donc de trouver un compromis entre apprentissage du jeu de données et capacité de généralisation: ceci est

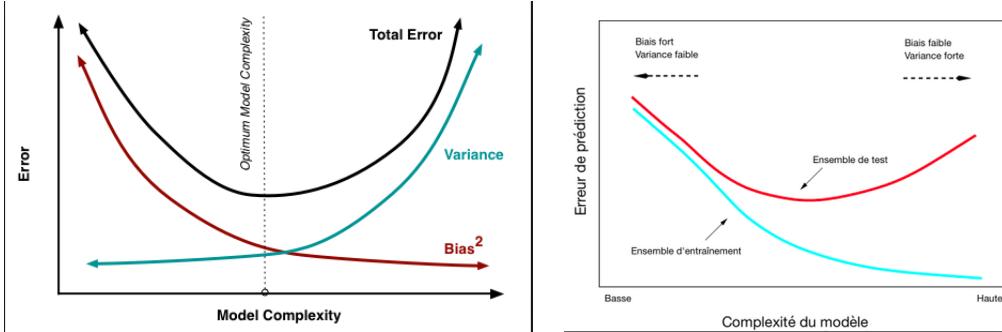


Figure 23: Compromis biais-variance et impact sur le modèle

appelé *compromis biais-variance*.

Une explication graphique est visible sur la Figure 23. Le biais représente l'erreur réalisée sur l'apprentissage des données. Ainsi, un biais très faible sous-entend un sur-apprentissage car le bruit aura été appris. Au contraire, la variance représente l'importance des variations de la fonction hypothèse approximée, i.e la sensibilité du modèle aux variations des données et donc au bruit. De ce fait, plus la variance est élevée, plus le modèle perd en généralisation et devient inutilisable. Et, comme observé sur la Figure 22, le sur-apprentissage favorise un comportement à forte variance (représenté par les oscillations). L'objectif est donc de réaliser un compromis: optimiser la diminution du biais et de la variance.

D'un point de vue expérimental, la notion de biais et de variance est représentée par l'erreur de prédiction réalisée sur l'ensemble d'apprentissage et de test. Ainsi, l'objectif est d'arrêter l'apprentissage lorsque la courbe de prédiction du jeu de test augmente de manière significative alors que la prédiction sur le jeu d'apprentissage tend à diminuer.

Bien sur, la problématique du sous-apprentissage existe aussi et représente un modèle trop généraliste qui n'explique pas suffisamment les données d'apprentissage qui discriminent ses prédictions. Un exemple est visible sur la Figure 24 où la fonction créée est bien trop "neutre". Il est évident qu'un modèle sous-entraîné possède des performances faibles.

5.2 Limiter le sur-apprentissage

Important: Cette partie est fondamentale et doit être comprise pour développer un réseau de qualité, notamment dans le cadre d'architecture très profonde que nous étudierons par la suite.

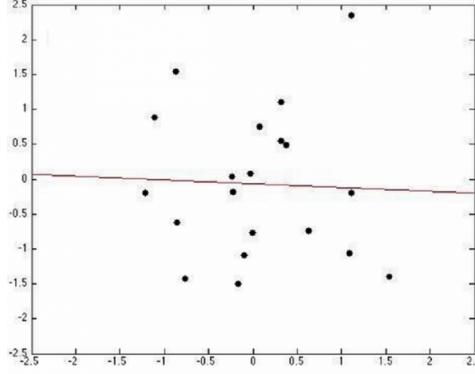


Figure 24: Exemple de sous-apprentissage

5.3 Régularisation

Les poids d'un neurone ne sont pas bornés⁸¹. Cette spécificité permet une grande disparité entre les valeurs que peuvent prendre les poids d'un neurone au risque de voir certains poids *explorer* alors que d'autres seraient *faibles*⁸². Cette différence de valeur est très préjudiciable et favorise un comportement de sur-apprentissage où un neurone sur-réagit aux stimulations portées par les poids de haute intensité. Pour limiter l'explosion des poids, une régulation peut être imposée.

Cette régulation peut être associée à la fonction de coût, lui rajoutant un nouveau critère de discrimination. Ainsi, supposons la fonction de coût définie par Mean Squared Error⁸³. Elle est définie par: $\mathcal{L} = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$. Avec une régulation, nous obtenons $\mathcal{L}_{reg} = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2 + \epsilon(\omega)$ où ϵ dépend uniquement des poids des neurones du réseau. Ainsi, la fonction de coût sera directement dépendant des poids des neurones et tendra à limiter leurs évolutions divergentes.

5.3.1 L1-Régulation

La première régulation est la L1-Régulation. Elle s'exprime sous la forme $\lambda|\omega|$ où λ correspond à l'intensité associée à la régulation. Plus λ est élevé, plus on pondère l'importance d'avoir des poids faibles. Ainsi, dans le cas de Mean Squared Error, nous obtenons $\mathcal{L}_{reg} = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2 + \lambda|\omega|$.

⁸¹Dans les cas d'une architecture standard

⁸²Cette particularité est aggravée si les données en entrée du neurone ne sont pas normalisées. Les données d'apprentissage peuvent être normalisées en entrée tout en provoquant ce phénomène en interne.

⁸³Voir la section 3.5.1 pour plus d'informations

Cette régulation est très utile dans le cas d'extraction de *feature*. En effet, cette régulation tend à rendre le réseau *sparse* en permettant que des poids deviennent très proches de zéro (annulant ainsi une entrée du neurone). Cette régulation favorise les entrées discriminantes et limite les entrées porteuses de bruit. Il est ainsi possible d'extraire les composantes principales du réseaux. Néanmoins, dans les faits, elle est rarement utilisée car présente de moins bons résultats que la régulation L2 (sauf dans le cas spécifique d'extraction de feature). De plus, n'étant pas quadratique, sa minimisation est difficile.

5.3.2 L2-Régulation

L2-Regulation est définie par la contrainte $\frac{1}{2}\lambda\omega^2$. La fonction de coût régulée devient donc $\mathcal{L}_{reg} = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2 + \frac{1}{2}\lambda\omega^2$.

Cette régulation favorise des valeurs de poids proches en limitant les pics de valeurs. Ainsi, elle permet de faire en sorte que le neurone ne se focalise pas sur des entrées dominantes tout en délaissant les autres jugées moins pertinentes. Elle limite donc la sur-spécialisation et de ce fait, l'overfitting. Cette régulation est efficace et très utilisée aujourd'hui.

5.3.3 ElasticNet-Régulation

ElasticNet-Régulation est une combinaison linéaire de la Régulation L1 et L2. Elle s'exprime sous la forme $\lambda_1|\omega| + \frac{1}{2}\lambda_2\omega^2$ où λ_1 et λ_2 définissent l'importance de cette régulation dans le calcul de la performance du modèle et la pondération entre les deux régulations (comportement équilibré ou pondération d'une des deux régulations). Ainsi, la fonction de coût est dorénavant définie par la relation suivant: $\mathcal{L}_{reg} = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2 + \lambda_1|\omega| + \frac{1}{2}\lambda_2\omega^2$

Cette régulation réunit *le meilleur des deux mondes* (issus de L1 et L2) bien que moins souvent appliquée que la L2-Régulation.

5.4 Max norm constraints

Contrairement à la régulation L1 et L2, Max norm constraints ne s'applique pas à la fonction de coût mais directement au vecteur de poids du neurone. Alors que L1 et L2 pénalisent les poids élevés⁸⁴, Max norm constraints agit directement sur ces derniers. Ainsi, chaque vecteur de poids doit respecter la condition $\|\omega\|_2 < cnst$ ou *cnst* de petite taille (2,3 voire 4 par exemple). Si l'égalité n'est pas respectée, l'intégralité des poids est normalisée⁸⁵ afin de respecter la condition.

⁸⁴Pénaliser augmente la fonction de coût mais il n'y a pas de contrainte directe imposée à l'intégralité du réseau. Ainsi, L1/L2 favoriseront la limitation des poids les plus élevées en priorité alors que Max norm constraints les corrigera tous sans distinction

⁸⁵Ce n'est pas la normalisation au sens statistiques !

Cette régulation est particulièrement performante associée avec le *DropOut*⁸⁶ mais présente la difficulté de paramétrage du seuil (c_{nst}) qui est un hyperparamètres. Sa détermination se fait de manière empirique et expérimentale, ce qui peut rendre sa recherche délicate.

5.5 DropOut

Contrairement à la régulation qui agit sur les neurones de manière isolée, le DropOut[104] est une approche de régulation d'architecture, i.e une approche qui influence l'interaction entre neurones. Le DropOut sélectionne aléatoirement (selon une probabilité ρ) des neurones du réseau et leur impose une activation⁸⁷ nulle. La sélection des neurones inactifs est renouvelée à chaque donnée d'apprentissage. Lors de l'utilisation du réseau hors apprentissage, chaque sorties des neurones concernées par le DropOut sont multipliées par ρ et toutes sont actives. Un exemple est visible sur la Figure 25.

Cette méthode permet de limiter l'inter-dépendance (Voir Section 3.7.3) entre les neurones et favorise la création d'un réseau épars. L'apprentissage du modèle est ainsi plus robuste, plus rapide et moins soumis aux problématiques associées au gradient. Le DropOut est souvent exploité en plus d'une *Régulation* des poids bien que son comportement soit comparable à la régulation L2. Une valeur standard pour ρ est 0.5. Cette approche est souvent employée pour les réseaux Full-Connected mais peut être appliquée à d'autres architecture notamment les réseaux convolutifs. Néanmoins, l'efficacité de son utilisation sur des couches convolutives est sujet à débat.

Important: DropOut est souvent utilisé avec des fonctions d'activation f tel $f(0)=0$. En effet, le DropOut ne force pas la sortie à 0 mais son activation. Ainsi, si une fonction d'activation n'est pas nulle en 0, le neurone ne sera pas (complètement) inactif bien que son comportement soit constant. De plus, le biais est indépendant de l'activation et n'est pas soumis au DropOut.

5.6 DropConnect

DropConnect[115] est une généralisation de DropOut. Au lieu de supprimer l'activité d'un neurone en bloquant son activation, cette méthode annule les poids d'entrée du neurone de manière indépendante selon une probabilité ρ . Ainsi, un neurone peut voir son activation éteinte (comparable à DropOut) ou juste partiellement. Un exemple de DropConnect est visible sur la Figure 25. Cette méthode est moins utilisée que DropOut et ses résultats plus rares. Il n'est pas aisés de déterminer quelle approche est la plus performante objectivement mais le DropConnect offre plus de souplesse et de configurations possibles.

⁸⁶Voir <http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf>, partie 5.1

⁸⁷L'activation correspond à la somme pondérée des entrées

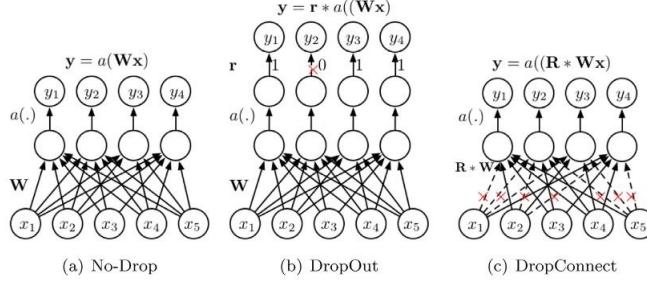


Figure 25: Comparaison d'un réseau sous DropOut et DropConnect

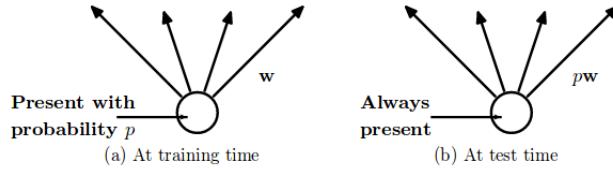


Figure 26: Relation des poids après DropOut

5.7 DisturbLabel

DisturbLabel[121] est une méthode de régularisation qui exploite l'approche par *bruitage*. L'idée derrière cette méthode est de modifier aléatoirement le label d'une donnée d'apprentissage dans chaque minibatch d'apprentissage. On suppose une problématique multi-classe bien qu'il soit possible de généraliser à une problématique multi-label.

Ainsi, DisturbLabel rajoute une étape de sampling supplémentaire qui génère, pour chaque couple de donnée (x_D, y_C) , un vecteur label $\hat{y} = [\hat{y}_1, \dots, \hat{y}_C]$ (avec C , nombre de label) issu d'une loi Multinouilli (Bernoulli généralisée) $P(\alpha)$ tel que:

$$\begin{aligned}\hat{c} &\sim P(\alpha) \\ \hat{y}_c &= 1 \\ \hat{y}_i &= 0, \quad \forall i \neq \hat{c}\end{aligned}$$

De même, $P(\alpha)$ est définie par $p_c = 1 - \frac{C-1}{C}$ et $p_i = \frac{1}{C} * \alpha$ pour tout $i \neq c$ et c , indice du label valide. α est le *taux de bruitage*. Si $\alpha = 0$, il n'y a pas de bruitage. Si $\alpha \rightarrow 100\%$, alors la labellisation du jeu d'apprentissage est aléatoire. Il est donc nécessaire de garder un α de faible valeur.

5.8 Dense-Sparse-Dense training

Le Dense-Sparse-Dense[30] training est une méthode comparable à une variante du DropOut/DropConnect. Cette méthode cherche à proposer une méthode d'apprentissage plus efficace dans le cadre de réseaux très profonds. L'idée soutenue par cette méthode est qu'un réseau très profond est capable d'acquérir une compréhension plus profonde d'un phénomène mais plus enclin à en apprendre le bruit et le comportement instable. Leur postulat est que le bruit est porté par les poids faibles des neurones. Ainsi, l'objectif est d'être capable d'isoler les poids d'importance qui propagent une information fiable et les poids qui propagent le bruit. Pour cela, cette méthode (Figure 27) se découpe en 3 phases:

1. **Initial Dense Phase:** L'apprentissage du réseau est fait normalement sans contrainte particulière. Les méthodes de régulations (et autres) peuvent être exploitées durant cette phase. L'objectif de cette phase n'est pas "d'apprendre" mais de détecter les poids influents et non influents (en fonction de leurs valeurs absolues).
2. **Sparse Phase:** Durant cette phase, les poids sont classées par couche selon leurs valeurs. Seuls les $\lambda\%$ plus élevés sont conservés intacts, les autres devenant nuls. λ est un hyperparamètre à déterminer (il est conseillé de prendre une valeur entre 25% et 50% comme valeur par défaut). Le réseau devient alors éparse, permettant d'obtenir un réseau plus robuste et plus économique. Il est empiriquement montré qu'un réseau éparse tend à avoir des résultats équivalents voire meilleurs qu'un réseau dense car ce dernier est plus sensible aux problématiques d'apprentissage. Un apprentissage du réseau est réalisé avec cette architecture.
3. **Final Dense Phase:** Durant cette phase, les connexions coupées sont restaurées. Les poids restaurés sont initiés à 0 et le pas d'apprentissage du gradient faible (1/10 du pas initial). En effet, le réseau actuel présente une stabilité et une convergence correcte. Il n'est donc pas souhaitable de "bouleverser" son architecture mais juste de favoriser sa performance en finalisant son comportement dans ses détails d'apprentissage. Le réseau peut ainsi converger vers un meilleur minimum local voire "en sortir" vers un autre selon le cas.
4. **Répétition:** Ce cycle (3 phases) peut être répété afin de favoriser une meilleure convergence.

5.9 Batch Normalization

Batch Normalization[46] est une méthode qui vise à augmenter la robustesse du modèle en favorisant sa capacité de généralisation. Supposons un jeu de données d'image de chiens blancs, si une image d'un chien noir apparaît lors de la prédiction, il est évident que le réseau ne fonctionnera pas car la distribution des

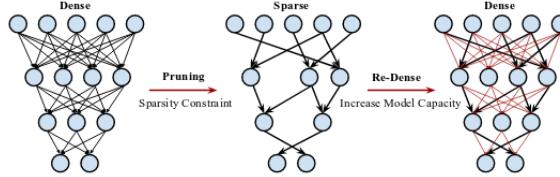


Figure 27: Dense-Sparse-Dense Routine

données d'apprentissage et de prédiction ne correspondent pas. Néanmoins, le comportement des données peut être similaire et donc, la fonction obtenue sur les chiens blancs pourraient être effective. Batch Normalization permet d'aligner les distributions et de limiter cette problématique. Un exemple illustratif est visible sur la Figure 28. La justification de performance de cette approche est très mathématique et repose sur un socle solide de Statistiques. Nous ne l'étudierons pas dans cette introduction.

L'idée de Batch Normalization est de généraliser la normalisation des données à travers le réseau de neurones. La méthode est comparable à une normalisation standard mais présente des spécificités associées à la présence d'un minibatch qui représente un sous-ensemble du jeu d'apprentissage (qui plus est, variable). La normalisation réalisée est visible sur la Figure 29. Veuillez vous référer à la publication associée[46] pour plus de détails mathématiques sur son fonctionnement.

De plus, cette méthode permet de mieux isoler les couches entre elles, limitant l'inter-dépendance, permet d'utiliser un pas d'apprentissage plus important car les données internes au réseau sont normalisées (ce qui limite les valeurs extrêmes et donc les valeurs du gradient) et possède une action de régulation. L'utiliser avec DropOut (ou équivalent) est efficace bien qu'il soit préférable d'utiliser une probabilité plus faible d'éteindre un neurone dans cette configuration. Un gain de vitesse est aussi observé du fait de la limitation des valeurs des données, ce qui favorise une implémentation optimisée.

De même que le DropOut, Batch Normalization fait partie des améliorations majeures proposées dans le cadre du développement d'un réseau de neurones et son utilisation est quasi-récurrente à tout réseau d'ampleur.

5.10 Critère d'arrêt de l'apprentissage

Pour réaliser un bon apprentissage, il est nécessaire de considérer le *compromis biais-variance* afin de limiter le sur-apprentissage. Bien que la problématique soit facilement assimilable, s'en prémunir reste difficile.

Une bonne pratique repose sur l'utilisation d'architectures de régulation qui

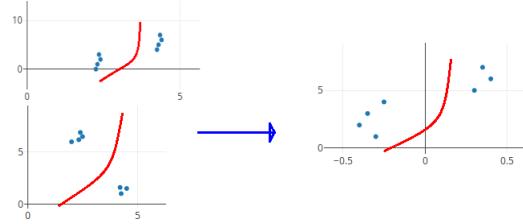


Figure 28: Non-alignement des distributions

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
 Parameters to be learned: γ, β
Output: $\{y_i = BN_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

Figure 29: Normalisation par Batch Normalization

tend à limiter le phénomène de sur-apprentissage⁸⁸. Bien que fonctionnelles, ces architectures ne garantissent pas un apprentissage contrôlé. Une approche complémentaire est donc de savoir **quand** arrêter l'apprentissage afin d'obtenir le meilleur état du modèle. Dans le cas idéal, l'arrêt se situe à l'intersection de la courbe de biais et de variance. Mais comment savoir si on se situe dans un intervalle proche de ce point de référence ?

La courbe de biais est associée aux erreurs de prédiction sur les données d'apprentissage alors que pour la courbe de variance, ce sont les erreurs de prédiction sur les données de validation (Figure 23). La courbe de biais décroît logiquement durant l'apprentissage jusqu'à atteindre une valeur asymptotique caractérisée par un résidu de bruit. Au contraire, la courbe de variance va décroître progressivement avant de croître lorsque le modèle commence à perdre sa capacité de généralisation. L'objectif étant d'assurer la meilleure performance prédictive tout en ayant une forte capacité d'abstraction, se concentrer sur le comportement de la courbe de variance semble prioritaire. Il est ainsi nécessaire de détecter le minimum global de cette fonction alors que la courbe de biais ne présente qu'une importance secondaire. Il serait même dangereux de se concentrer sur la courbe d'apprentissage car une erreur très faible d'apprentissage est souvent associée à un sur-apprentissage important.

⁸⁸Phénomène présent lorsque la courbe de biais diminue et la courbe de variance augmente

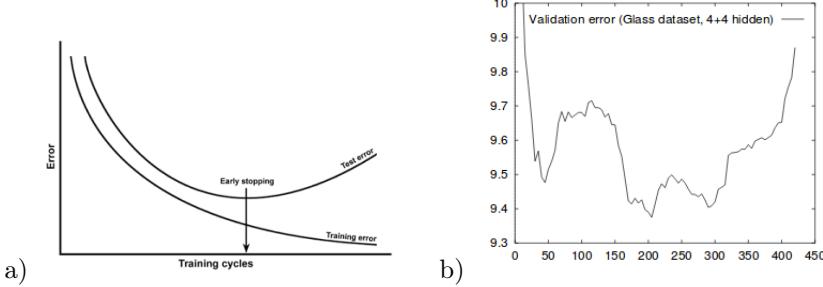


Figure 30: a) Early Stop théorique b) Comportement-type de l'erreur de validation en situation réelle

Sur la Figure 23, la courbe de variance est idéalisée. Dans les faits, elle présente souvent du bruits et des oscillations locales qui rendent son étude délicate (Figure 30). La problématique du minimum local est au coeur du problème et il n'existe pas de solution idéale actuellement pour garantir un arrêt optimal de l'apprentissage du modèle. Néanmoins, plusieurs approches existent et présentent des résultats satisfaisants. Ces fonctions de régulation sont appelées **Early Stopping**.

5.10.1 Early Stopping

Les méthodes *Early Stopping* reposent sur la valeur de l'erreur de validation. De ce fait, nous définissons $E_{opt}(t)$, l'erreur de validation la plus faible jusqu'à l'instant t , définie par: $E_{opt}(t) = \min_{t' \leq t} (E_{va}(t'))$ avec E_{va} , erreur de validation.

5.10.1.1 Generalization Loss (GL_α)

La première méthode, nommée Generalization Loss (GL_α)[83], repose sur l'augmentation relative de l'erreur à l'instant t par rapport à $E_{opt}(t)$. Ainsi, si l'augmentation est supérieure à une valeur donnée, l'apprentissage est stoppé. Ce critère examine la tendance absolue de l'évolution de l'erreur de validation et néglige la tendance évolutive locale.

Pour formaliser mathématiquement ce comportement, nous définissons l'erreur (en pourcentage) par:

$$GL(t) = 100 * \left(\frac{E_{va}(t)}{E_{opt}(t)} - 1 \right)$$

$$Stop : GL(t) > \alpha$$

Cette approche est peu efficace pour lutter contre les minimum locaux. Exploiter ce critère d'arrêt aura tendance à stopper l'apprentissage lorsqu'un minimum local aura été observé. Bien qu'efficace en cas de comportement convexe de la

fonction d'erreur, elle est peu performante dans le cas contraire. Expérimentalement, ce critère limitera la capacité d'exploration du réseau durant sa phase d'apprentissage et peut favoriser un arrêt prématué.

5.10.1.2 Quotient of Generalization Loss and Progress (PQ_{alpha})

La méthode GL_α ignore le comportement du modèle sur les données d'apprentissage. En ignorant ces informations, ce critère n'a pas connaissance de la situation d'apprentissage du réseau. La variation de l'erreur d'apprentissage peut donc être faible ou élevée sans qu'elle n'ait d'impact sur le critère d'arrêt. Expérimentalement, on observe que le sur-apprentissage arrive souvent lorsque l'erreur d'apprentissage décroît "lentement" après une diminution brutale en début d'apprentissage (Figure 31). De même, intuitivement, il est pertinent de penser qu'une variation importante de l'erreur d'apprentissage est corrélée avec une amélioration significative du réseau ou tout du moins, une variation significative de son comportement prédictif. L'impact de cette variation sur le comportement prédictif doit être considéré avant d'imposer l'arrêt de l'apprentissage. Pour considérer cet aspect, la notion de *Progress* (P_k)[83] est introduite. Elle évalue l'importance de la variation de l'erreur moyenne sur un intervalle donné par rapport à l'erreur minimale observée.

Supposons un intervalle de k itérations successives et $E_{tr}(t)$, erreur d'apprentissage à l'instant t alors:

$$P_k(t) = 1000 * \left(\frac{\sum_{t'=t-k+1}^t E_{tr}(t')}{k * \min_{t'=t-k+1}^t E_{tr}(t')} - 1 \right)$$

Nous voulons poursuivre l'apprentissage si l'erreur d'apprentissage s'améliore significativement. De ce fait, la valeur de P_k doit minorer l'expression de $GL(t)$. Pour cela, nous définissons PQ_α [83] tel que:

$$Stop : \frac{GL(t)}{P_k(t)} > \alpha$$

5.10.1.3 Successive Generalization Error (UP_s)

Les méthodes précédentes exploitent une approche absolue pour l'analyse de la tendance de la courbe d'erreur. Un procédé complémentaire serait de considérer les tendances locales de la courbe d'erreur. Pour cela, UP_1 observe la variation de E_{va} pour un intervalle $[n, n+k]$ donné et provoque un arrêt de l'apprentissage si la variation correspond à une augmentation de l'erreur de validation.

En généralisant, nous obtenons:

$$UP_1 : stop : E_{va}(t) > E_{va}(t - k)$$

$$UP_s : stop : UP_{s-1} \rightarrow stop \cup E_{va}(t) > E_{va}(t - k)$$

Remarque: Il est important d'observer le comportement récursif de UP_s où s indique le nombre de variations successives considérées.

Cette méthode favorise la capacité d'exploration du modèle en l'émancipant de la considération absolue de sa performance⁸⁹. UP_s est donc plus performant pour détecter le meilleur minimum local que GL_α . Néanmoins, cette approche facilite le risque de divergence du modèle et peut être inefficace en cas de variations progressives.

Supposons $UP_{s=4}$. Si la courbe d'erreur de validation suit un cycle itératif défini par 3 hausses positives⁹⁰ de l'erreur (e_1^+, e_2^+, e_3^+) suivie d'une diminution de l'erreur (e_4^-) tel que $\sum_{i=1}^4 e_i^{+-} > 0$, alors le critère d'arrêt est inefficace et la fonction d'erreur divergera.

Ces deux familles de *Early Stopping* peuvent être combinées afin d'exploiter les deux axes d'analyse de la tendance de l'erreur de validation. Il est donc possible de réaliser une combinaisons linéaire de plusieurs de ces critères et de pondérer les critères selon l'importance qu'on souhaite leur attribuer.

5.10.2 Arrêt supervisé

L'apprentissage d'un réseau de neurones est (généralement) une tâche de longue durée qui rend possible une supervision visuelle. Exploiter des outils de visualisation durant l'apprentissage est une approche à ne pas sous-estimer et souvent, présentera les meilleurs résultats.

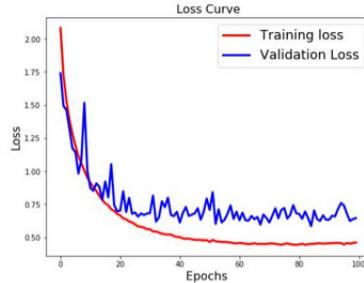
Les API de Deep Learning proposent des interfaces de visualisation pour faciliter ce travail, notamment *Tensorboard* associé à l'API de Google, *Tensorflow*.

5.11 Data Augmentation

Une des difficultés principales des réseaux de neurones est l'exigence d'une quantité importante de données d'apprentissage, quantité qui augmente avec la profondeur du réseau. Il n'est pas rare d'utiliser des jeux d'apprentissage contenant des millions voire des milliards d'entités, notamment dans l'étude du langage (Natural language Processing).

Obtenir un jeu de données est difficile, chronophage (et potentiellement cher si il doit être annoté dans le cas du traitement du langage ou de la segmentation d'image). Il est donc indispensable d'utiliser des méthodes afin, à partir d'une entité, d'en créer d'autres de manière artificielle. Il est possible de créer tout type de données de manière artificielle à partir d'une base suffisante le permettant (texte, signal 1D, 2D, ...). Chaque type de données utilise des méthodes

⁸⁹Le critère d'arrêt se limite à observer ses évolutions locales et non absolues
⁹⁰ou moins



On observe que la courbe d'erreur d'apprentissage est comparable à un logarithme inversé. Sur cet exemple, le sur-apprentissage ne semble pas présent.

Figure 31: Exemple de courbe d'erreur d'apprentissage et de validation

différentes propres à son type.

Ainsi, dans le cas d'une image (signal 2D), plusieurs approches sont possibles:

- **Variation géométrique:** Rotation, inversion, symétrie, translation
- **Analyse de densité de la distribution de pixels (via l'histogramme):** contraste, filtrage, effet flou
- **Analyse statistiques:** Création par PCA, Création par ZCA
- **Signal bruité:** Ajout de bruit gaussien, bruit poivre et sel, bruit multiplicatif
- **Création de contenu:** Création de patterns génériques (fusionner deux photos par "collage" brut par exemple - efficace dans le cadre de la segmentation)

Il existe de nombreuses autres méthodes et ce, pour tout type de données. Un jeu d'apprentissage est de qualité si il est le plus représentatif et diversifié possible. Cette étape est donc importante car un jeu de données de qualité est une **condition nécessaire et primordiale** pour un apprentissage de qualité. **Peu importe la qualité du modèle, si les données sont de médiocre qualité, l'apprentissage sera mauvais.**

5.12 Complexité de l'architecture et mémoire

Le Deep Learning a connu un regain de popularité grâce à l'évolution matérielle qui, dorénavant, supporte ce type d'architecture. Néanmoins, les réseaux très profonds demandent des ressources très importantes qui ne sont pas à la portée de tous (particulier comme professionnel). Un modèle complexe demande un

temps très important d'apprentissage et le temps de prédiction peut être trop important pour supporter les conditions du temps réel (analyse vidéo ou vocale par exemple). De plus, une contrainte importante de mémoire dédiée peut avoir lieu dans le cadre de structures limitées telle que l'embarqué. Il est donc important de proposer des approches pour optimiser l'architecture d'un réseau. En effet, il est **important** de ne pas associer la profondeur d'un réseau avec une preuve de qualité. Un réseau très profond peut être moins performant qu'un réseau plus simple bien que sa capacité d'abstraction soit plus forte. Les difficultés d'apprentissage sont un facteur déterminant et il est souvent plus aisés d'optimiser un réseau plus petit qu'affronter la complexité d'un modèle théoriquement plus performant mais plus délicat à développer.

Pour cela, différentes méthodes de simplification de modèle existe, notamment *Deep Compression*[29]. Cette méthode reprend l'idée de suppression des poids faibles issue du *Dense-Sparse-Dense training*. L'objectif de cette compression est de limiter le temps d'apprentissage du modèle (de ce fait, sa complexité) et la mémoire nécessaire pour le stocker. L'approche *Sparse* limite la complexité du modèle et l'optimisation de la mémoire est réalisée en deux étapes. Nous ne détaillerons pas ces deux étapes en détails, pour cela, veuillez vous référer au papier associé. *Deep Compression* se déroule ainsi:

1. **Sparse:** Suppression des poids faibles du réseau
2. **Optimisation des poids et des gradients:** Afin de limiter le stockage de valeurs, une approche par clustering (Kmeans) est réalisée afin de réaliser des clusters de poids semblables. Ces poids seront donc, dans un premier temps, représentés par la valeur du centroïde⁹¹ du cluster (souvent associé au barycentre). Les matrices du réseau seront donc associées à des index pointant sur la valeur du centroïde concerné. Les gradients de chacun des poids sont calculés, additionnés entre eux selon la répartition de leurs poids associés parmi les clusters définis précédemment et les gradients de chaque cluster multipliés par le pas d'apprentissage. La valeur des centroïdes sera alors mise à jour par la soustraction de la valeur du centroïde avec celle du gradient du cluster associé.
3. **Application du codage de Huffman:** Afin de limiter le poids des matrices de poids (ou d'index), on applique un codage de Huffman. Le codage de Huffman permet de réaliser une compression de données sans perte.

Un exemple illustratif de la phase 2 est visible sur la Figure 32. Il existe d'autres méthodes bien que ce genre d'approche soit, aujourd'hui encore, expérimental et peu appliquée dans l'industrie. Cependant, il est fort probable qu'elles se développeront dans le futur du fait de la complexité grandissante des modèles créés.

⁹¹L'initialisation du Kmeans est une étape importante à étudier et à ne pas négliger

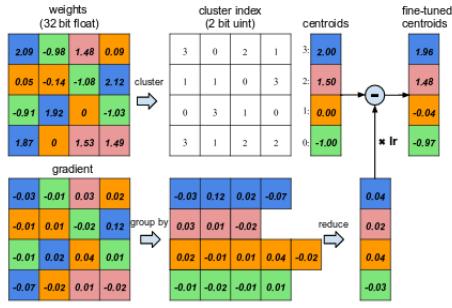


Figure 32: Optimisation réalisée sur les matrices de poids par Deep Compression

5.13 Transfer Learning

Important: Cette partie va supposer que vous avez une connaissance générale des **réseaux convolutifs**. Si non, veuillez vous référer à la Section 12. On supposera le cas d'un réseau convolutif pour illustrer cette partie.

L'une des grandes problématiques du Deep learning est son temps d'apprentissage. Même sur des modèles jugés simples, l'apprentissage demande des ressources matérielles importantes et ce, pour une longue durée. L'objectif du *Transfer Learning* est d'étudier les capacités d'un apprentissage à se généraliser à différents problèmes. Par exemple, un réseau convolutif ayant appris sur un jeu de d'images spécifiques peut-il être réutilisé sur un autre jeu d'images ? Si oui, dans quelle mesure ? Cette problématique est l'un des sujets de recherche parmi les plus actifs avec des retombées applicatives très importantes. Cette fonctionnalité est très importante dans le cas d'analyse d'image ou de texte (projection vectorielle de mots ou Words Embeddings)

Dans le cadre du Deep Learning, l'ensemble des approches de Transfer Learning sont exploitées. Chaque approche possède ses propres familles de réseaux et spécificités.

5.13.0.1 Inductive Transfer - Parameter Transfer

La méthode *Parameter Transfer* est particulièrement populaire, notamment pour les réseaux convolutifs. La problématique sous-jacente à cette approche revient à définir si des poids d'un réseau peuvent être utilisés par un autre réseau pour son apprentissage.

Classiquement, les couches de convolution d'un réseaux convolutifs peuvent être interprétées comme des extracteurs d'attributs d'une image, i.e extraire l'information utile et discriminante. Ces attributs sont alors différenciables par un réseau Feed-Forward standard. Il est évident que le modèle Feed-Forward

est entraîné à discriminer les entités selon le jeu de données d'apprentissage. Il n'est pas pas ou très peu ré-utilisable dans d'autres contextes⁹². Mais il est logique de se demander si extraire des attributs d'une image est généralisable ou spécialisé d'un jeu de données particulier.

Les couches de convolution agissent comme des couches d'abstraction de l'image. Les premières couches isolent grossièrement les attributs de l'image et les couches finales, analysent ces attributs de manière plus spécifique et détaillée. Ainsi, plus le modèle est profond, plus sa représentation finale des données aura un haut niveau d'abstraction spécifique aux données d'apprentissage et ses couches finales seront très affiliées au données d'apprentissage (couches généralistes → couches spécialisées). Les couches de convolution peuvent ainsi être généralisées car l'extraction d'attribut d'une image n'est pas nécessairement associée à une catégorie de données spécifiques (notamment dans les premières couches de convolution) car très généralistes.

Pour illustrer, supposons que nous souhaitons différencier deux voitures. Tout d'abord, nous considérerons la taille, la couleur dominante par exemple, i.e le comportement des premières couches de convolution alors que par la suite, nous étudierons la forme des jantes ou des rétroviseurs. Analyser la taille et la couleur peut être utile pour discriminer deux chats mais analyser la forme des jantes est peu efficace... Ainsi, le *Transfer Learning* permet de limiter le "gaspillage" d'apprentissage mais des précautions doivent être prises selon les spécificités du modèle qu'on souhaite réutiliser et les nouvelles données d'apprentissage.

Aujourd'hui, il existe des jeux de données variées et de grande qualité (ImageNet, Cifar,...). Ces jeux de données sont très diversifiées (avec des centaines voire des milliers de classes). Ils sont donc généralistes⁹³. De nombreux modèles pré-entraînés sur ces données sont en open-source et facilitent grandement le temps d'apprentissage.

Trois possibilités s'offrent à l'utilisateur:

1. **Extracteur figé:** Dans cette configuration, on juge que les couches exportées sont fiables et finalisées. On figera donc leurs poids en bloquant les modifications par rétropropagation. Dans le cas d'une extraction de couches de convolution, la sortie obtenue par ces couches est appelée **CNN codes**.
2. **Extracteur variable:** Dans cette configuration, les couches exportées ne sont pas figées et continuent d'apprendre durant la spécialisation du réseau sur le nouveau jeu de données. Le *Transfer learning* agit donc comme une initialisation des poids du réseau (bien plus pertinent que l'aléatoire).

⁹²Un réseau qui a appris à discriminer un chat ne pourra discriminer une voiture

⁹³Si on cherche à étudier des entités d'un même domaine d'activité. Etudier des images de microbiologie à partir d'un modèle ayant appris sur des images issues d'expériences en astrophysique sera difficile quelque soit la diversité de l'apprentissage sur son domaine.

3. **Modèle pré-entraîné:** Bien qu'il y ait des modèles puissants réalisés (en général) par des universités et/ou des entreprises privées, une communauté importante s'est développée pour favoriser la création de modèles déjà pré-entraînés. Ces modèles sont souvent moins "performants" que les modèles réputés et reconnus du fait de la différence de ressources et de temps d'apprentissage du réseau. Néanmoins, il est possible de trouver des réseaux ayant appris sur un jeu de données plus semblables au votre que les modèles standards ayant appris sur des données généralistes. Ces modèles offrent donc une alternative très performante notamment dans le cadre de l'initialisation de poids. On parlera alors de *Fine-Tuning*.

Le choix de l'approche à choisir dépend essentiellement de la taille du nouveau jeu de données et de leurs différences.

- **Nouveau jeu de données similaire au jeu d'apprentissage et de petite taille:** A cause de la petite taille du nouveau jeu de données, le risque de sur-apprentissage est important. Comme le jeu de données d'apprentissage et proche du nouveau jeu de données, il est préférable de figer les couches de convolution et d'apprendre uniquement un nouveau modèle Feed-Forward pour la classification.
- **Nouveau jeu de données similaire au jeu d'apprentissage et de grande taille:** Grâce à la grande taille du jeu de données, il est possible de mettre à jour les poids des couches transférées car le risque de sur-apprentissage est plus faible. Le Transfer Learning joue un rôle d'initialisation des poids dans cette configuration et l'apprentissage spécialisera votre modèle.
- **Nouveau jeu de données de petite taille et très différent du jeu d'apprentissage:** Cette situation est peu propice au Transfer Learning. En effet, le jeu d'apprentissage de petite taille ne permet pas un apprentissage des poids performant, il est donc nécessaire de limiter la mise à jour des poids des couches transférées. Cependant, la différence importante entre les deux jeux de données présente un risque de mauvaise performance due à la sur-spécialisation de ces couches. Afin de trouver un compromis, il est intéressant de ne pas garder **l'intégralité** des couches de convolutions mais d'en garder un sous-ensemble en supprimant les dernières couches qui présentent la spécialisation la plus forte. Ainsi, nous transférons que les couches les plus généralistes qu'on peut supposer performantes pour différencier des images de catégories différentes. Bien sûr, un nouveau modèle Feed-Forward doit être appris intégralement.
- **Nouveau jeu de données différent du jeu d'apprentissage et de grande taille:** Dans cette configuration, la grande taille du jeu de données permet un apprentissage. Néanmoins, la différence entre les deux jeux pose problème quant à la quantité des couches à transférer. Il est souvent préférable, grâce à la capacité d'apprentissage du nouveau jeu de données, d'utiliser un réseau pré-entraîné (autres que les modèles standards) qui a

apris sur des données comparable aux vôtres afin d'initialiser les poids et d'utiliser vos données pour le spécialiser en mettant à jour les couches transférées. Un nouveau modèle Feed-Forward doit être appris.

Important: Le modèle Feed-Forward n'est pas impératif. En effet, d'autres algorithmes d'apprentissage automatique peuvent être utilisés notamment le SVM ou le XGboost. Néanmoins, les réseaux de neurones présentent une bonne efficacité et permettent une implémentation plus aisée en permettant la réalisation d'un modèle qui limite les dépendances de concepts.

Il est très difficile de considérer l'ensemble des caractéristiques d'un phénomène, c'est pourquoi la capacité de généralisation d'un modèle de *Machine Learning* est très importante. Il n'est pas capital de représenter l'intégralité des hypothèses possibles⁹⁴ mais il est nécessaire de survoler l'ensemble des grandes possibilités de représentation. Plus ce travail de diversité sera fait, plus le jeu de données sera de qualité. Il est donc évident qu'un jeu de données de grande taille aura tendance à être de meilleure qualité. Néanmoins, ce n'est pas une condition suffisante !

5.13.0.2 Transductive Transfer - Domain Adaptation - A FAIRE

5.14 Réseaux de neurones et méthodes d'Ensemble - A FAIRE

5.14.1 Bagging - A FAIRE

5.14.2 Boosting - A FAIRE

⁹⁴C'est d'ailleurs un travail irréalisable

6 Réseaux convolutifs

Important: Tout ce qui a été vu précédemment s'applique à la spécificité des réseaux convolutifs.

6.1 Généralités

Les réseaux convolutifs constituent l'une des grandes familles d'architecture associées aux réseaux de neurones. Popularisés par Yann Lecun en 2012, ces réseaux ont montré des résultats remarquables pour l'analyse de données structurées telles que les images (mais aussi les signaux ou textes). Mais pourquoi cette architecture est préférable à un modèle Feed-Forward (Perceptron Multicouche) standard ?

Supposons une image de petite taille soit 15×15 par exemple. L'image est codée en RGB (images couleurs). Ainsi, l'image possède $15 \times 15 \times 3$ valeurs distinctes correspondant à ses pixels. Dans le cas d'un modèle Feed-Forward, les neurones de la couche d'entrée devront avoir une entrée (et donc un poids) associée à chacune des valeurs de l'image soit $15 \times 15 \times 3 = 675$ entrées. Bien que ça puisse impressionner, les capacités de calculs des technologies d'aujourd'hui peuvent le supporter⁹⁵. Considérons une image de taille un peu plus standard soit 500×500 . On obtient donc $500 \times 500 \times 3 = 750 \times 10^3$ poids par neurones de la couche d'entrée. Il est évident que le nombre d'entrées est bien trop important et rendrait l'apprentissage irréalisable. De même, le risque de sur-apprentissage est très important. De ce fait, les modèles Feed-Forward classique ne **peuvent pas se mettre à l'échelle**.

Les couches de convolution permettent d'extraire l'information d'une image et d'en réaliser une représentation à un plus haut niveau d'abstraction. Ces couches présentent deux avantages notables. Tout d'abord, elles permettent de représenter le contenu utile d'une image dans une dimension plus faible⁹⁶ que l'image d'origine⁹⁷, ce qui permettrait à un modèle Feed-Forward d'apprendre dessus. Dans un second temps, les couches de convolution réalisent le travail d'extraction d'information réalisée traditionnellement par des méthodes-tiers comme HoG par exemple et ce, avec une capacité d'apprentissage. Alors que les méthodes-tiers sont des méthodes figées et généralistes, les couches de convolution se spécialisent dans la discrimination des images de sa base d'apprentissage. Ainsi, elles sont souvent plus efficaces et mieux optimisées⁹⁸. Une illustration est visible sur la Figure 33

Un réseau convolutif possède (dans sa version générale) l'architecture suivante:

⁹⁵Bien sûr, il ne doit pas y avoir trop de couches cachées...

⁹⁶Après traitement d'une couche de pooling

⁹⁷On peut comparer ce procédé à un pré-traitement des données

⁹⁸Il a été montré qu'elles offrent une capacité de discrimination assez généraliste dans les couches de bas niveau malgré tout

Convolutional Neural Network

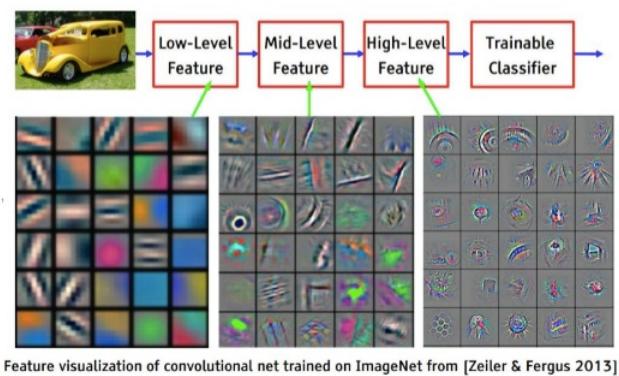


Figure 33: Relation entre l'abstraction d'un réseau convolutif et sa profondeur

- **Couche d'Entrée:** Ceci correspond à la donnée proposée au réseau pour son apprentissage ou une prédiction. L'entrée stocke ainsi l'intégralité des valeurs de la matrice de l'image. Elle sera donc de dimension $X*Y*3$ dans le cas d'une image de dimension $X*Y$ en RGB.
- **Couche d'extraction de caractéristiques:**

Le cycle suivant est répété k fois en accord avec l'architecture du réseau.

1. **Couche de convolution:** Les couches de convolution vont extraire l'information de l'image et la représenter à un plus haut niveau d'abstraction. Il peut y avoir une ou plusieurs couches de convolution. La nature des sorties d'une couche de convolution se présente sous la forme d'une matrice de dimension $x*y*z$ où x,y,z dépendant des paramètres de la couche⁹⁹.
2. **Couche d'activation:** Cette couche réalise une activation selon la fonction d'activation utilisée sur les valeurs de la matrice de sortie de la couche de convolution. ReLU est la fonction la plus populaire actuellement mais d'autres peuvent être utilisées.
3. **Couche de régulation:** Afin de limiter le sur-apprentissage, il est utile d'exploiter des couches qui aident à prévenir cette dérive de l'apprentissage. Ces couches n'ont pas de pouvoir d'extraction

⁹⁹ x et y inférieurs ou égaux à la dimension X et Y de l'entrée

d'informations et se limitent à une action régulatrice. Ce type de couche modifie localement l'architecture du réseau (DropOut par exemple) ou les valeurs de sortie des couches de convolution (Batch Normalisation par exemple).

4. **Couche de redimensionnement:** Les sorties des couches de convolution peuvent être dans un format qui ne facilite pas l'apprentissage et/ou l'optimisation de ressources matérielles. Pour limiter ce problème, un redimensionnement vers une dimension inférieure (ou supérieure) est souvent réalisé. Le redimensionnement, selon l'objectif souhaité, s'applique selon une dimension spécifique du signal donné.

- **Couche de classification (Full-Connected i.e perceptron multicouche):** Cette couche réalise la décision du réseau à partir de l'information extraite des couches de convolution.

Il est important de noter que certaines couches possèdent des paramètres et d'autres non. En effet, les couches ReLu et Pooling appliquent une transformation fixe sur les données (donc aucun paramètre¹⁰⁰) alors que les couches de convolution et Full-connected "apprennent"¹⁰¹ durant l'apprentissage et donc, possède des paramètres.

6.2 Couche d'Entrée

La couche d'Entrée est représentée par une couche de neurones où un neurone se limite à la propagation d'une valeur de la matrice de l'image observée. Chaque neurone est ainsi défini par une pondération unitaire avec une fonction d'activation linéaire. Il y a autant de neurone que de pixels dans l'image (d'où l'importance d'avoir des images de taille constante).

6.3 Couche de Convolution

6.3.1 Nature d'une couche de convolution

Les couches de convolution suivent une architecture Feed-Forward modifiée par la condition de connectivité locale, i.e un neurone de la couche de convolution n'est pas connecté à l'intégralité des neurones de la couche précédente (Entrée ou autre couche de convolution) mais uniquement aux neurones "les plus proches". Cette particularité diminue drastiquement le nombre de liaison des neurones, permettant leur apprentissage en temps humain. Le degrés de connectivité locale est un paramètre variable dépendant de la taille du filtre souhaité. Un filtre détermine le nombre de neurones adjacents à considérer. Plus le filtre sera grand, plus il y aura de neurones considérés. Néanmoins, bien que la taille du

¹⁰⁰Ceci n'est pas entièrement vrai pour les couches de Pooling car il existe des variantes qui exploitent des paramètres. De même, des variantes plus avancées de la fonction ReLU apprennent durant l'apprentissage

¹⁰¹Par rétro-propagation

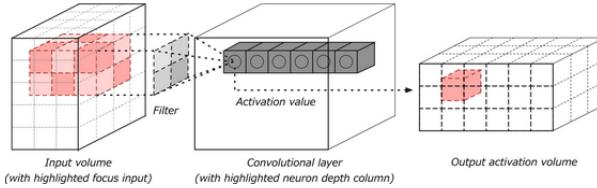


Figure 34: Architecture d'une couche de convolution

filtre soit variable, sa profondeur reste constante et égale à la profondeur de l'image d'entrée. Par exemple, supposons une image RGB. Sa profondeur est de 3 (channels rouge, vert et bleu). Chaque neurone possédera une connexion avec chacun de ces trois channels en accord avec les limitations du critère de connectivité locale. Il y a donc une restriction selon la largeur et la longueur de la donnée d'entrée mais pas sa profondeur. Une couche de convolution possède un filtre de dimension fixe mais peut posséder plusieurs sous-couches associées à des filtres différents (même dimension mais valeurs différentes). L'architecture d'une couche de convolution se représente donc comme un volume en 3 dimensions où la profondeur correspond au nombre de sous-couches. Une représentation graphique est visible sur la Figure 34.

Chaque neurone d'une même sous-couche de convolution possède un même paramétrage. Ainsi, chaque neurone possède le même biais et le même vecteur de pondération. Ils sont donc tous *identiques*. Cette spécificité implique une *invariance par translation* car, comme chaque neurone est identique, la localisation spatiale des pixels n'a pas d'influence sur l'analyse par un filtre donné. De ce fait, une couche de convolution limite grandement le nombre de paramètres tout en conservant les corrélations spatiales locales des images.

Un filtre représente le vecteur poids du neurone. Les poids des neurones ne sont pas identiques pour chaque channel. Ainsi, dans le cas d'une image RGB, un filtre possédera 3 sous-filtres qui seront appliqués sur chacun des channels de manière indépendantes. Les 3 résultats obtenus par les sous-filtres seront unis lors du calcul de la convolution. Dans la configuration standard, on considère souvent un biais nul et une fonction d'activation identité.

Du fait de l'utilisation d'une architecture de neurones, l'apprentissage des filtres se fait par rétro-propagation. Grâce à la connectivité locale du réseau et de l'unicité des pondération par filtre, le nombre de paramètres à apprendre est drastiquement diminué et permet au réseau d'apprendre en temps humain.

6.3.2 Interprétation graphique

En vulgarisant, une convolution est une opération mathématique qui décrit une règle sur comment unir deux données. Une convolution prend ainsi une entrée,

applique un kernel et réalise une *feature map* en sortie.

Dans le cadre d'une couche de convolution, la convolution est réalisée par un produit scalaire entre un filtre (kernel) et l'entrée. Un exemple est visible sur la Figure 36. Le filtre est de dimension inférieure à l'entrée (dans le cadre d'une image, inférieure à hauteur*longueur) mais de profondeur égale à la profondeur de l'image (nombre de channels d'entrée). Un filtre se comporte comme une *fenêtre glissante* afin de parcourir l'intégralité de la données d'entrée. La matrice de sortie obtenue est appelée *feature map*. Il y a autant de *feature map* que de filtre ainsi, si il y a 4 filtres appliquées sur la donnée d'entrée, il y aura 4 *feature map* en sortie.

La *fenêtre glissante* est associée à un hyperparamètre appelé **stride**. En effet, il est possible de définir les intervalles de déplacement. Un stride de 1 implique de réaliser une convolution en appliquant le filtre sur chaque valeur, un stride de deux impliquera un saut de une valeur entre chaque calcul de convolution, etc... Un exemple est visible sur la Figure 35. Le stride est très utilisé pour diminuer la dimension des couches de convolution, notamment pour limiter le nombre d'entrée de la couche Feed-Forward qui réalise la décision.

Important: Le kernel est appliqué sur une valeur de la matrice d'entrée en son **centre**, pas l'un de ses sommets. De plus, un filtre ne réalise la convolution que si chaque élément du filtre s'applique à une valeur de la matrice d'entrée. Observons la Figure 37. Sur la représentation de gauche, chaque entité du kernel est associée à une valeur. La convolution peut être définie. A droite, le kernel "dépasse". Le filtre ne peut donc intégralement s'appliquer à la matrice d'entrée: la convolution n'est pas réalisée.

Cette particularité soulève une problématique. Dans cette configuration, la sortie obtenue après convolution ne peut être que de dimension inférieure à l'entrée à cause des effets de bord, effet qui s'aggrave avec des filtres de grandes tailles. Bien qu'une sortie de dimension inférieure ne soit pas un problème en tant que tel (ce serait même bénéfique d'un point de vue temps-machine), elle dénonce une perte d'information de l'entrée sur ses bords. Supposons un kernel de dimension $K \times K$ et une entrée $D \times D$, alors la dimension de la sortie sera de dimension $(D-K+1) \times (D-K+1)$ ¹⁰².

Afin de traiter ce problème, la méthode du **padding** est appliquée. Cette méthode consiste à appliquer une valeur arbitraire et constante afin de combler les valeurs manquantes de la matrice d'entrée. Par convention, la valeur utilisée est 0 (d'où le nom standard de 0-padding¹⁰³). Un exemple est visible sur la Figure 38. Cette approche permet de considérer l'information de l'intégralité de la donnée d'entrée et de créer une sortie de même dimension que l'entrée.

¹⁰²On supposera un stride de 1

¹⁰³Parfois, la valeur 1 est employée

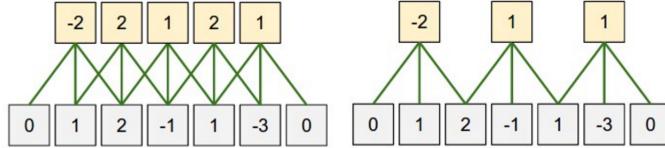


Figure 35: Impact du stride sur la sortie de la convolution

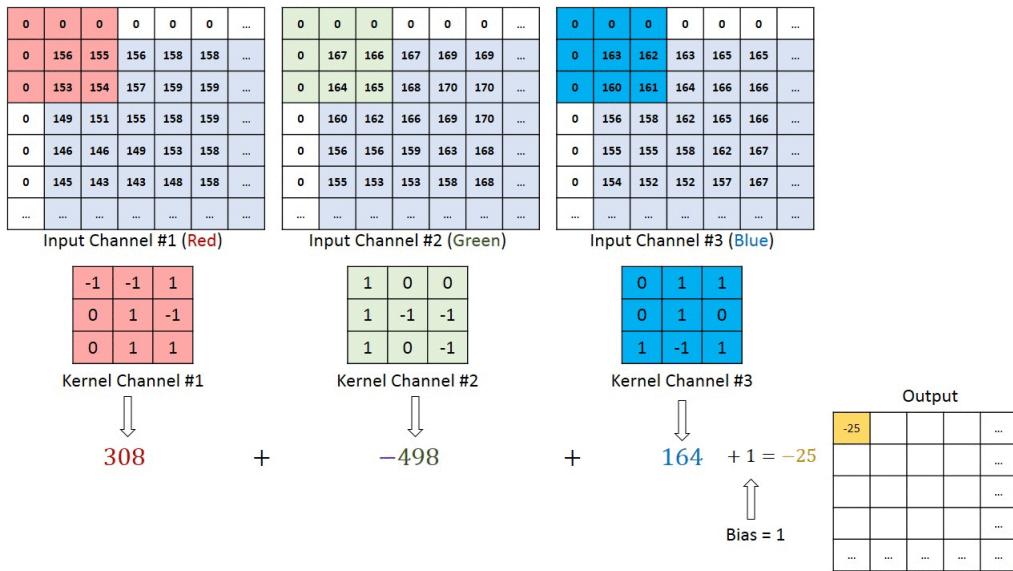


Figure 36: Exemple d'une convolution par filtre

Le site Web <https://ezyang.github.io/convolution-visualizer/index.html> propose une application intuitive pour visualiser l'impact de différentes configurations sur le comportement des filtres.

En pratique, il faut éviter les filtres de grande taille (5×5 , 7×7 et supérieurs) sauf si il y a une vraie pertinence dans le choix de cette échelle de filtre. Les filtres de grande taille sont plus lents à apprendre, gourmand en ressource machine et peuvent être remplacés par des alternatives plus légères avec une efficacité similaire. Il est donc préférable de cumuler des filtres 3×3 dans ce type de cas. Cette approche simple demande moins de paramètres, moins de ressources machine et surtout, plus de non-linéarité.

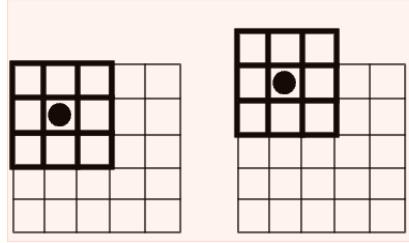


Figure 37: Problématique de la dimension du filtre: gauche (valide), droite (invalidé)

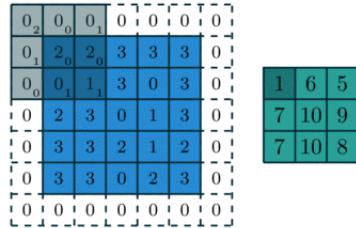


Figure 38: Application du 0-Padding

6.3.3 Couche de Pooling

Le *Pooling* est une méthode permettant de compresser (et généraliser) une information locale de la donnée d'entrée. Elle est employée afin de diminuer la dimension des *feature map* tout en conférant une capacité de généralisation à la couche. En effet, le Pooling conserve l'information locale importante, permettant de s'émanciper de détails trop spécifiques (et du bruit).

Le fonctionnement d'une couche de Pooling est comparable à celle d'une couche de convolution si ce n'est qu'elle s'applique sur une *feature map* indépendamment des autres (la profondeur du kernel est de 1). Un filtre (de taille choisie) est appliquée afin de définir *l'environnement local*¹⁰⁴ et un opérateur est appliqué sur les valeurs d'entrée ciblées par le filtre. La nature de cet opérateur est variable: Max, Average¹⁰⁵. Expérimentalement, la fonction Max est la plus utilisée. Sur la Figure 39, nous pouvons observer l'application d'un Max-Pooling de filtre 2*2.

Remarque: Le Pooling doit être exploitée avec parcimonie. Il est souvent un *bottleneck* pour la vélocité du modèle. De même, son utilisation ne fait pas l'unanimité dans le milieu scientifique. En effet, Geoffrey Hinton, chercheur de renom dans le domaine de l'apprentissage profond, a déclaré: "The pooling op-

¹⁰⁴Comparable à une isolation spatiale

¹⁰⁵Ces deux opérateurs sont les plus fréquents

eration used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster." (Reddit AMA).

Ce scepticisme repose sur plusieurs défauts de cette approche. Le Pooling est une action non évolutive (fixe donc sans apprentissage), ce qui nuit à la capacité de généralisation/spécialisation. De plus, son action est locale à une *feature map*. Il y a donc une perte importante d'informations associées aux relation inter-feature map. Pour finir, le Pooling a une capacité explicative lorsque les couches de convolutions présentent un fort taux d'abstraction. Dans le cas contraire, le filtre est souvent de taille trop faible pour considérer une information discriminante. Augmenter la taille du filtre est envisageable mais le Pooling étant *destructeur*, l'augmentation du filtre aurait des conséquences désastreuses sur la transmission de l'information aux couches supérieures.

Plus récemment, des propositions de fonctions de Pooling ont été faites. Nous pouvons citer:

- **L_p Pooling**[44]: Cette approche est issue des études biologiques des cellules vivantes. Elle est définie par: $y_{m,n,k} = \left[\sum_{(i,j) \in R_{mn}} (a_{i,j,k})^p \right]^{\frac{1}{p}}$ avec R_{mn} , surface du filtre et k, channel analysé.

Il s'agit de la norme L_p des valeurs ciblées par le filtre du Pooling. Ainsi, si $p=1$ alors L_p correspond à l'average Pooling. Si $p = \infty$, L_p correspond au max Pooling. Cette méthode permet de réaliser un compromis entre l'isolation discriminante d'une information et la généralisation du message locale.

- **Mixed Pooling**[127]: Cette méthode combine max Pooling et average Pooling au sein d'une combinaison linéaire dont les coefficients pondérateurs dépendent d'un paramètre λ tel que $\lambda \in \mathcal{U}(0,1)$. Ainsi, nous obtenons:

$$y_{m,n,k} = \lambda \max_{R_{mn}} a_{i,j,k} + (1 - \lambda) \frac{1}{|R_{mn}|} \sum_{(i,j) \in R_{mn}} a_{i,j,k}$$

Expérimentalement, elle présenterait de meilleurs résultats que l'approche max ou average tout en proposant une plus grande résistance au surapprentissage.

- **Gated max-average Pooling**[127]: Cette approche est similaire à *Mixed Pooling* mais considère l'état de la surface couverte par le Pooling. En effet, *Mixed Pooling* applique un coefficient indépendamment des valeurs couvertes par le filtre de Pooling. *Gated max-average Pooling*, au contraire, va réaliser une pondération évolutive selon les valeurs couvertes par le filtre. On suppose x, valeurs couvertes par le filtre de Pooling et w, "Gate" du Pooling. Cette idée est traduite par:

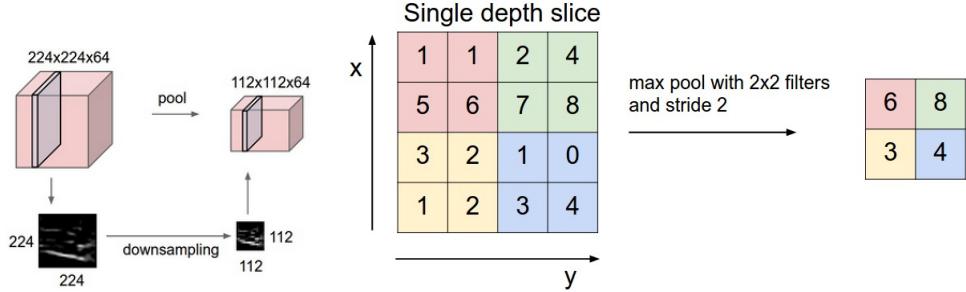


Figure 39: Application du Pooling: Réduction et exemple avec la fonction max selon un filtre 2×2

$$f_{gate} = \sigma(w^T x) f_{max}(x) + (1 - \sigma(w^T x)) f_{avg}(x)$$

$$\sigma(w^T x) = \frac{1}{(1 + \exp(-w^T x))}$$

La "Gate" peut être unique à un réseau, une couche, à une région de la couche mais identique sur toute la profondeur ou au contraire, différente sur toutes les profondeurs. Néanmoins, cela ajoute des paramètres à apprendre et donc un ralentissement de la vitesse d'apprentissage. Cette approche présente des résultats plus performants que *Mixed Pooling*.

- **Stochastic Pooling**[131]: Cette méthode choisit une valeur selon une distribution de probabilité définie par $p_i = \frac{a_i}{\sum_{k \in R_{mn}} a_k}$ avec $i \in [1, |R_{mn}|]$ et R_{mn} , l'ensemble des valeurs isolées par le filtre du Pooling.

Ainsi, plus une valeur est élevée, plus la chance d'être choisie est importante. Cette méthode permet de limiter le sur-apprentissage comparée au max-Pooling en introduisant un comportement probabiliste sur la sélection de la valeur.

Remarque: Dans les faits, ces méthodes de Pooling sont assez marginales et peu exploitées. L'écrasante majorité des réseaux actuels exploitant des approches par Pooling reposent sur l'opérateur Max ou Average.

L'opérateur Max est utile lorsque l'on cherche à extraire une information discriminante au détriment des autres. L'objectif est d'isoler les éléments les plus discriminants afin de les analyser par la suite (notamment par un réseau Full-Connected). Cette approche est très utile pour la classification d'images par exemple. En effet, pour classer, un élément discriminant est nécessaire et suffisant. D'autres données moins informatives risqueraient de se comporter comme

une forme de bruit.

L'opérateur Average ne se focalise pas sur les éléments discriminants du signal mais sur son comportement général. Sa valeur est donc un résumé global de ses caractéristiques et non d'un aspect discriminant spécifique. Cet opérateur est donc moins destructeur mais possède un pouvoir discriminant plus faible.

La couche de Pooling, selon la configuration, peut conserver des informations utiles pour d'autres couches. Par exemple, dans le cas du Max-Pooling, une mémorisation de la localisation de la valeur max sur la matrice de données peut être réalisée. Cette information permet de réaliser des mises à l'échelle (via Unpooling) avec plus de fidélité mais demande de la mémoire pour stocker les valeurs.

Remarque: Expérimentalement, on a tendance à éviter le chevauchement des filtres de Pooling. Ainsi, on exploite un stride en équation avec la dimension du kernel pour que chaque valeur de la *feature map* soit analysée une unique fois. Par exemple, dans le cadre d'un kernel 2×2 , on aura tendance à utiliser un stride de 2.

6.3.3.1 Global Pooling

Global Pooling est l'équivalent du pooling mais son application n'est pas limitée par la fenêtre correspondant à la taille du filtre. Il ne s'effectue donc pas sur un sous-ensemble (défini par le filtre) d'une *feature map* mais sur l'intégralité d'une *feature map*. Une *feature map* est donc représentée par une unique valeur à l'issu d'un Global Pooling. Ainsi, supposons une donnée de la forme $15 \times 15 \times 3$ avec 3, profondeur¹⁰⁶ de la sortie. La sortie sera donc représentée par un vecteur de dimension dans R^3 .

Le Global Pooling est une méthode utilisée pour lutter contre le sur-apprentissage et diminuer le volume des calculs des réseaux convolutifs, notamment de la couche Full-connected réalisant la classification. Les capacités de régulation du Global Pooling s'expriment sous différents aspects:

- **Sans apprentissage:** Étant donné que le Global Pooling considère l'intégralité d'une *feature map*, il n'y a pas d'apprentissage associée à cette couche, ce qui n'ajoute pas de temps d'apprentissage
- **Optimisation de la relation Label - Feature Map:** La couche de Full-connected réalise la prédiction à partir des *feature map*. Les *feature map* sont donc les porteuses d'informations qui doivent retranscrire les spécificités d'un label (catégorie). Il existe un risque important que les *feature maps* "brutes" soient trop spécifiques du fait des dimensions matricielles élevées de leurs structures, ce qui provoque des dérives vers le

¹⁰⁶Nombre de *feature map* présents dans la sortie considérée

sur-apprentissage. L'approche du Global Pooling, en limitant à une valeur par *feature map*, favorise la correspondance entre une *feature map* et une caractéristique générale.

- **Généralisation Spatiale:** En résumant une *feature map* à une valeur, le Global Pooling permet de généraliser l'information. Cette méthode permet donc d'extraire une caractéristique indépendamment de considérations spécifiques due à l'image d'entraînement telles que l'angle de vue, le contraste ou encore la luminosité par exemple.
- **Temps d'apprentissage:** Le Global Pooling limite grandement le nombre de valeurs d'entrée de la couche Full-connected. Cette méthode permet donc d'en diminuer la taille, ce qui rendra le modèle plus rapide à apprendre, à calculer ses prédictions et plus léger d'un point de vue mémoire.

Le Global Pooling est une approche de plus en plus utilisée pour réaliser la liaison entre l'ensemble des couches de convolution et le réseau Full-Connected, remplaçant ainsi la méthode dite *Flatten* classique qui consiste à vectoriser les features map dans leur intégralité. Néanmoins, Global Pooling est très destructeur et présente des faiblesses notamment dans le cas de données textuelles (classification de textes par réseau convolutif par exemple).

6.3.3.2 k-Max Pooling

Afin de corriger le comportement destructeur du Global Pooling, k-Max Pooling[49] applique l'opérateur pour conserver les k valeurs¹⁰⁷ les plus importantes (au lieu d'une seule pour le Global Pooling). Du fait d'un résultat non unitaire, l'opérateur Average n'est pas exploitable par cette approche.

Le choix de la valeur de k peut être difficile à réaliser. Une valeur trop faible posséderait les mêmes faiblesses qu'un Global Max Pooling et une valeur trop élevée nuirait à la capacité de généralisation recherchée par l'exploitation de ce type de couche. De plus, comme le Pooling s'exerce sur une *feature map* uniquement, la profondeur est à considérer lors du choix de la valeur de k . En effet, en supposant n *feature map* en entrée, la sortie de la couche de k-Max Pooling sera de dimension $n * k$.

Les couches de convolution permettent d'extraire l'information utile d'un signal. Néanmoins, elles sont directement dépendantes de la qualité de la donnée en entrée. Une donnée trop bruitée provoquera un résultat médiocre peu importe la qualité du modèle employé. Une couche de Pooling mal exploitée peut être une source de contre-performance du réseau du fait de son grand pouvoir destructeur. Il est donc intuitif de penser que dans un réseau qui cumule plusieurs couches de Pooling, le pouvoir généralisant doit être progressivement exploité pour ne pas perdre d'informations en amont des couches convolutives.

¹⁰⁷Si k est égal à 1, le comportement est identique à un Global Max Pooling

En d'autres mots, dans le cadre du k-Max Pooling, k doit avoir une valeur plus importante au niveau des couches basses et une valeur plus faible au niveau des couches élevées.

Pour répondre à cette problématique, Dynamic K-Max Pooling [49] a été proposé. Cette approche propose une détermination automatique de la valeur de k en fonction de l'architecture du réseau et de la position de la couche de Pooling dans ce dernier.

La valeur de k est définie par: $k_l = \max(k_{top}, \lceil \frac{L-l}{L} * s \rceil)$ avec k_{top} , valeur minimale de k pour les couches supérieures, L, nombre total de couches de convolution dans le réseau, l, position de la couche où le Pooling est appliqué et s, dimension¹⁰⁸ de la *feature map*.

La fonction proposée permet d'avoir une diminution progressive de la valeur de k (jusqu'au seuil critique k_{top} et ainsi, d'augmenter le pouvoir discriminant des couches de Pooling en adéquation avec les caractéristiques informatives des données entrantes. La fonction proposée par [49] est "arbitraire" et ne repose sur aucune véritable justification mathématique si ce n'est l'intuition des auteurs. Elle peut être modifiée afin d'obtenir un comportement plus spécifique sur la vitesse de décroissance. De plus, elle est spécifique à une donnée textuelle. Dans le cadre d'une image, il est pertinent d'envisager une autre fonction qui sera en adéquation avec le comportement d'un signal 2D.

6.3.4 Couche de Unpooling

Le Unpooling est une approche de *Upsampling*, i.e l'augmentation de la dimension de représentation d'une données. Cette approche est utilisée après une une couche de pooling lorsqu'une mise à l'échelle est nécessaire. Elle exploite la capacité de la couche de pooling à conserver des informations de position¹⁰⁹ (notamment pour Max-Pooling). La configuration de la *feature map* de sortie dépend de la méthode de pooling choisie. Ainsi, dans le cas d'un max-pooling, seule la valeur à la position correspondant à la position de la valeur max sur la donnée d'entrée (bornée par la surface couverte par le filtre) sera définie, les autres valeurs étant considérées comme 0. Une illustration est visible sur la Figure 40.

L'exemple présenté est statique (réutilise les valeurs max sans apprentissage). Une approche similaire consiste à ne pas utiliser la valeur max de la *feature map* d'entrée et à appliquer un autre filtre, entraînable, afin de définir les valeurs de la nouvelle *feature map*. De ce fait, seules les informations de position sont

¹⁰⁸A l'origine, cette approche de Pooling a été proposée pour l'analyse de texte. De ce fait, s correspond à la dimension du texte en entrée. Il est possible de généraliser à un signal 2D comme un image mais il est probable que la fonction doive être modifiée pour mieux correspondre aux caractéristiques d'une image

¹⁰⁹Dans le cas de l'average-pooling, ce n'est pas nécessaire

Input	Max-pooling [3, 3]	Unpooling																																								
<table border="1"> <tr><td>0.8</td><td>0</td><td>3.3</td><td>0</td><td>4</td><td>1</td></tr> <tr><td>1.2</td><td>1.2</td><td>0</td><td>2.1</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>3</td><td>0.2</td><td>3.1</td><td>5.2</td><td>0</td></tr> </table>	0.8	0	3.3	0	4	1	1.2	1.2	0	2.1	0	0	0	3	0.2	3.1	5.2	0	Value: <table border="1"><tr><td>3.3</td><td>5.2</td></tr></table> Position: <table border="1"><tr><td>(0,2)</td><td>(2,1)</td></tr></table>	3.3	5.2	(0,2)	(2,1)	<table border="1"> <tr><td>0</td><td>0</td><td>3.3</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td><td>5.2</td><td>0</td></tr> </table>	0	0	3.3	0	0	0	0	0	0	0	0	0	0	0	0	0	5.2	0
0.8	0	3.3	0	4	1																																					
1.2	1.2	0	2.1	0	0																																					
0	3	0.2	3.1	5.2	0																																					
3.3	5.2																																									
(0,2)	(2,1)																																									
0	0	3.3	0	0	0																																					
0	0	0	0	0	0																																					
0	0	0	0	5.2	0																																					

Figure 40: Exemple de Max-Unpooling statique

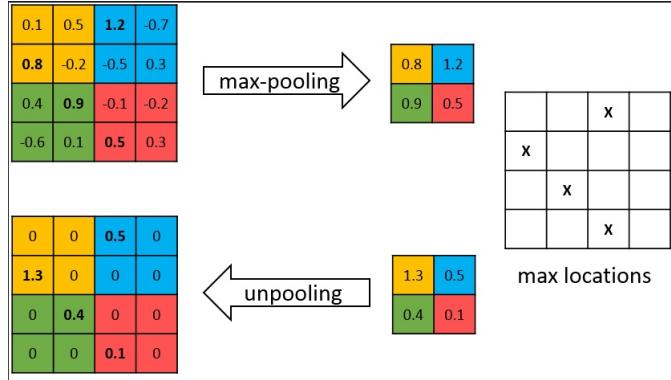


Figure 41: Exemple de Max-Unpooling dynamique

utilisées. Un exemple est visible sur la Figure 41.

6.3.5 Couche de convolution 1*1

Un cas particulier est la couche de convolution 1*1. Au premier regard, cette convolution semble non pertinente voire sans intérêt¹¹⁰. Néanmoins, du fait de la considération obligatoire de la profondeur de la donnée d'entrée, ce type de convolution possède un intérêt certain. En effet, elle a la capacité de modifier la profondeur de sortie de la couche de convolution où la profondeur correspondra au nombre de filtre présent dans la couche de convolution 1*1. Supposons une donnée d'entrée de dimension $X \times Y \times Z$ avec Z , profondeur de cette donnée. L'application d'une couche de convolution 1*1 avec N filtres créera une sortie de dimension $X \times Y \times N$. Cette méthode complète l'approche par pooling. En effet, le pooling limite la dimension de la *feature map* et la convolution 1*1 limite la profondeur de sortie, i.e le nombre de *feature map*.

Cette convolution ne se focalise que sur la valeur ciblée par le filtre sans considération des corrélations locales mais tient compte de l'ensemble des channels de la donnée d'entrée¹¹¹. Elle permet donc d'ajouter de la non-linéarité dans

¹¹⁰Pour les lecteurs sensibilisés au traitement du signal, c'est un quasi non-sens

¹¹¹La profondeur est toujours considérée dans son intégralité

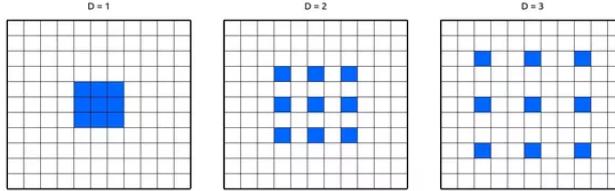


Figure 42: Exemple d'une convolution dilatée

l'architecture du réseau¹¹².

6.3.6 Couche de convolution dilatée

Les couches de convolution dilatées[128] introduisent un nouveau paramètre: **le taux de dilatation**. Le taux de dilatation détermine l'espace entre chaque valeur effective du kernel. Par exemple, un kernel $3*3$ avec un taux de dilatation de 2 deviendra un kernel de $5*5$ car le filtre ne se focalisera pas sur des éléments adjacents a_i, a_{i+1} mais sur des éléments distants en accord avec le taux de dilatation. Ainsi, pour une dilatation de 2, les éléments ciblés seront a_i, a_{i+2} . Les valeurs contenues dans le champ réceptif mais non effectives auront une valeur imposée à 0 (car le poids associé sera nul). La Figure 42 illustre ce type de convolution.

Cette approche permet d'élargir le champ d'analyse de la couche en conservant le même coût de calcul. Elle est très utilisée pour la détection d'entités ou la segmentation car elle offre un bon compromis de performance. Elle permet une analyse générale d'une image sans multiplier les convolutions ou l'utilisation de filtres larges qui sont très gourmands en ressources. Par exemple, supposons un filtre $3*3$ (stride 1) classique. En combinant 3 couches composées de ce filtre, le champ réceptif de la dernière couche sera de $7*7$ ($3 * 3 \rightarrow 5 * 5 \rightarrow 7 * 7$). Au contraire, avec une dilatation de 2, la surface couverte sera de $15 * 15$.

6.3.7 Couche de convolution transposée

Remarque: Cette couche est aussi appelée *fractionally strided convolutions* ou *déconvolution*. L'appellation "déconvolution" est un abus de langage. Cette couche ne réalise pas une déconvolution au sens mathématique du terme.

Une couche de convolution transposée a pour objectif de mettre à une dimension **supérieure** choisie, une donnée d'entrée. Il s'agit ainsi d'une technique de *Upsampling*. C'est très utilisé en segmentation afin de mettre à l'échelle la sortie d'un réseau convolutif.

¹¹²Ce qui est souvent favorable aux performances du réseau

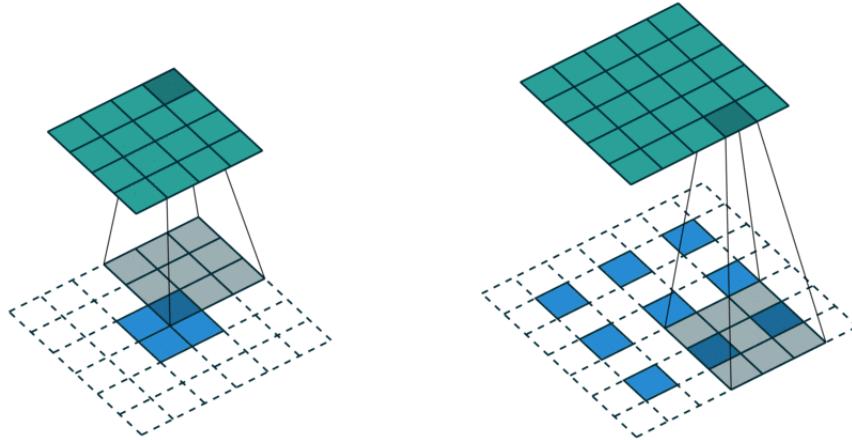


Figure 43: Exemple de convolution transposée: Gauche: entrée 2×2 sans stride, Droite: Entrée 3×3 avec stride de 1

Son fonctionnement est comparable à une convolution standard avec l'application de padding (ajout de 0 arbitraire) afin de compléter la donnée d'entrée et mettre à l'échelle voulue la *feature map* de sortie. Le padding dépend de la taille du filtre choisie, du stride et de la dimension de sortie du *feature map*. Deux exemples sont visibles sur la Figure 43.

Les couches de convolution transposées permettent uniquement de conserver les dimensions des features map. Ainsi, supposons une entrée N alors l'application d'une couche de convolution sur cette entrée puis d'une couche de déconvolution (avec le même paramétrage de couche que la couche de convolution) permettra d'obtenir une sortie de même dimension que N. Par contre, la sortie n'est **pas égale à N** ! D'où l'abus de la dénomination *déconvolution* qui est faux mathématiquement parlant.

6.3.8 Couche de Depthwise/Pointwise

Bien qu'efficaces, les couches de convolution standards restent gourmandes en temps de calcul et en mémoire nécessaire pour stocker le modèle. Dans des contextes particuliers avec des limitations de ressources tels que l'embarqué ou encore les smartphones/tablettes, l'utilisation de modèle très profond classique n'est pas envisageable pour cause de limitation matérielle. En plus des méthodes standards de réduction vues dans les Sections précédentes, il est possible d'agir au niveau de l'architecture-même des couches de convolutions avec les couches de **Depthwise/Pointwise** (ou separable depthwise). Ce type de couche a été popularisé par la création de l'architecture *MobileNet*[36] créée par Google afin de répondre aux problématiques des prédictions sur smartphone.

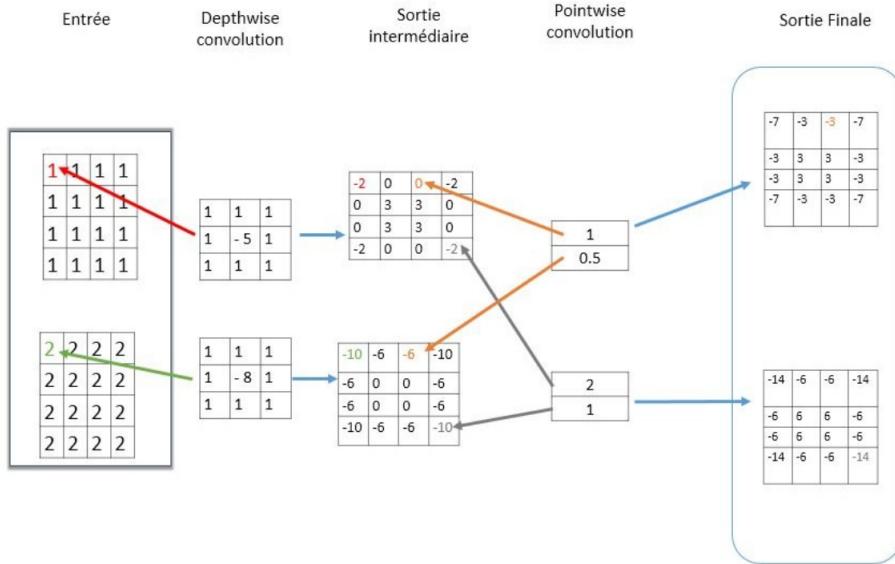


Figure 44: Exemple d'une couche Depthwise/Pointwise avec 2 filtres

Le comportement général de ce type de couche est comparable aux couches de convolution standards. La différence se situe au niveau des spécificités du nombre de filtres et de leurs dimensions. Une couche de depthwise/pointwise se découpe en deux parties: la partie depthwise et la partie pointwise. La couche depthwise est comparable à une couche de convolution standard mais avec **uniquement** un filtre. Dans une couche de convolution depthwise/pointwise, il n'y a qu'une couche depthwise. La couche pointwise est comparable à une couche de convolution standard avec un filtre de dimension 1×1 et s'applique sur la sortie *intermédiaire*¹¹³ de la couche depthwise. Cette couche peut être cumulée N fois. Une illustration de cette couche est visible sur la Figure 44.

Étudions dorénavant les bénéfices de ce type d'approche. Supposons une entrée (matrice carrée) de dimension $X \times X \times M$ avec M, profondeur de l'entrée. On lui applique N filtres (composés de sous-filtres en accord avec la profondeur de la donnée d'entrée). On supposera que chaque filtre possède une dimension identique de taille $K \times K$. Le coût de calcul d'une couche de convolution standard est donc de $(X \times X) \times M \times (K \times K) \times N$ ¹¹⁴.

Considérons maintenant une couche depthwise/pointwise. On a alors:

¹¹³Avant l'étape d'addition par rapport à une convolution standard

¹¹⁴On suppose un stride de 1 et un 0-padding nécessaire pour conserver la même dimension entre les données d'entrée et la *feature map* de sortie

$$\begin{aligned}
D_{depthwise/poitnwise} &= D_{depthwise} + D_{pointwise} \\
D_{depthwise} &= (X * X) * M * (K * K) * 1 \\
D_{pointwise} &= (X * X) * M * (1 * 1) * N \\
D_{depthwise/poitnwise} &= (X * X) * M * (K * K) * 1 + (X * X) * M * (1 * 1) * N
\end{aligned}$$

La différence de coût de calcul est donc définie par:

$$\frac{(X * X) * M * (K * K) * 1 + (X * X) * M * (1 * 1) * N}{X * X) * M * (K * K) * N} = \frac{1}{N} + \frac{1}{K^2}$$

Ce résultat permet d'évaluer la pertinence de ce type de couche. En effet, on peut observer que plus le nombre de couche est importante (N) et/ou la taille du filtre grand (K), plus l'économie en temps de calcul est important. Cette économie est associée au temps de calcul machine et au temps d'apprentissage, le rapport étant identique pour les deux économies. Ainsi, pour les réseaux très profonds sur des supports de faible puissance, cette approche propose une solution performante.

Néanmoins, cette approche donne des résultats moins performants bien que ce défaut soit négligeable face au temps machine et d'apprentissage. L'utilisation ou non de ce type de couche relève des spécificités du problème à résoudre. Il est intéressant de noter que cette approche a été très sollicitée (et avec succès) sur les supports mobiles comme les smartphones.

6.4 Conversion d'une couche Full-Connected en couche de convolution

Une couche Full-Connected, par un "jeu de couches", peut être remplacée par une approche exclusivement convolutive. Bien que cette approche puisse sembler *tricky*, elle est très utile afin d'optimiser certains réseaux qui exigent un calcul de prédiction rapide en limitant la redondance de calcul (algorithmes de *Tracking* par exemple).

Observons la Figure 45. En haut, un réseau convolutif classique avec deux couches Full-Connected en fin de réseau finalisées par un *Softmax*. Une sortie *Softmax* est caractérisée par une couche composée de neurones comportant autant de sortie que de classes discriminées suivie d'une activation par une fonction *Softmax*. Dans notre exemple, supposons que le réseau discrimine 4 classes, i.e 4 neurones sur la couche de sortie.

La première couche de Full-Connected est connectée à une sortie de dimension $5*5*16$. Un neurone possède donc $5*5*16$ entrées distinctes. Observons le réseau entièrement convolutif (réseau du bas sur la Figure 45). La couche

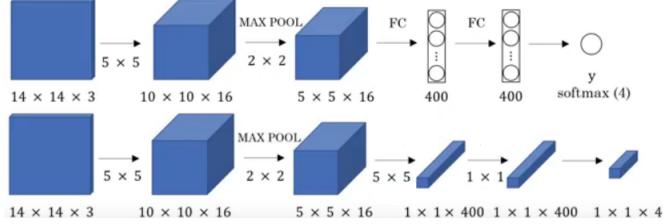


Figure 45: Exemple d'une conversion de couche Full-Connected vers une structure convolutive

Full-Connected est remplacée par une couche convective de kernel 5×5 et de profondeur 400. La sortie de cette couche sera donc de dimension $1 \times 1 \times 400$. Cette couche de convolution est donc déterminée par 400 neurones où seul un neurone caractérise un filtre. Un kernel, par convention, s'applique sur toute la profondeur. Ainsi, chaque neurone de la couche de convolution reçoit $5 \times 5 \times 16$ (hauteur * largeur * profondeur) valeurs pondérées par des poids distincts (et apprenants). Par conséquent, ces deux architectures sont mathématiquement identiques car le logit formé est issu d'une relation linéaire similaire et la couche présente autant de neurones.

Afin de réaliser la conversion, il faut donc créer une couche de convolution dont la dimension du kernel est égale à la dimension (hauteur et largeur) des features map en entrée. Le nombre de filtre de cette couche de convolution est égale au nombre de neurones de la couche Full-Connected.

6.5 Convolutional Neural Network (CNN) et Fully Convolutional Network (FCN)

Avec l'évolution de la Recherche, une famille d'architecture a été créée: *Fully Convolutional Network*[99] (FCN).

Contrairement à l'architecture CNN standard, FCN n'utilise pas de couche Full-Connected mais exclusivement des couches convolutives¹¹⁵. La prise de décision du réseau est donc assurée par des couches convolutives tout comme l'extraction d'attributs.

Un réseau FCN va donc essayer d'extraire les informations et de prendre une décision selon des données **spatialement indépendante** du fait de l'isolation réalisée par le fonctionnement de la convolution. Chaque *sous-décision* de la couche convective est basée sur l'information isolée par la taille du filtre. Au contraire, CNN aura tendance à considérer l'information dans sa **globalité** du fait de l'utilisation de couches Full-Connected qui exploite intégralement les

¹¹⁵D'où le nom Fully Convolutional Network

données. Il n'y a donc pas de considération de *localité*. Ces dernières années, les réseaux FCN gagnent en popularité du fait de leur efficacité similaire et de leur plus grande vitesse d'apprentissage. De même, un FCN permet une plus grande robustesse face à des données de dimensions variables¹¹⁶ tout en réalisant la conversion d'échelle plus rapidement.

6.6 Les modèles convolutifs de référence

6.6.1 AlexNet

AlexNet[61](2012) est une référence historique du Deep Learning. Il est le modèle précurseur des architectures convolutives (reposant sur l'architecture LeNet développée par Lecun) et a présenté des résultats remarquables en gagnant le 2012 ILSVRC¹¹⁷ (ImageNet Large-Scale Visual Recognition Challenge) avec 10% d'erreur en moins que le 2nd. Il constitue un cas d'école pertinent pour apprendre les fondations d'un réseau convolutif (couche de convolution, max-pooling, dropout, ReLu...) et a été défini pour classer 1000 catégories au plus. Du fait de son ancienneté, il tend à devenir "obsolète" aujourd'hui.

6.6.2 ZF Net

ZF Net[132](2013) est un modèle basé sur AlexNet et vainqueur du 2013 ILSVRC. La différence majeure se situe sur la taille des filtres, notamment des premières couches. L'idée derrière cette réduction était de limiter un maximum la perte d'information associée à un filtre trop grand (trop généraliste) de la donnée initiale. De plus, il n'a été entraîné que sur 1.3 million d'image et non 15 millions comme AlexNet.

L'apport majeur de ce modèle (et du papier de recherche associé) est la méthode de visualisation DeConvNet (Deconvolutional Network). Cette méthode permet d'examiner les caractéristiques d'activation des différentes couches et d'offrir un aperçu de ce que "voit" et discrimine la couche observée. Cette architecture a permis de mieux comprendre le comportement interne des réseaux convolutifs et leurs critères de discrimination¹¹⁸. On le nomme DeConvNet du fait de l'inversion de la transformation réalisée. Au lieu de passer de "pixels à map feature", ce réseau passe de "map feature à pixels".

6.6.3 VGG Net

VGG Net[101](2014) est un finaliste (mais perdant) de 2014 ILSVRC. Ce réseau exploite une architecture "simple" mais profonde. En effet, ce modèle se limite à cumuler des couches de convolution utilisant des filtres 3*3 - stride 1

¹¹⁶Les couches Full-Connected sont dépendantes de la dimension de la donnée d'entrée.

¹¹⁷Il s'agit d'une des compétitions - voire la compétition - la plus prestigieuse et compétitive de vision par ordinateur

¹¹⁸On est encore loin de comprendre parfaitement le fonctionnement des réseaux neuronaux. C'est toujours un sujet de recherche très intense et d'une nécessité prioritaire

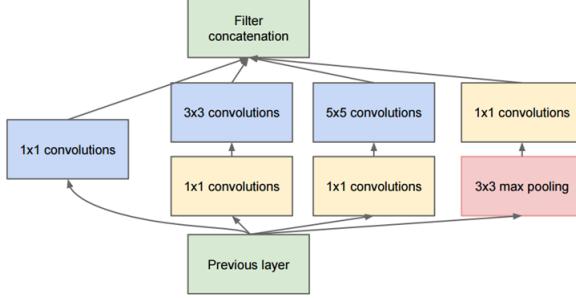


Figure 46: Architecture de l’Inception module

et des couches de max-pooling 2×2 - stride 2. La contribution principale de ce modèle a été de montrer que la profondeur est un composant critique de bonne performance. Néanmoins, c’est un modèle lourd (en ressources matérielles et paramètres) bien qu’il puisse être grandement simplifié en supprimant des couches de Full-connected sans une perte significative de performance. Il s’agit d’un modèle encore très utilisé aujourd’hui.

6.6.4 GoogleNet/Inception

GoogleNet[109](2014) est le vainqueur du 2014 ILSVRC. L’architecture de ce modèle est très novatrice (elle s’inspire, néanmoins, de l’approche Network in Network[67])¹¹⁹. Elle s’oppose au dogme du modèle séquentiel (succession de couches à flux unitaire) et propose une approche en "largeur" (en parallèle). Ce modèle a ainsi proposé une nouvelle forme de *block*: l’**Inception module**. Sa structure est représentée sur la Figure 46.

L’Inception module se démarque par la mise en parallèle de différentes couches de convolution appliquées sur une même source d’entrée dont les *feature map* produites sont **concaténées** à la fin du bloc. Cette mise en parallèle des couches permet de créer différents *flux* indépendants et différenciés les uns des autres. Chaque block peut avoir sa propre topologie, ce qui rend le modèle difficile à implémenter du fait de cette liberté. Évaluer la topologie d’un block est délicat et relève essentiellement de l’intuition. Il est important de relever la présence de convolution 1×1 . Une approche naïve serait de les supprimer, leurs importances semblant être secondaires. Néanmoins, ce serait une erreur majeure. En effet, ces convolutions possèdent deux caractéristiques critiques dans le cadre de ce module:

- **Réduction de dimension:** Les convolutions 1×1 possèdent la capacités de réguler la profondeur de la sortie d’une couche. Cette capacité est

¹¹⁹Nous n’aborderons pas ce papier. Il a une importance théorique mais n’apporte pas d’élément applicatif concret dans le cadre de ce cours

primordiale dans le cas de l’Inception module afin d’éviter une explosion inévitable de la dimension de sortie. Elle permettent ainsi de maîtriser la profondeur souhaitée d’une des branches du modules en sortie pour la concaténation (branche 3×3 max pooling) ou de diminuer la profondeur de la donnée d’entrée sur les branches 3×3 convolutions et 5×5 convolutions. En effet, l’application de filtre volumineux est gourmand en temps machine. Réduire la dimension de la donnée à observer favorise la vélocité du modèle. Les convolutions 1×1 permettent ainsi de réguler la dimension des données internes au module en maîtrisant les dimensions de sortie et en optimisant les dimensions de la données d’entrée pour favoriser un bon rapport performance/temps.

- **Non-linéarité:** Les convolutions 1×1 sont suivies d’une activation ReLu. De ce fait, elles permettent la mise en place de non-linéarité dans le module. Les couches de convolution étant linéaires dans leurs formes fondamentales, cet ajout ne peut être que bénéfique.

Cette architecture a eu plusieurs améliorations. La plus moderne est l’Inception-v4[108] qui propose une approche liant l’architecture Inception et Resnet.

6.6.5 Xception

Le modèle Xception[12](2016) a été inventé par François Chollet, chercheur de Google à l’origine du framework *Keras*. Xception propose une approche *extrême* de l’approche *Inception* d’où son appellation.

Afin de comprendre l’idée proposée par François Chollet, revenons sur le modèle Inception standard. Sur la Figure 47, nous pouvons observer deux block Inception. A gauche, un block arbitraire (topologie variable) et à droite, un block simplifié avec une structure uniforme. Supposons un block de N flux. Par un jeu de manipulation, nous pouvons remplacer les N couches de convolution 1×1 indépendantes en une couche 1×1 unique de dimension $N \times P$ avec P profondeur des sorties des couches 1×1 indépendantes¹²⁰. Les *feature map* de la couche 1×1 unitaire seront alors divisés en N parties sans chevauchement et alimenteront les différents flux de manière indépendante. Cette approche peut être poussée à *l’extrême* en associant un flux à une unique *feature map*. Une représentation graphique est visible sur la Figure 48. Le postulat du modèle Xception est une hypothèse plus forte que l’hypothèse d’Inception: il suppose que les corrélations inter-channels (profondeur) et les corrélations spatiales (largeur et longueur d’une *feature map*) peuvent être complètement traitées séparément et non par sous-groupes.

Cette approche isolante est très similaire à une convolution Depthwise/Pointwise. Une différence majeure existe: une couche de Depthwise/Pointwise réalise une convolution standard pour créer les *feature map* intermédiaires qui seront

¹²⁰On supposera les sorties uniformes

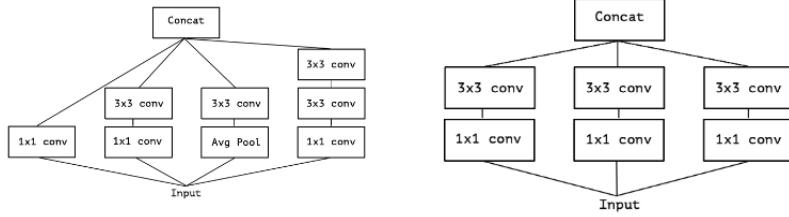


Figure 47: Architecture de l'architecture Xception: à gauche, un block standard et à droite un block simplifié (topologie uniforme)

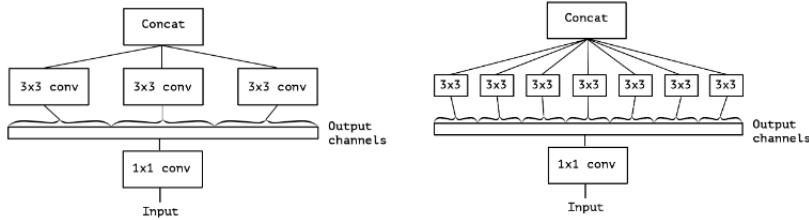


Figure 48: Architecture de l'architecture Xception: à gauche, un block Inception reformulé et à droite un block Inception dans sa forme extrême

alors traitées par des convolutions 1×1 . *Extreme Inception* réalise exactement l'inverse. Les créateurs du modèle ne justifie pas concrètement le réel impact de cette inversion si ce n'est par leur intuition. De plus, dans un block Inception, une activation ReLu est appliquée entre les différentes couches afin d'appliquer de la non-linéarité, ce n'est pas le cas avec les couches Depthwise/Pointwise. Cette spécificité permet de mettre en avant un résultat remarquable. Alors que la non-linéarité est bénéfique dans le cas du block Inception standard[111], elle semble être néfaste pour les blocks Extrem Inception avec Depthwise/Pointwise[12]. Ce résultat a été vérifié expérimentalement. Il est difficile de justifier clairement ce résultat mais l'idée principale repose sur la différence de profondeur. En effet, dans le block Inception standard, la profondeur de la données d'entrée dans un flux est profonde alors que dans un block Extrem Inception, la *feature map* est unitaire. L'hypothèse est donc qu'appliquer une régularisation sur une donnée peu profonde provoque une perte d'informations trop importante.

La spécificité de la couche Depthwise/Pointwise permet ainsi de créer un réseau très simple d'implémentation. En effet, un block Extreme Inception n'est donc rien d'autre qu'une couche de Depthwise/Pointwise. L'architecture générale est standard et ne présente pas de particularité notable si ce n'est l'utilisation des *résidus* (voir Section 6.6.6 pour plus de détails sur cette notion).

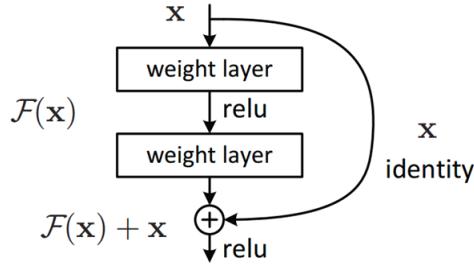


Figure 49: Architecture du Residual Block

6.6.6 Microsoft Resnet

Resnet[34](2015) est le vainqueur de 2015 ILSVRC. Son taux d'erreur est exceptionnel (3,6%). Un homme, en fonction de son expertise, obtient un taux d'erreur d'environ 5 à 10%. Cette architecture est donc la première à faire significativement mieux que l'homme dans le cadre de cette compétition. Cette architecture repose sur le **Residual Block**. Une illustration est visible sur la figure 49.

Un *Residual Block* est caractérisé par la non-perte de la donnée d'entrée. Ainsi, la sortie de ce block calcule un changement de faible ampleur (un "Delta") afin d'obtenir une version légèrement altérée de la donnée d'entrée. Pour conserver la donnée d'entrée, une concaténation est réalisée à la sortie du block où la sortie des couches internes est **additionnée**¹²¹ à la donnée d'entrée¹²². Cette architecture favorise un bon apprentissage en facilitant la propagation d'un gradient "valide", notamment en luttant contre le problème de *vanishing gradient*. L'idée derrière la notion de *Résidus* est qu'il est plus facile d'approximer une fonction qui nullifie les résidus plutôt que d'approximer une fonction identité (Figure 50). Néanmoins, son efficacité théorique n'est pas démontrée malgré une efficacité empirique et expérimentale incontestable.

Important: Les architectures *state-of-the-art* reposent essentiellement sur ce type d'architecture¹²³. Il est souvent pertinent de considérer ce type de réseau convolutif lorsqu'un choix par défaut s'impose.

6.6.7 ResNet et améliorations

Du fait de l'efficacité des réseaux Resnet, la recherche est active autour de cette architecture. Plusieurs améliorations ont été proposées:

¹²¹Au sens strict du terme, non concaténée

¹²²En cas de dimension différente entre l'entrée et la sortie des couches internes du bloc, du 0-padding et/ou des convolutions 1*1 sont utilisées

¹²³Plusieurs architectures peuvent être cumulées !

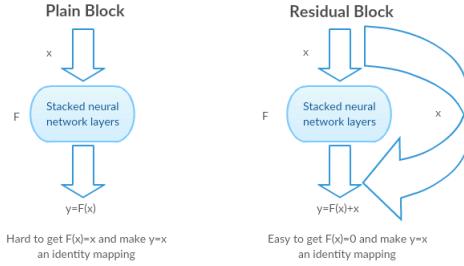


Figure 50: Intuition sur la notion de Résidus

6.6.7.1 Wide Residual Network

Wide Residual Network[129](2016) propose une architecture qui s'étend en "largeur"¹²⁴ et non uniquement en profondeur comme l'approche initiale des ResNets. La démarche principale consiste augmenter le nombre de filtres par couche ou implanter des couches spécifiques (dropout, 1*1 convolution,...). Ainsi, cette architecture crée plus de *feature map* en interne et possède un plus fort pouvoir de discrimination. Néanmoins, il peut y avoir une hausse du nombre de paramètres donc un temps de calculs et d'apprentissage supérieurs. Pour s'opposer à ce problème, la profondeur est minorée par rapport à la largeur. De plus, les capacités de calculs sont optimisées avec les approches en largeur car le calcul des différentes *feature map* au sein d'une même couche est fortement parallélisable, ce qui favorise la vélocité du modèle. Cette architecture propose aussi l'ajout de DropOut ou de convolutions 1*1 à l'intérieur des blocs. La plus-value de cet article est de montrer qu'un réseau ResNet ne doit pas être que profond mais favoriser son expansion en largeur aussi. Une illustration de cette architecture est représentée sur la Figure 51.

6.6.7.2 Aggregated Residual Transformations for Deep Neural Networks (ResNext)

Le réseau ResNext[122](2016) est le modèle arrivé 2nd au 2016 ILSVRC. Cette architecture propose de lier les caractéristiques des modèles ResNet et Inception. La principale caractéristique de ResNext repose sur son expansion en largeur avec l'ajout de flux internes dans un block. Sa différentiation majeure avec l'approche d'Inception est l'utilisation d'une topologie de bloc identique pour chaque bloc. La différenciation des blocks se fait au niveau du facteur de profondeur qui détermine le nombre de flux parallèles au sein d'un block.

¹²⁴Ne pas confondre ! Il y a une augmentation sur un unique flux de couches (ajout de filtres ou d'un DropOut par exemple) uniquement. L'augmentation se fait en largeur car elle ajoute des entités sur une couche ciblée au sein d'un même flux, non en largeur avec l'ajout de flux parallèles comme les modèles Inception !

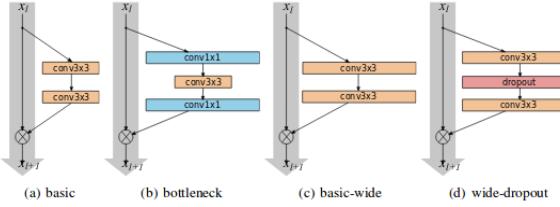


Figure 51: Architecture d'un Wide Residual block (les étapes de Batch normalization et de ReLu ne sont pas affichées) - la largeur des rectangles représentant les couches déterminent graphiquement le nombre de filtres employés pour la couche associée

L'architecture Resnext propose trois formes de blocks équivalents pour structurer son réseau.

- **Approche ResNet:** Avec cette approche, chaque flux réalise une harmonisation de la dimension indépendamment des autres flux via une convolution $1*1$. Les *feature map* issues des différents flux du block sont additionnées. La sortie est alors additionnée avec l'entrée du block.
- **Approche Inception:** Avec cette approche, l'harmonisation de la dimension est mutualisée. Ainsi, l'ensemble des *feature map* issues des flux sont concaténées sans application de convolution $1*1$ et produisent une sortie intermédiaire. L'application de la convolution $1*1$ est réalisée sur la sortie intermédiaire et les *feature map* obtenues sont additionnées à l'entrée du block.
- **Approche par convolution groupée:** Cette approche repose sur l'implémentation du réseau AlexNet qui a groupé ses couches afin de paralléliser son réseau sur différents GPU. Resnext propose cette méthode en considérant un flux comme un groupe. Ils peuvent donc être partagés sur différents GPU et favoriser la vitesse du modèle. Cette représentation est essentiellement théorique et peu exploitée dans les faits.

Une représentation graphique des différentes architectures du block ResNext est visible sur la Figure 52.

6.6.7.3 Densely Connected Convolutionnal Networks (DenseNet)

Densely Connected Convolutionnal Networks[41](2016) généralise la transmission de l'entrée des blocs ResNet en projetant l'entrée d'une convolution sur l'ensemble des convolutions suivantes et non juste le suivant direct. Ainsi, la n^{ieme} couche possédera une entrée issue de n sources. De même, la n^{ieme} couche transmettra sa sortie aux $(N-n)$ couches suivantes avec N , nombre total de couche. Il y a donc $\frac{N(N+1)}{2}$ ¹²⁵ connexions dans l'ensemble du réseaux issues

¹²⁵Équivalent à la somme de 1 à n

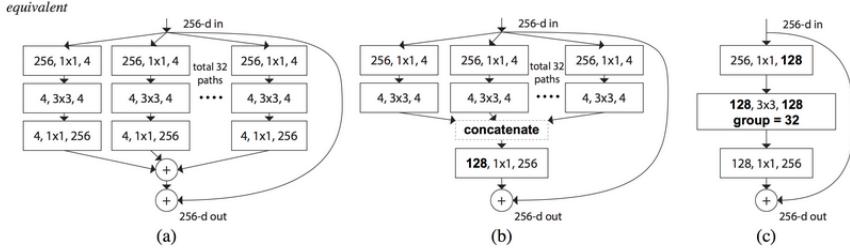


Figure 52: Différentes représentations de l'architecture d'un block du réseau ResNext: 1) Approche ResNet, 2) Approche Inception, 3) Approche par convolution groupée

des divers blocs au lieu de N d'où le nom de *Densely connected*. De plus, l'entrée est **concaténée** et non additionnée comme dans un bloc ResNet standard. La concaténation implique une augmentation constante de la profondeur de l'entrée transférée à chaque bloc. L'ajout de convolution $1*1$ et de pooling est donc une nécessité pour éviter l'explosion du nombre de données en entrée. Veuillez vous référer à l'article [41] pour plus de détails sur cette régulation.

Le réseau DenseNet, visible sur la Figure 54, est composé de plusieurs Dense block séparés par un block de transition formé d'une convolution $1*1$ et d'un Pooling afin de redimensionner les *feature map*. Ce redimensionnement pose problème car les *feature map* des block précédents ne sont plus de même dimension que les block qui suivent. De ce fait, la propagation se limite à un Dense block et non à l'ensemble du réseau.

La conversion de l'addition vers la concaténation apporte une plus-value importante. La concaténation préserve la structure de la *feature map* peu importe l'éloignement du bloc par rapport au bloc considéré. Cette différence permet de s'opposer au problème de corruption de l'information transférée ou de son effacement qui augmente alors que le nombre de *feature map* additionnées augmente linéairement. De plus, le modèle peut apprendre une combinaison optimale entre les features map concaténées, ce qui permet une meilleure interprétation de l'information utile.

Bien que paradoxale, cette approche permet de limiter le nombre de paramètres nécessaire pour créer le modèle. Dans un réseau neuronal, chaque couche lit une entrée donnée par la couche précédente associée et écrit à la couche suivante. La couche actuelle modifie donc l'état mais transmet aussi l'information importante à conserver. Cette information est conservée, dans les réseaux ResNet, par addition avec l'identité de l'entrée. Cependant, il a été montré que certaines couches contribuent peu voire produisent de l'information redondante (il se peut qu'elles redéfinissent une information perdue déjà calculée précédemment dans

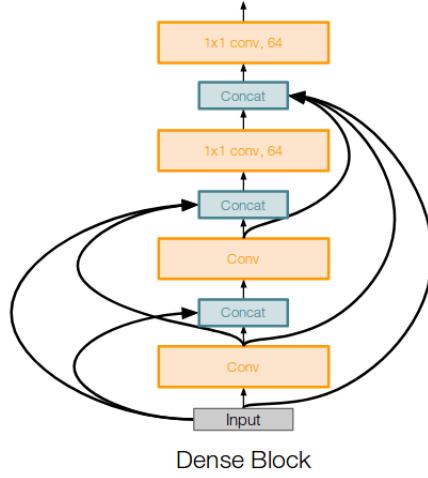


Figure 53: Architecture d'un réseau Densely Connected

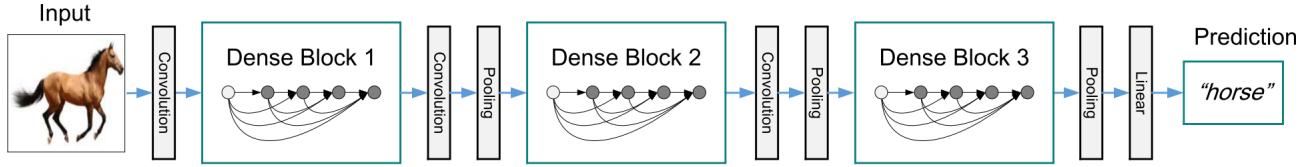


Figure 2. A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature map sizes via convolution and pooling.

Figure 54: Réseau DenseNet

le modèle) et donc, peuvent être supprimées. Afin de considérer cette particularité, la structure Densely connected propose une meilleure mémoire de la donnée utile et des modifications réalisées en projetant l’information sur l’ensemble des blocs et non uniquement le suivant. Cette propagation (et conservation) de l’information permet de limiter grandement le nombre de couches/blocs et ainsi de limiter le nombre de paramètres, rendant le modèle plus léger et rapide à apprendre. L’intuition derrière cette idée peut être contre-intuitive. Pour une explication détaillée, veuillez vous référer à l’article associé [41]. La structure d’un réseau Densely Connected est visible sur la Figure 53.

6.6.7.4 Stochastic Depth

Stochastic Depth[42](2016) met en relation le DropOut et les spécificités des *Résidual block* des réseaux résiduels. Afin de limiter l’inter-dépendance des couches du réseaux, les blocks sont conservés (sinon ils sont équivalents à une

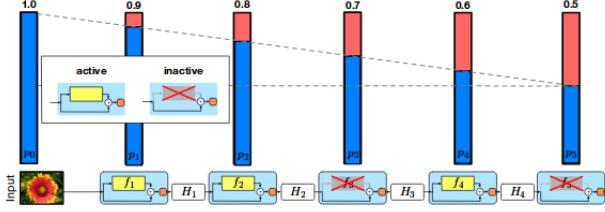


Figure 55: Stochastic Depth routine

couche *identité*) selon une probabilité p_l . p_l est une probabilité constante (peu optimale) ou linéaire dégressive telle que $p_l = 1 - \frac{l}{L} * (1 - p_L)$ (par défaut, p_L est égal à 0.5 et correspond à la valeur limite du dernier block considéré dans le réseau) avec L , nombre total de block et l , le block ciblé. Comme pour le DropOut, durant la phase de test, chaque sortie de residual block (avant union avec la valeur d'entrée du bloc) est active et pondérée selon la valeur du pourcentage d'activation associée. Un exemple illustratif est visible sur la Figure 55.

6.6.7.5 Residual Networks of Residual Networks: Multilevel Residual Networks

Residual Networks of Residual Networks[133](2016) propose une approche basée sur la structure de groupe. Ainsi, le réseau est composé d'une succession de groupes composés de block à la structure similaire. Le créateur du modèle a utilisé des convolutions $3*3$ uniquement de profondeur variable appartenant à $[16, 32, 64]$. Trois groupes sont donc formés en fonction de la largeur des couches. Il n'y a pas d'intersection de groupe, i.e chaque groupe est unique et les groupes sont homogènes. Supposons un réseau avec N block alors un groupe possèdera $\frac{N}{3}$ block. Ce réseau se différencie du ResNet en ajoutant une connexion inter-block pour chaque groupe. Ainsi, les *feature map* en entrée du premier block d'un groupe seront additionnées aux *feature map* de sortie de ce groupe. Un exemple de ce réseau est visible sur la Figure 56.

La notion de groupe est variable selon le niveau de coupure P choisi. Ainsi, pour $P=1$, on considère un groupe uniquement. Le réseau est donc semblable à un ResNet standard (pas de liaison inter-groupe). Pour $P=2$, on considère 2 groupes. Ainsi, la connexion inter-block est unique et relie l'entrée du premier bloc avec la sortie du dernier bloc du réseau. De même, pour $P=3$, on considère 3 niveaux de groupe. De ce fait, une connexion inter-block est faite entre le premier et dernier block de même largeur de couche (16,32 ou 64 dans le cas présent).

L'architecture du modèle est généraliste et peut être associée à des modèles existants tels que Wide Residual Network par exemple. L'idée portée par l'auteur

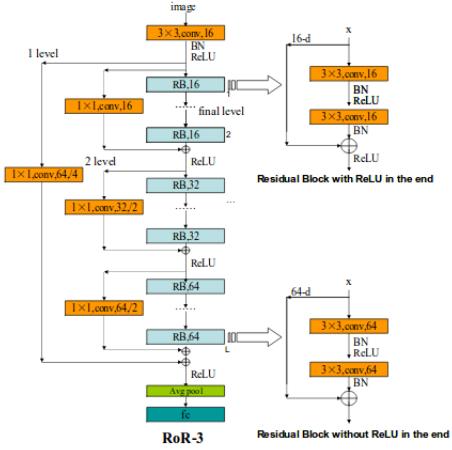


Figure 56: Exemple d'un réseau RoR de niveau 3

est de ne pas se limiter à la connexion entre un block et son successeur mais réaliser une connexion entre l'entrée et la sortie d'un *Super-Block* composé d'un ensemble de block de même nature. Cette approche propose ainsi une approche dépendante de l'architecture de groupe de réseau alors que l'approche Densely conenected et Sparsely Connected considère le block de manière unitaire et isolée.

6.6.7.6 Sparsely Connected Convolutional Networks (SparseNet)

Sparsely Connected Convolutional Networks[138](2018) repose sur les fondations du réseau DenseNet à la différence que les sorties des blocks ne sont pas transmises à l'ensemble des couches suivantes mais en fonction d'un offset exponentiel. En effet, pour un block B_n , seules les sorties des couches $B_{n-1}, B_{n-2}, B_{n-4}, B_{n-8}, \dots, B_{n-2^k}$ avec k , plus grand entier non négatif tel que $2^k < l$, sont considérées. Une comparaison entre SparseNet et DenseNet est visible sur la Figure 57.

L'approche par concaténation proposée par DenseNet a permis une amélioration significative du modèle ResNet en corrigeant le défaut de la méthode additive. Néanmoins, cette approche pose un problème du fait de l'explosion du nombre de connexions qui est de complexité $O(N^2)$. Cette augmentation de paramètres présente un risque d'overfitting et une explosion de la consommation mémoire du réseau lorsqu'il est profond. De plus, diverses expériences ont montré que l'utilisation de l'information transmise par les connexions inter-block est mal exploitée par le réseau avec une part importante de *feature map* transférées sans réel pouvoir explicatif. Le réseau DenseNet est donc trop dense. L'ajout d'un offset exponentiel d'ordre 2 permet ainsi de limiter ce défaut en limitant

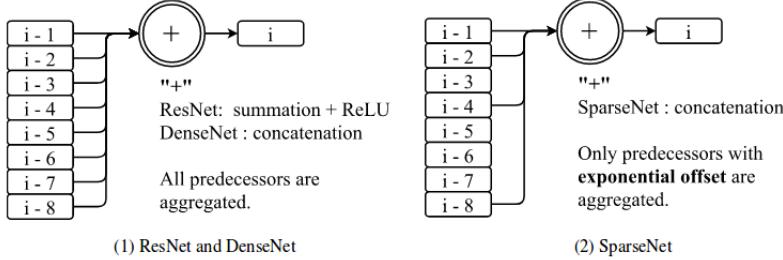


Figure 57: Différence entre un block issu de ResNet, DenseNet et SparseNet

la complexité à $O(\log(n))^{126}$ tout en conservant l’information utile portée par les différentes connexions inter-block. L’hypothèse est faite qu’il est préférable de privilégier l’information portée par les block adjacents que les block très éloignés.

6.6.7.7 Résumé des différentes approches appliquées aux ResNet

Ci-dessous, un tableau récapitulatif de l’ensemble des améliorations étudiées concernant les réseaux ResNet. Certaines de ces méthodes sont cumulatives. Approfondir leurs interactions est une piste de recherche intéressante et pertinente.

Méthodes	Spécificités
Wide Residual	Elargissement du flux d’un bloc Utilisation de couches spécifiques (DropOut...)
Aggregated Residual Transformations	Ajout de flux dans un block
Densely Connected	Propagation de la sortie sur l’ensemble des couches suivantes Concaténation de la sortie du block avec son entrée
Residual Networks of Residual Networks	Propagation de la sortie selon les groupes de block
Sparsely Connected	Propagation de la sortie selon un offset exponentiel Concaténation de la sortie du block avec son entrée
Stochastic Depth	Non-utilisation de block durant l’apprentissage (probabiliste)
Squeeze-and-Excitation	Pondération des <i>feature map</i>

6.6.8 FractalNet: Ultra-Deep Neural Networks without Residuals

Le modèle FractalNet[62](2016) est une approche qui s’émancipe de l’architecture des ResNet (ou dérivés) et se présente comme une alternative viable. Ce réseau démontre que, bien que très efficace, l’approche par résidu n’est pas la seule approche compétitive et performante. Néanmoins, FractalNet est un réseau récent encore méconnu. Il est donc difficile de juger pleinement de ses capacités. La

¹²⁶On peut grossièrement considérer que le nombre de connexions inter-block total est borné du fait de la faible croissance de la fonction logarithme

structure du réseau Fractalnet est comparable au comportement d'une fractale, i.e la structure du réseau suit un modèle similaire peu importe le sous-réseau observé. La structure fondamentale du ce réseau est explicitée par la Figure 58.

FractalNet exploite une alternative au DropOut afin de régulariser son réseau. Le processus, appelé **Drop-path** se divise en deux parties:

- **Local:** L'étape Local désactive des flux (ou entrées de couche *join*) aléatoirement selon une probabilité fixée. Il y a une garantie de l'existence d'au moins un flux pour chaque couche *join*.
- **Global:** L'étape Global sélectionne un unique path pour tout le réseau et désactive tous les autres flux. Cette approche permet de limiter les dépendances entre flux, de masquer l'origine du flux et ainsi, de bloquer le comportement du réseau qui viserait à considérer un flux comme un résidu ou un terme correctif.

La couche *join* peut sembler similaire à l'additivité employée dans un réseau ResNet. Cependant, il existe des différences critiques non négligeables:

- **Non différenciation de la source:** Contrairement au réseau ResNet, le réseau FractalNet ne différencie pas le flux associé à un résidu et le flux associé à la couche interne du block. Cette non-différenciation est permise grâce au transfert direct sans transformation préalable à la couche de convolution suivante.
- **Régularisation par Drop-path:** Ce procédé permet de forcer chaque flux à être *viable*. De ce fait, la régularisation favorise un apprentissage qui punit les flux qui évoluent vers un comportement de *résidu*.
- **Sous-réseau unitaire et performant:** Le Drop-path, du fait de son étape *Global* favorise les performances de sous-réseaux unitaires, i.e qui n'exploite pas les caractéristiques de la couche *join*. En effet, l'étape *Global* force les couches *join* à n'avoir qu'une seule entrée. Ainsi, ces sous-réseaux sont fonctionnels et produisent un signal qui ne devrait pas "recevoir" de résidu pour être efficace.

6.6.9 Classement des architectures standards

Le graphique présent sur la Figure 59 résume les caractéristiques des différentes architectures vues précédemment. Ainsi, en résumé, nous pouvons observer que SENet et NASNet font office d'état de l'art actuellement en terme de performances brutes. On peut observer la domination de NASNet sur ses concurrents pour chaque catégorie de modèle (léger/intermédiaire/lourd). Néanmoins, NASNet est critiqué pour son architecture qui présenterait des difficultés à être efficace sur des jeux de données autres que ImageNet ou CIFAR, contrairement à SENet ou des modèles standards reposant sur des architectures Inception ou

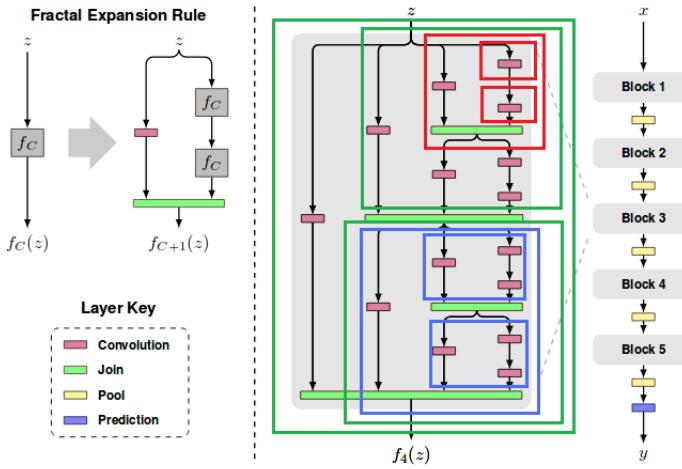


Figure 58: Construction d'un bloc du réseau FractalNet

ResNet. Les modèles VGG sont très lourds, lents et avec une efficacité en déclin bien que toujours très employés comme extracteur d'attribut dans le cadre d'architectures complexes. Tout comme AlexNet, ils tendent à devenir obsolètes. Pour une étude complète et détaillée des différents modèles de Deep Learning, veuillez vous référer à l'article [9]¹²⁷.

Remarque: NASNet est une architecture qui a été créée par un générateur automatique (développé par Google) selon l'algorithme *Neural Architecture Search* qui exploite l'apprentissage par renforcement. Il s'agit donc d'une architecture entièrement créée indépendamment de l'Homme. A la vue de ses performances, il est possible qu'à l'avenir, l'optimisation d'architecture neuronale se fasse grâce à des algorithmes automatiques.

6.6.10 Réseaux de neurones compressés

Les modèles étudiés précédemment sont pour la plupart gourmands et lourds. Ils posent un problème de mémoire et de performance sur des supports de faible capacité comme l'embarqué ou les appareils nomades tels que les smartphones. Afin de répondre à cette problématique, une optimisation du temps de calculs et du nombre de paramètres sont nécessaires tout en limitant la perte de la qualité prédictive du réseau. Ce type de réseaux dits *compressés* ont une importance élevée du fait de leurs capacités de commercialisation notamment à travers les applications des supports mobiles (téléphones, tablettes, domotiques...)

¹²⁷Étant donné la rapide évolution des modèles convolutifs, cette étude peut être obsolète lors de votre lecture.

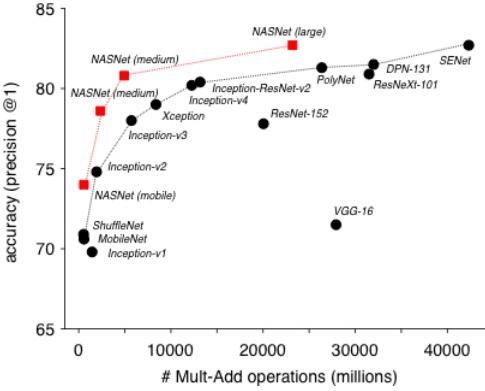


Figure 59: Analyse des réseaux de Deep Learning (2018)

6.6.10.1 SqueezeNet

Le réseau SqueezeNet[45](2016) est un réseau développé pour être supporté par les supports les plus *faibles*. Sa structure cherche donc à limiter le nombre de paramètres et son impact mémoire. Ainsi, par rapport à AlexNet (qui lui sert de référence), SqueezeNet est 50x plus léger et son poids peut être 510x plus faible avec application de méthode de compression de réseau, notamment Deep Compression (voir Section 5.12). Sans compression, ce réseau pèse 4.8MB et après compression, 0.47. Cette taille minime lui permet d'être implémenter sur de nombreuses structures de faible capacité bien que la précision du réseau soit significativement inférieure aux autres réseaux (bien plus volumineux). Il reste cependant satisfaisant pour des tâches de faible complexité ou sans nécessité de précision de haute performance.

La structure de ce réseau repose sur trois principes:

- **Utilisation en majorité de filtre 1*1 à la place de 3*3** afin de limiter le nombre de paramètres à apprendre et le nombre de calculs nécessaires.
- **Diminution du nombre de channels entrants dans une couche de convolution.** Cette diminution est réalisée grâce à une couche *squeeze*¹²⁸. Afin de ne pas trop perturber la précision du modèle en supprimant en excès les filtres 3*3, limiter le nombre de channels à analyser est une alternative efficace. En effet, le nombre de calculs à réaliser pour une couche de convolution 3*3 est: $nbr_{inputChannel} * nbr_{filtre} * (3 * 3)$.
- **Conservation de la dimension de l'entrée** afin de favoriser une bonne précision du modèle. En effet, les principes précédents favorisent une réduction du poids du réseau au détriment de sa performance. Il est donc nécessaire de lutter contre une trop grande détérioration de la précision. Pour cela, la dimension des *feature map* est faiblement modifié

¹²⁸Nous verrons par la suite la configuration de cette couche

jusqu'en fin de réseau contrairement à la plupart des réseaux plus volumineux qui réalisent de nombreuses transformations internes, notamment via l'application d'un stride supérieur à 1 ou des approches par *Pooling*. Cette idée relève d'une intuition des créateurs du réseau et n'a pas de preuve véritablement démontrée.

Afin de respecter les deux premiers principes, un nouveau type de block est créé: le *Fire Module*. Ce module est séparé en deux parties: la partie *Squeeze* et la partie *expand*. La partie *Squeeze* est uniquement composée de filtres 1×1 (Principe 1), la partie *Expand* analyse les *feature map* issues de la partie *Squeeze* et est composée de filtres 1×1 et 3×3 . Le nombre de filtre 1×1 dans la partie *Squeeze* est inférieur au nombre de filtres (1×1 et 3×3) dans la couche *Expand*, ce qui permet de maîtriser la profondeur des entrées et de conserver un nombre de dimension restreint sans négliger le nombre de filtre (Principe 2). L'absence de *Pooling* et de stride permet de garder les mêmes dimensions de *feature map* (hauteur et largeur). Le block possède de nombreux hyperparamètres afin de définir la répartition des filtres et l'évolution de la configuration du filtre au fil du réseau. Pour plus de détails, veuillez consulter l'article[45]. Une illustration du block est visible sur la Figure 60.

L'architecture du réseau repose sur une succession de Fire Module régulièrement régulée par des couches de pooling inter-block. Il est intéressant de noter que le nombre de *feature map* de sortie augmente régulièrement jusqu'à la fin du réseau, ce qui traduit une augmentation du nombre de filtres. De plus, ce réseau possède une architecture standard ou résiduelle. Dans le cas d'une architecture résiduelle, deux approches sont possibles. La première consiste à réaliser une connexion interblock à chaque fois que la dimension de l'entrée est identique à la dimension de sortie (*simple bypass*). La seconde crée une connexion interblock pour chaque bloc. En cas de différence de dimensions, une couche de convolution 1×1 mettra à l'échelle la profondeur du résidu du block considéré (*complex bypass*). L'ajout de la structure résiduelle sert à palier la perte d'informations associée à la couche *Squeeze* des blocs.

6.6.10.2 MobileNet

L'une des architectures les plus populaires est MobileNet[36](2017). Ce modèle exploite les convolutions Depthwise/Pointwise afin de limiter son impact mémoire et accélérer sa vitesse. Les filtres des Depthwise sont de taille 3×3 uniquement, ce qui est un standard avec le filtre 5×5 . Cette dimension de filtre reste néanmoins l'un voire le meilleur rapport temps/précision/mémoire pour un modèle qui se veut léger et rapide.

L'architecture se veut modulable et adaptatif. Pour cela, deux hyperparamètres ont été instaurés: le **multiplicateur de largeur** et le **multiplicateur de résolution**. Ces paramètres de modifier la structure du réseau tout en conservant son intégrité et sa structure, uniforme. Il est donc possible de paramétriser

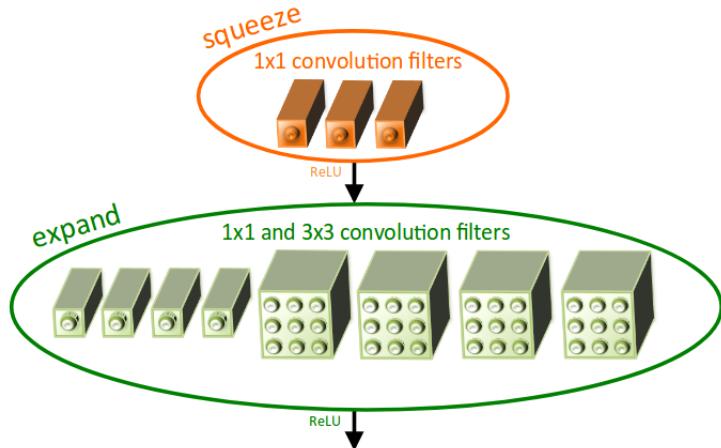


Figure 60: Architecture du Fire Module

aisément sa profondeur afin de l'adapter aux différentes capacités des supports d'application.

- **Multiplicateur de largeur:** Ce critère (noté α) joue sur la profondeur des entrées et des sorties des couches de convolutions en adaptant leurs dimensions selon un coefficient. Ainsi, en reprenant les résultats obtenus dans la Section 6.3.8, nous obtenons:

$$D_{\text{depthwise/poinwise}, \alpha} = (X*X)*\alpha M*(K*K)*1 + (X*X)*\alpha M*(1*1)*\alpha N$$

En pratique, cet hyperparamètre est implémenté de manière à limiter le nombre de *feature map* de sortie d'une couche de convolution. En effet, cette sortie sera l'entrée de la couche suivante. Il est donc évident que si la dimension d'une sortie est minorée, la dimension de l'entrée de la couche suivante le sera aussi.

- **Multiplicateur de résolution:** Ce critère (noté ρ) modifie la dimension de l'image d'entrée, réduisant ainsi la représentation interne dans chaque couche du réseau. Nous obtenons ainsi:

$$D_{\text{depthwise/pointwise}, \rho} = (\rho X * \rho X) * M * (K * K) * 1 + (\rho X * \rho X) * M * (1 * 1) * N$$

Ce critère est peu explicité dans l'article publié et son implémentation douteuse. Il est possible qu'il soit théorique et au final, non représenté concrètement.

Avec l'application des deux coefficients, l'équation du coût de ce réseau s'exprime donc sous la forme:

$$D_{\text{depthwise/poinwise}, \alpha, \rho} = (\rho X * \rho X) * \alpha M * (K * K) * 1 + (\rho X * \rho X) * \alpha M * (1 * 1) * \alpha N$$

6.6.10.3 Modèles expérimentaux récents

Afin d'approfondir l'état de l'art, voici une liste de modèles récents prometteurs qu'il peut être intéressant d'étudier:

- ShuffleNet[134] (Dec 2017): Architecture pour support de faible puissance (smartphone par exemple). Ses résultats semblent meilleurs que MobileNet pour un coût machine plus faible.
- EffNet[22] (En cours d'écriture - Mars 2018): Architecture pour support de faible puissance (smartphone par exemple). Plus récent que MobileNet et ShuffleNet, il cherche à corriger leurs défauts. En cours d'écriture, il n'est pas possible de juger les résultats de cette architecture actuellement mais les résultats intermédiaires sont prometteurs.

6.6.11 Les jeux de données de référence

Le Deep Learning nécessitant de nombreuses données, la création de jeux de données de qualité a été au coeur de la Recherche. Bien que ce soit encore une thématique en cours d'étude, de nombreux jeux de données ont été créés pour effectuer des *benchmarks* compétitifs, répondre à une problématique ou tout simplement, faciliter le développement des algorithmes de Deep Learning en soulageant la difficulté de création d'un jeu de données.

L'analyse d'image étant le secteur historique principal du Deep Learning, de nombreux jeux de données ont été créés. Il est intéressant d'en connaître les principaux afin de pouvoir les utiliser dans des projets. Dans cette section, nous allons considérer les jeux de données concernant l'analyse d'image uniquement.

- **MNIST: handwritten digits**[64]: Ce jeu de données est historique et a accompagné l'ascension des réseaux convolutifs. Il est constitué de chiffres entre 0 et 9 écrit à la main représentée en noir et blanc¹²⁹. Souvent utilisé pour évaluer la pertinence d'un modèle, il tend à devenir obsolète car trop "facile" à résoudre. Les meilleurs modèles actuels dépassent 99% sur ce jeu de données.
- **CIFAR10 / CIFAR100**[59][60]: Il s'agit d'un jeu de données généraliste composé d'image 32*32 réparties en 10/100 classes. Il n'est pas très utilisé mais permet d'évaluer les performances d'un modèle de manière préventive avant d'exploiter des jeux de données plus populaires et plus "exigeants".
- **Pascal VOC**[20]: Ce jeu de données est relativement ancien et n'est plus mis à jour. Il est rarement utilisé dans le cadre d'apprentissage pour de la classification d'image mais très populaire pour la détection d'objet.
- **ImageNet**[15]: C'est l'un des jeu de données les plus riches actuellement. Avec plus de 10 millions d'image pour 1000 catégories, il offre

¹²⁹Des variantes ont été créées en niveau de gris

une richesse d'apprentissage de premier ordre. Une partie de ce jeu de données a été convertie en *bounding boxes*. Avec de nombreuses API disponibles, ce jeu de données a l'ambition de devenir une voire la référence pour l'apprentissage générique. Il est associé à une compétition annuelle (ILSVRC) qui est considérée comme l'une voire la compétition la plus prestigieuse en analyse d'image actuellement.

- **MS COCO[70]:** Il s'agit d'un jeu de données générique (2.5 millions d'image pour 91 classes) associé à une compétition d'analyse d'image (COCO). Étant légèrement dans l'ombre d'ImageNet en terme de popularité, ce jeu de données s'illustre avec ses images annotées (*captioning*).

Ces jeux de données sont les références *généralistes*. Il existe de nombreux autres jeux de données généralistes tout comme des spécialisés, notamment d'images de visage (pour la reconnaissance faciale), des images aériennes/satellites ou encore des images issues de la *Physique* (astrophysique/cosmologie, physique fondamentale, mécanique...) et de la Médecine/Biologie (imagerie médicale) par exemple.

7 Encoder-Decoder

7.1 Généralités

Les *Encoder-Decoder* sont une famille de réseaux dont l'appellation provient du *traitement du signal*. Un réseau *Encoder-Decoder* se compose de deux parties: la partie Encoder et la partie Decoder. Ces deux parties sont formées par deux réseaux indépendants.

L'objectif de l'Encoder est de représenter la donnée d'entrée (définie dans un espace \mathcal{X}) dans un espace \mathcal{Z} souvent de dimension inférieure. La donnée obtenue s'appelle *vecteur de contexte* (context vector ou code). Dans le cas où la dimension de l'espace de sortie est inférieure à celle de l'espace d'entrée, on dit que la représentation de l'entrée est *compressée*.

Le Decoder est le complémentaire de l'Encoder. Il exploite la projection de l'Encoder afin de créer une nouvelle projection dans un autre espace souhaité. Il n'y a aucune obligation à conserver le même espace que celle de la donnée d'entrée ou du vecteur de contexte. La non-obligation de conservation de l'espace permet de rendre l'application de ce type d'architecture très diversifiée, notamment dans le cadre du nettoyage/réduction des données (pré-processing), la traduction multi-langue et la génération de données artificielles.

Ainsi, nous pouvons résumer l'action d'un Encoder-Decoder par:

$$\begin{aligned}\phi : \mathcal{R}^x_x &\xrightarrow{\quad} \mathcal{R}^z \\ \varphi : \mathcal{R}^z_z &\xrightarrow{\quad} \mathcal{R}^y \\ ED : \mathcal{R}^x_x &\xrightarrow{\quad} \varphi(\phi(x))\end{aligned}$$

Avec ϕ , la fonction de l'Encoder et φ , celle du Decoder.

7.2 Autoencoder

Un *Autoencoder*[6] est un cas particulier d'architecture Encoder-Decoder. Il est caractérisé par un espace de projection du Decoder identique à celui de la donnée d'entrée et l'objectif est de "retrouver" la donnée d'entrée. Ainsi, l'action d'un Autoencoder peut être définie par:

$$\begin{aligned}\phi : \mathcal{R}^x_x &\xrightarrow{\quad} \mathcal{R}^z \\ \varphi : \mathcal{R}^z_z &\xrightarrow{\quad} \mathcal{R}^x \\ ED : \mathcal{R}^x_x &\xrightarrow{\quad} \varphi(\phi(x)) \rightarrow x\end{aligned}$$

L'Autoencoder est composé de structures comparables à un modèle Feed-Forward (Full-Connected) ou convolutif. Son apprentissage est donc possible avec une approche classique. Pour cela, la rétropropagation du gradient et les fonctions de coût standards sont employées. Ainsi, pour une donnée d'entrée binaire x de dimension n , nous utilisons la *Binary Cross-Entropy*:

$$\mathcal{L}(\theta) = - \sum_{i=1}^n [x_i \log(\hat{x}_i) + (1 - x_i) \log(1 - \hat{x}_i)]$$

Dans le cas d'une entrée x à valeur réelle dans \mathcal{R}^n , nous préfèrerons la *Mean squared error* (MSE):

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n (\hat{x}_k - x_k)^2$$

Du fait que l'objectif est d'obtenir à nouveau la donnée d'entrée, la labellisation de la donnée d'apprentissage est inutile, ce qui fait d'un Autoencoder, une méthode d'apprentissage *non supervisé*.

Un Autoencoder peut avoir une ou plusieurs couche cachées. Dans le cas d'accumulation de couches cachées, nous parlerons de *Deep Autoencoder*. Les Deep Autoencoder favorisent l'utilisation de fonction d'activation non linéaire, l'exploitation de couche de convolution au détriment des couches Full-Connected et ont un grand pouvoir explicatif. Le risque d'overfitting est donc plus important et demande une attention particulière.

L'approche standard de l'Autoencoder est essentiellement utilisée dans le cadre d'une réduction de dimension¹³⁰. Elle est donc utile pour réaliser une compression de donnée ou du pré-processing de données. Ils sont aussi utilisés dans le cadre d'une recherche de complétion de données en présence de données incomplètes ou partiellement corrompues.

7.2.1 Autoencoder Undercomplete

L'*Autoencoder Undercomplete* est souvent associé à l'architecture de base d'un Autoencoder. L'idée derrière cette architecture est de réaliser une projection de la donnée d'entrée sur un espace de dimension plus **faible**. Ainsi, l'Encoder est défini par:

$$\phi : \mathcal{R}^X \xrightarrow{\quad} \mathcal{R}^Z \quad \text{avec } Z \ll X$$

La projection sur un espace de plus petite dimension force l'Autoencoder à extraire l'information importante des données d'entrée. Si la dimension de l'espace de projection est identique à celle de l'espace de la donnée d'entrée, le modèle aura tendance à apprendre la fonction *identité*, ce qui empêchera l'extraction d'information des données. Cette particularité rendra donc inutile l'Encoder et

¹³⁰Ce modèle est capable d'approximer les résultats d'une ACP linéaire et non linéaire par exemple

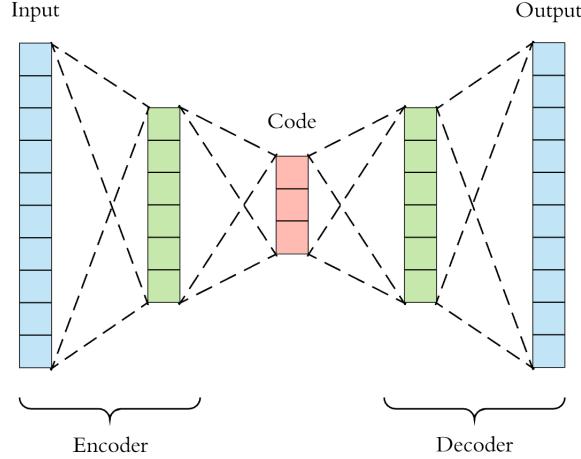


Figure 61: Architecture d'un Autoencoder

de ce fait, le Decoder car le vecteur de contexte tendra à être équivalent à la donnée d'entrée. Une représentation d'un Autoencoder est visible sur la Figure 61.

7.2.2 Autoencoder Regularisé ou Overcomplete

Un *Autoencoder Overcomplete* s'émancipe de la dimension du vecteur de contexte et propose de réaliser une projection sur un espace de dimension identique voire supérieure. Afin de lutter contre le risque de création d'une représentation *identité*, cette architecture *régularise* la fonction de coût pour pousser le modèle à extraire des propriétés des données au lieu d'une simple représentation de l'identité de la donnée d'entrée. Cette approche tend à être plus robuste et plus efficace que la méthode Undercomplete.

Il existe différentes régularisations possibles afin de réaliser ce type d'Autoencoder. Ces méthodes peuvent être associées et sont encore un sujet de recherche actuel.

7.2.2.1 Denoising Autoencoder

Au lieu d'analyser la donnée d'entrée intègre, un *Denoising Autoencoder*[114] exploite la donnée d'entrée *bruitée*. Ce réseau doit donc apprendre à débruiter au lieu d'uniquement *copier* l'entrée.

Il existe différentes méthodes pour bruiter les données (application d'un bruit gaussien, filtre quelconque, etc...). En pratique, l'approche la plus populaire consiste à désactiver aléatoirement des neurones de la couche d'entrée en forçant leurs sorties à 0. Cette méthode est comparable à l'application d'un DropOut sur la couche d'entrée du réseau. Une illustration de ce réseau est visible sur la

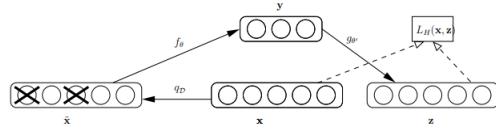


Figure 62: Architecture d'un Denoising Autoencoder

Figure 62.

Important Lors du calcul de la fonction de coût, la comparaison est réalisée avec la donnée intègre et non la donnée bruitée ! Comparer à la donnée bruitée reviendrait à réaliser un Autoencoder qui apprendrait à représenter les données bruitées et non intègres.

Un *Denoising Autoencoder* est donc défini par:

$$\begin{aligned} \text{Noise} : \mathcal{R}^x &\xrightarrow{\quad} \mathcal{R}^x \\ \phi_{\text{Noise}} : \mathcal{R}^x &\xrightarrow{\quad} \mathcal{R}^z \\ \varphi : \mathcal{R}^z &\xrightarrow{\quad} \mathcal{R}^x \\ ED : \mathcal{R}^x &\xrightarrow{\quad} \varphi(\phi(\text{Noise}(x))) \rightarrow x \end{aligned}$$

Cette architecture peut être utilisée pour créer un *débruiteur*. En effet, dans la méthode standard, la donnée intègre est corrompue par une architecture particulière du réseau ou l'ajout d'un bruit artificiel en amont. Nous créons donc, en interne, un jeu de données composé d'entité $[\text{donnee}_{\text{intègre}}, \text{donnee}_{\text{bruitee}}]$. Le réseau devient donc robuste à corriger les effets du bruit introduit dans ces données. Neanmoins, la correction du bruit est, dans cette configuration, peu utile d'un point de vue métier. Une autre approche serait de réaliser un apprentissage *supervisé* où les données d'entraînement sont issues d'un jeu de données corrompues par un bruit spécifique. L'entraînement sur ce jeu de données permettra ainsi d'obtenir un modèle apte à débruter un message utile soumis au bruit présent dans les données d'entraînement. Ce modèle obtenu aura donc une plus-value métier car capable de réaliser un débruitage *utile*. On peut voir une application directe dans le cadre des télécoms.

7.2.2.2 Sparse Autoencoder

Le *Sparse Autoencoder*[79] cherche à rendre son architecture épars¹³¹, i.e éviter que l'intégralité des neurones soit forcée d'être stimulée à chaque donnée d'entrée. Cette approche tend à rendre les neurones plus spécialisés en limitant leurs degrés de liberté qui pourraient nuire à la qualité du neurone si les données sont

¹³¹On parle de *sparsity constraint*

significativement différentes. L'objectif est donc de rendre un neurone sensible à une stimulation spécifique et non l'ensemble des stimulation qu'il reçoit. Ainsi, un neurone doit être inactif la majorité du temps. Cette caractéristique est très importante en présence de données significativement distinctes tels que des jeux de données multi-classes.

Afin de réguler l'activité des neurones, deux approches sont essentiellement employées: La régularisation par la Divergence de Kullback-Leibler ou la k-Sparse constraint.

- **KL-divergence:** La régularisation par la KL-divergence modifie la fonction de coût en rajoutant un facteur correspondant à la divergence de Kullback-Leibler.

Supposons un neurone i appartenant à une couche cachée et a_i , son activation. On notera $a_i(x_j)$ l'activation du neurone i selon la donnée d'entrée x_j . On peut donc définir la valeur moyenne d'activation du neurone i sur un jeu de données de m éléments par: $\hat{\rho}_j = \frac{1}{m} \sum_{j=1}^m [a_i(x_j)]$.

Nous voulons que $\hat{\rho}_j = \rho$ où ρ est le paramètre déterminant le degrés de dispersion du réseau. ρ doit être faible (supposons 0.05). Cela signifie que nous souhaitons que la moyenne d'activation tende vers 0.05. Ainsi, cette condition impose que le neurone soit constamment éteint (activation proche de 0) en dehors de phase d'activation ponctuelle où la valeur tendra vers 1. Cette condition suppose une activation dans $[0, 1]$ telle que l'activation via une sigmoïde¹³² par exemple.

Le statut d'un neurone peut être simplifié par une approximation binaire: allumé ou éteint. Ainsi, on peut considérer l'état d'un neurone comme un problème explicable par la loi de Bernouilli. La divergence de Kullback-Leibler permet de calculer la différence entre deux distributions. Ainsi, il est possible de déterminer la différence entre la distribution des activations du neurone réel (variable aléatoire de Bernouilli de moyenne $\hat{\rho}_j$) et la distribution idéale souhaitée (variable aléatoire de Bernouilli de moyenne ρ_j). Le calcul de la différence est défini par:

$$KL(\rho || \hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j}$$

$KL(\rho || \hat{\rho}_j)$ est égale à 0 en cas d'égalité des distributions et augmente lorsque les distributions se diffèrent. Son action a donc le même comportement qu'une régularisation classique telle que \mathcal{L}_2 par exemple. La

¹³²Il est possible de considérer d'autre fonction telle que tanh en modifiant les conditions d'excitation/repos du neurone. Il est cependant nécessaire de considérer des fonctions qui présentent des bornes asymptotiques vers les infinis.

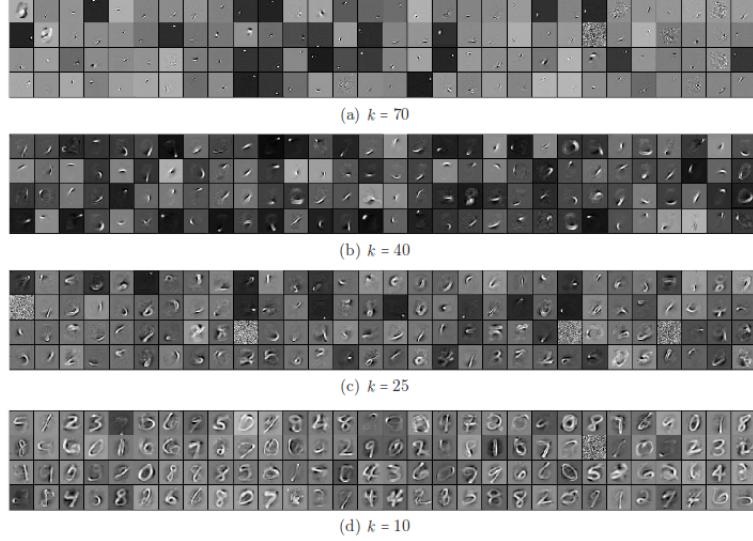


Figure 63: Filtre d'un k-Sparse Autoencoder selon la valeur de k sur le jeu de données MNIST

fonction de coût est donc définie par:

$$\mathcal{L}(\theta)_{sparse} = \mathcal{L}(\theta) + \beta \sum_{j=1}^{s_2} \text{KL}(\rho || \hat{\rho}_j)$$

On suppose qu'il y a s_2 neurones dans les couches cachées. β est un coefficient qui contrôle l'importance donnée à la condition de dispersion du réseau.

- **k-Sparse constraint**[76]: Un k-Sparse Autoencoder réalise une action comparable à un DropOut/DropConnect. En effet, l'approche proposée par ce modèle consiste à ne considérer que les k plus grandes valeurs d'activation d'une couche cachée (généralisable à l'ensemble des couches cachées indépendamment les unes des autres) et à mettre à 0, toutes les autres. La rétropropagation du gradient n'est effective que sur les neurones activés. Lors de l'utilisation du modèle, il faudra exploiter les αk valeurs les plus importantes avec $\alpha \geq 1$. Il a été montré qu'un α supérieur à 1 donne des résultats sensiblement meilleurs¹³³. L'impact de la valeur de k est visible sur la Figure 63.

¹³³Sans pour autant exagérer sur sa valeur...

7.2.2.3 Contractive Autoencoders

Un Contractive Autoencoders [92] repose sur l'idée que le modèle doit être robuste aux changements et éviter des variations importantes dans la représentation du vecteur de contexte (autour des données d'apprentissage). Pour cela, une régularisation par la norme Frobenius de la matrice jacobienne de l'Encoder est utilisée.

La nouvelle fonction de coût est exprimée par:

$$L = \|X - \hat{X}\|_2^2 + \lambda \|J_h(X)\|_F^2$$

$$\|J_h(X)\|_F^2 = \sum_{ij} \left(\frac{\partial h_j(X)}{\partial X_i} \right)^2$$

Détaillons le calcul de la régularisation. Supposons une fonction d'activation sigmoïde. Nous avons donc:

$$Z_j = W_i X_i \quad (2)$$

$$h_j = \phi(Z_j) \quad (3)$$

$$\frac{\partial h_j}{\partial X_i} = \frac{\partial \phi(Z_j)}{\partial X_i} \quad (4)$$

$$= \frac{\partial \phi(W_i X_i)}{\partial W_i X_i} \frac{\partial W_i X_i}{\partial X_i} \quad (5)$$

$$= [\phi(W_i X_i)(1 - \phi(W_i X_i))] W_i \quad (6)$$

$$= [h_j(1 - h_j)] W_i \quad (7)$$

On obtient:

$$\|J_h(X)\|_F^2 = \sum_{ij} \left(\frac{\partial h_j}{\partial X_i} \right)^2 \quad (8)$$

$$= \sum_i \sum_j [h_j(1 - h_j)]^2 (W_{ji}^T)^2 \quad (9)$$

$$= \sum_j [h_j(1 - h_j)]^2 \sum_i (W_{ji}^T)^2 \quad (10)$$

Ce calcul est très semblable au calcul utilisé durant la retropropagation du gradient. En supposant une couche caché de 20 neurones, alors le problème est déterminé par 20 fonctions définies par un vecteur de gradient chacune d'où

l'obtention d'une matrice jacobienne. Il est important de noter que cette régulation exploite une norme de la jacobienne. Cette particularité est intéressante car elle permet d'éviter de devoir définir la matrice diagonale de $[h(1 - h)]^2$. En effet, Ce facteur dépend d'une variable autre que celle de W. Il faut donc associer à chaque dimension de W, l'entité (unique) de $[h(1 - h)]^2$ correspondant, ce qui impose l'exploitation d'une matrice diagonale qui peut être délicate à obtenir.

La régularisation de $\|J_h(X)\|_F^2$ est uniquement destructrice. Elle punit toute variation associée à l'activation des neurones de l'Encoder. Néanmoins, cette régularisation est compensée par $\|X - \hat{X}\|_2^2$ qui peut avoir un bon résultat. $\|X - \hat{X}\|_2^2$ possède un résultat amélioré lorsque la variation demandée au réseau lors d'une mise à jour des poids est bénéfique à l'apprentissage. Au contraire, lors d'une variation sans impact, $\|X - \hat{X}\|_2^2$ stagne. Dans ce dernier cas, l'impact de la régularisation est trop fort et l'optimisation du réseau ne considérera pas cette possibilité d'évolution. Ainsi, seules les modifications de faible ampleur¹³⁴ autour des données d'apprentissage seront considérées, les autres variations étant *contractées*. Il serait intéressant d'approfondir les problématiques de convergence de cette approche qui présente une faible capacité d'exploration¹³⁵.

Le facteur λ est important. En effet, ce paramètre régule l'importance donnée à la régulation. Plus λ est élevé, plus les variations seront pénalisées. Un facteur trop important bloquera donc le réseau car aucune variation ne sera jugée suffisamment bénéfique pour compenser la pénalisation de la variation.

7.3 Variational Autoencoders

Important: Comprendre ce modèle d'un point de vue théorique nécessite un solide bagage en probabilité (inférence bayesienne). Nous n'approfondirons donc pas l'origine et les démonstrations théoriques de ce modèle.

Le *Variational Autoencoders*(VAE)[16][55] est l'architecture d'Autoencoder la plus récente. La spécificité de ce modèle est qu'il apprend un modèle de **variable latente**. Un VAE est donc un **modèle génératif à conditions**. Contrairement au GAN¹³⁶ qui génère des entités aléatoirement, ce type de modèle est capable de générer des entités sous conditions, permettant ainsi de cibler la nature des résultats que nous souhaitons obtenir.

Un Encoder standard réalise une projection d'un vecteur sur \mathcal{R}^{dim} . Chaque image d'entraînement est donc clairement définie par un vecteur unique sur \mathcal{R}^{dim} . Cet aspect est très problématique lorsque nous désirons un modèle **génératif**

¹³⁴Une variation massive qui produit un gain de performance important peut cependant être accepté. Tout dépend du degrés de variation et du gain de performance associé

¹³⁵Le compromis Exploration/Exploitation est récurrent en apprentissage automatique, notamment en apprentissage par Renforcement

¹³⁶Nous étudierons ce type de modèle par la suite

car le Decoder ne peut s'émanciper des données d'apprentissage. En effet, l'espace de projection aura tendance à être très éparse et à présenter différents clusters éloignés les uns des autres et formés par des entités d'apprentissage similaires. Cette discontinuité et les difficultés d'interpolation associées font que le Decoder n'est pas capable de considérer des vecteurs sur \mathcal{R}^{dim} non référencés¹³⁷. Il n'est donc pas capable d'analyser des vecteurs de contexte inconnus, ce qui est problématique dans le cadre de la génération de données.

Le *Variational Autoencoders* propose une solution à ce problème en introduisant la notion de *variable latente* dans son vecteur de contexte. Au lieu de prédire des valeurs discrètes, l'Encoder va déterminer des paramètres d'une distribution de probabilité, ce qui permet une généralisation efficace. Ainsi, l'Encoder inférera la moyenne et l'écart-type d'une distribution normale pour chacune des dimensions du vecteur de contexte. L'architecture du réseau est visible sur la Figure 64. L'Encoder prédit un vecteur moyenne et un vecteur écart-type où chaque dimension de ces deux vecteurs correspondent deux-à-deux aux paramètres d'une distribution normale associée à la dimension concernée du vecteur de contexte. Une explication graphique est montrée par la Figure 65. Le Decoder recevra donc un vecteur inféré selon les distributions du vecteur de contexte.

Cette configuration présente une problématique majeure. Ne pas régulariser les paramètres favorisera la détermination de moyennes de distribution très éparses et diversifiées (selon les classes de données) avec un écart-type qui aura tendance à être faible. Cet aspect est préjudiciable car nous souhaitons que les données soient les plus proches possibles afin de faciliter les interpolations nécessaires à la création de nouvelles données. Afin de lutter contre ce phénomène, la fonction de coût est régularisée par la divergence de KullBack-Leibler qui permet de déterminer les divergences entre deux distributions. Ainsi, dans le cas du VAE, le facteur de régularisation sera définie par la somme des KL-divergence par rapport à la distribution normale centrée en 0 d'écart-type unitaire. L'objectif est donc de minimiser les écarts entre les distributions en les contrignant à être le plus proche possible d'une même distribution. L'union de Mean squared error et de la KL-divergence permet donc de conserver l'association des classes par cluster tout en favorisant une répartition dense.

Pour réaliser une génération de données, il n'y aura qu'à proposer un vecteur latent issu d'une distribution gaussienne unitaire au Decoder. L'extrapolation est possible du fait de la proximité des clusters, il est donc possible d'obtenir une infinité de génération en variant les valeurs obtenues par les différentes distributions. Un exemple de génération est visible sur la Figure 66.

Ce type d'architecture est un sujet de recherche très étudié, notamment en ex-

¹³⁷Le Decoder produira un résultat souvent mauvais car il ne "comprendra" pas le vecteur de contexte proposé

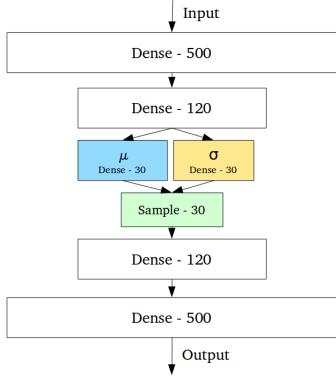


Figure 64: Architecture d'un Variationnal Autoencoder

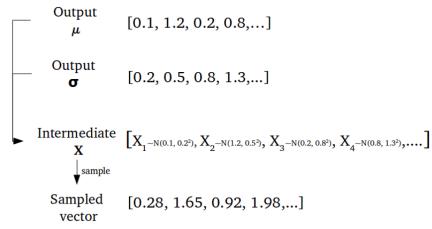


Figure 65: Détermination du vecteur de contexte par un Variationnal Autoencoder

ploitant des architectures de réseaux neuronaux déjà existants pour améliorer les capacités de l'Encoder/Decoder (tels que les LSTM). Les capacités de génération de données sont très convoitées par les domaines créatifs tels que les Arts ou la génération de données artificielles afin de répondre à la problématique du déficit de données d'apprentissage. On peut ainsi noter *PixelVAE*[27] dans le cadre de la génération d'image; *MusicVAE*[1] pour la génération de musique et [97][39] pour la génération de texte.

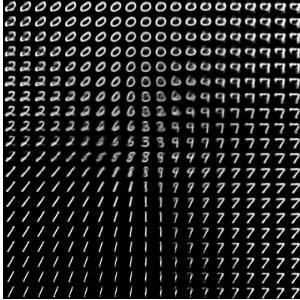


Figure 66: Generation d'image par un Variationnal Autoencoder entrainé sur MNIST

8 Réseaux antagonistes génératifs

8.1 Généralités

Un Réseau antagoniste génératif[26] (Generative Adversarial Networks (GAN)) est un réseau dont l'architecture s'inspire d'un problème issu de la *Théorie des Jeux*, le *Minimax*¹³⁸. Ses performances sont remarquables dans le cadre de la génération de données, notamment les images. Ce réseau est composé de deux sous-réseaux: un *Générateur* et un *Discriminateur*.

L'objectif du *Générateur* est de produire une donnée artificielle alors que le *Discriminateur* doit dissocier les images réelles des images artificielles. Le réseau va ainsi apprendre en cherchant à améliorer ses deux sous-réseaux: un *Discriminateur* plus efficace à détecter les *faux* et un *Générateur* plus efficace pour produire des *faux* "invisibles".

Le *Générateur* prend en entrée, un vecteur *bruitée* $c(\theta) \in R^{d139}$ issu d'une distribution normale ou uniforme entre -1 et 1 avant de produire une donnée artificielle de même dimension que les données réelles. Le *Discriminateur* est un réseau de classification binaire standard et reçoit en entrée, une donnée réelle et une donnée artificielle qu'il doit discriminer. Une illustration est visible sur la Figure 67.

Lorsque l'apprentissage est effectué, les données générées non discriminées par le *Discriminateur* constituent l'ensemble exploitable par l'utilisateur.

8.1.0.1 Fonction de perte

Supposons:

¹³⁸Pour plus d'informations sur ce jeu, consultez http://people.maths.ox.ac.uk/griffit4/Math_Alive/3/game_theory3.pdf

¹³⁹Par convention, d est égal à 100

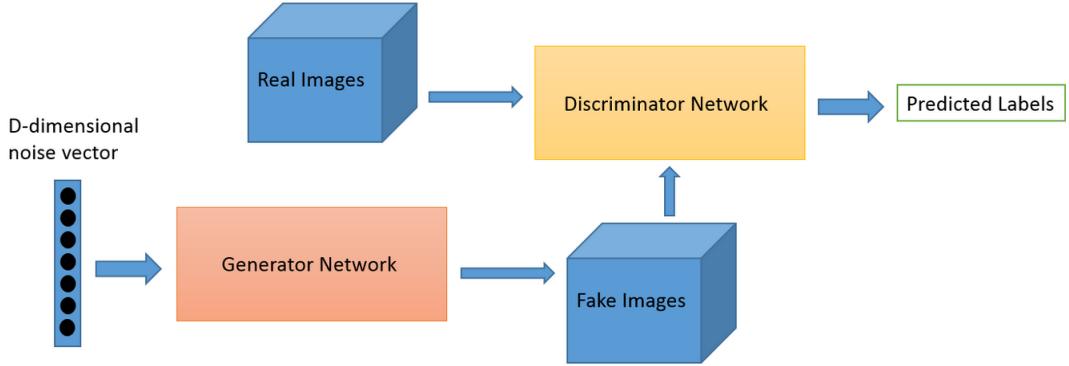


Figure 67: Architecture d'un réseau antagoniste génératif

- p_y , distribution du *bruiteur* associée à une entrée y
- p_r , distribution associée aux données réelles x_{real}
- D , fonction représentant le Discriminateur
- G , fonction représentant le Générateur

Dans un premier temps, nous voulons que le Discriminateur donne une forte probabilité d'être valide à une donnée réelle. Nous voulons donc que $D(x_{real}) \rightarrow 1$. Ainsi, nous pouvons considérer la fonction:

$$E_{x \sim p_r(x_{real})}[\log(D(x))] \\ E_{x \sim p_r(x_{real})}[\log(D(x))] \rightarrow 0 \text{ si } D(x_{real}) \rightarrow 1$$

De même, nous voulons que le Discriminateur prédise une faible probabilité d'être valide pour une donnée artificielle soit $D(G(y)) \rightarrow 0$. Nous obtenons alors la fonction:

$$E_{y \sim p_y(y)}[\log(1 - D(G(y)))]$$

Nous obtenons notre fonction de perte définie par:

$$L(D, G) = E_{x \sim p_r(x_{real})}[\log D(x)] + E_{y \sim p_y(y)}[\log(1 - D(G(y)))]$$

Une particularité importante est à remarquer. La fonction L est comparable au **négatif** de la fonction *Cross-Entropy*. Ainsi, pour l'apprentissage du Discriminateur, il est nécessaire de réaliser une optimisation par *gradient ascent* et non une descente de gradient standard car la "minimisation" de l'erreur du Discriminateur revient à maximiser cette fonction et non la minimiser comme pratiqué classiquement en Deep Learning. En effet, $L(D, G) \leq 0$.

Au contraire, pour l'apprentissage du Générateur, nous souhaitons une valeur élevée pour $D(G(y))$. Nous pouvons réutiliser la fonction $E_{y \sim p_y(y)}[\log(1 - D(G(y)))]$. L'apprentissage de ce réseau s'oppose ainsi à celui du Discriminateur et se basera sur la minimisation de la fonction de perte.

Le Générateur et le Discriminateur cherchent à optimiser deux fonctions de pertes **opposées**. Nous pouvons ainsi définir un jeu *minimax* basé sur la probabilité que la donnée artificielle soit considérée comme vrai.

En unifiant les différentes conditions, nous obtenons:

$$\min_G \max_D L(D, G) = E_{x \sim p_r(x_{real})}[\log D(x)] + E_{y \sim p_y(y)}[\log(1 - D(G(y)))]$$

Plus spécifiquement, l'apprentissage se fait en alternant l'optimisation des sous-réseaux indépendamment l'un de l'autre selon les fonctions suivantes:

- Générateur: descente de gradient

$$\min_G E_{y \sim p_y(y)}[\log(1 - D(G(y)))]$$

- Discriminateur: gradient ascent

$$\max_D E_{x \sim p_r(x_{real})}[\log D(x)] + E_{y \sim p_y(y)}[\log(1 - D(G(y)))]$$

L'alternance de l'apprentissage peut être de 1(D):1(G) ou de k:1. De nombreux chercheurs ont observé expérimentalement que l'apprentissage était plus stable avec un apprentissage plus soutenu du Discriminateur. La stabilité des GAN est une des problématiques majeures de ce type d'architecture car il est difficile de faire apprendre mutuellement deux réseaux. Si l'un des deux modèles dominent l'autre, la performance de l'ensemble deviendra mauvaise. La création de fonctions de perte plus performantes est un sujet de recherche toujours très actif.

Dans les faits, la fonction de perte du Générateur est problématique. En effet, cette fonction ne permet pas à G d'apprendre efficacement. Lorsque le Générateur produit un faux et que ce dernier est détecté, il est nécessaire que le Générateur apprenne efficacement de son erreur. Or, la valeur du gradient tend à être infinitésimal lorsque la fonction de perte tend vers 0, cas obtenu lorsque la probabilité $D(G(z))$ est nulle. Au contraire, le gradient augmente lorsque le Générateur produit des faux capables de tromper le Discriminateur. Ce comportement est inadéquat car il ne permet pas au Générateur de bien apprendre lorsqu'il n'a aucune connaissance. Sur la courbe bleue de la Figure 68, nous pouvons observer le comportement du gradient selon la probabilité prédictive par le Discriminateur.

Une alternative consiste à exploiter $\log(D(G(z)))$. L'approche du comportement du Discriminateur vis-à-vis du Générateur est inversée. Au lieu de minimiser la

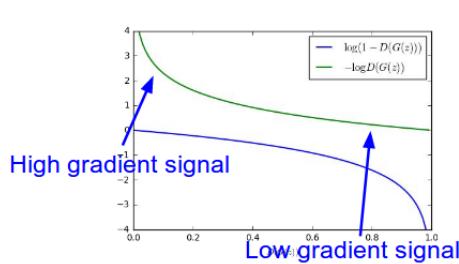


Figure 68: Comparaison des gradients de la fonction de perte du Génératuer

probabilité que le Discriminateur détecte le faux, on cherchera à maximiser le fait que le Discriminateur se trompe sur sa prédiction. Nous obtenons:

$$\max_G E_{y \sim p_y(y)} [\log(D(G(y)))]$$

Le comportement du gradient de la nouvelle fonction de perte du Génératuer est visible sur la courbe verte de la Figure 68. On peut observer que le comportement du gradient est inversé et qu'il favorise l'apprentissage lorsque le modèle est peu performant.

8.2 Difficultés d'apprentissage

Bien que très puissants, les réseaux GAN sont reconnus comme étant difficiles à entraîner. En effet:

- **Convergence:** Le risque de non-convergence est important avec ce type de réseau. L'apprentissage est alors caractérisé par un comportement très oscillant qui a tendance à finir par diverger.
- **Diversité:** Les images générées par les GAN ont tendance à être très similaires et à manquer de diversité malgré un jeu de données d'apprentissage représentatif.
- **Génération aléatoire:** Il n'est pas possible de maîtriser la génération d'images. Le comportement est exclusivement aléatoire (contrairement aux *Variational Autoencoders*).
- **Equilibre des réseaux:** Pour que le réseau GAN fonctionne, il est nécessaire que le Génératuer et le Discriminateur soit globalement aussi performant l'un que l'autre. Si l'un des modèles domine l'autre, le GAN ne sera pas capable de fonctionner (et d'apprendre) convenablement.

- **Hypersensibilité:** Les architectures GAN sont très dépendantes d'hyperparamètres difficiles à évaluer et sont sujets au sur-apprentissage

Ces différentes contraintes ont été la source d'une recherche active, notamment au niveau de la fonction de perte qui, dans sa version initiale, est globalement peu performante. De nombreuses améliorations ont été proposées à ce niveau notamment le *Wasserstein GAN*[3] qui repose sur la *distance Wasserstein*¹⁴⁰.

8.3 Deep Convolutional Generative Adversarial Networks (DCGAN)

Les réseaux DCGAN[84], contrairement aux réseaux GAN classiques, exploitent une architecture convulsive profonde pour le Générateur et le Discriminateur. Le Discriminateur repose sur les structures convolutives classiques telles que Inception, ResNet ou encore VGG. Une particularité notable est le remplacement des couches de Pooling par des convolutions avec un stride supérieur à 1 qui permettent d'avoir une réduction de dimension moins agressive en terme de perte d'informations. Au contraire, le Générateur repose sur des méthodes de *Upsampling* afin de pouvoir générer une image qui est de dimension supérieure au signal bruité exploité comme source génératrice. La méthode standard de *Upsampling* est la *Convolution transposée* (aussi appelée Déconvolution¹⁴¹).

¹⁴⁰Cette métrique permet d'évaluer la distance entre deux distributions de probabilités.

¹⁴¹Appellation abusive

9 Réseaux siamois

9.1 Généralités

Les réseaux siamois[58] constituent une famille d'architectures formées de deux (ou plus) sous-réseaux identiques, i.e même configuration avec des paramètres et poids identiques. Lors de l'apprentissage, les sous-réseaux sont mis à jour de manière identique. La nature des architectures des sous-réseaux est variable (convolutif, récurrent, Full-Connected...) et dépend de la nature des données à analyser. Ce réseau est composé de deux parties:

- **Extraction d'attributs:** Cette tâche est réalisée par les sous-réseaux qui vont représenter la donnée d'entrée sous la forme d'un vecteur d'attribut résument l'information portée. Les sous-réseaux étant identiques, le comportement des extracteurs est identique peu importe la donnée analysée (donnée de référence et donnée à analyser). Ainsi, dans le cas de la reconnaissance faciale, les deux visages à comparer seront représentés par un vecteur définissant les attributs physiques des deux individus.
- **Mesure de similarité:** Cette tâche est réalisée par une métrique de distance qui évalue la similarité entre deux vecteurs d'attributs. On peut citer la *Mean Squared Error* et la *Distance Cosinus* par exemple¹⁴². La sortie est donc une valeur comprise entre 0 et 1 qui détermine le degrés de similarité entre deux entités.

Néanmoins, ce type de métrique est peu effective dans les faits car elle favorise des convergences peu effectives, notamment en tolérant que les valeurs de sortie des entités jugées similaires et dissimilaires soient "proches". Il est préférable de favoriser un écart important entre les résultats positifs et négatifs, i.e imposer une *marge*. Cette particularité est réalisé par la *Contrastive Loss function*[28]¹⁴³. Soit \vec{X}_i , une valeur d'entrée et $G(\vec{X}_i)$, la sortie obtenue après action d'un sous-réseau et y , valeur binaire de prédiction. *Contrastive Loss function*¹⁴⁴ est ainsi définie¹⁴⁵ par:

$$(1 - y) * \frac{1}{2} D_w^2(\vec{X}_1, \vec{X}_2) + y * \frac{1}{2} (\max(0, m - D_w(\vec{X}_1, \vec{X}_2)))^2$$
$$D_w(\vec{X}_1, \vec{X}_2) = \|G(\vec{X}_1) - G(\vec{X}_2)\|_2$$

Un exemple de réseau siamois est visible sur la Figure 69. Les sous-réseaux suivent une architecture Full-connected. Les vecteurs caractéristiques sont dans R^2 .

¹⁴²Il existe de nombreuses autres métriques utilisables !

¹⁴³Il existe aussi des métriques qui respecte cette condition en ne se basant pas sur une marge

¹⁴⁴On supposera qu'une similarité parfaite équivaut à un résultat de 0

¹⁴⁵Voir la Section 9.2 pour plus de détails sur la notion de marge.

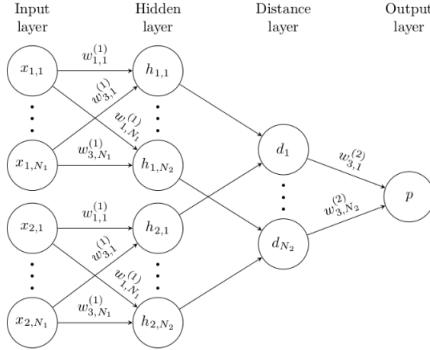


Figure 69: Exemple simple d'un réseau siamois

L'objectif de ce type de réseau est d'évaluer la similarité entre deux entités, ce qui s'oppose avec la tâche classique des réseaux de neurones qui classifient une entité. De nombreux problèmes nécessitent une comparaison entre deux entités (peu importe si le réseau est capable de les classifier), notamment la reconnaissance faciale (*Face Recognition*), de signature ou encore, d'écriture (par rapport au style graphique si écrit à la main ou au style syntaxique). Plus récemment, de nouvelles thématiques ont été abordées telles que l'évaluation de la pertinence d'une réponse à une question donnée¹⁴⁶ ou encore le *Tracking*. Ce type de réseau est très exploité dans le domaine du **One-Shot Learning**.

9.2 Triplet Loss

Une autre fonction populaire - comparable à la *Contrastive Loss function* - est la *Triplet Loss*[96]. La *Contrastive Loss function* cherche à minimiser la distance si les deux images sont de la même classe et à maximiser (associer une valeur supérieure à une marge) si les deux images sont de classes différentes. Cette métrique considère des couples d'entités et réalise sa discrimination en fonction de la concordance des entités de ce couple. Ainsi, elle maximise (ou minimise) les distances entre entités de manière dissociées.

Au contraire, *Triplet Loss* exploite des triplets d'entités (A,P,N) avec A, image de référence, P, image positive (de même classe) et N, image négative (de classe différente). L'objectif de cette métrique est de permettre que la projection de l'image positive soit plus "proche" de la référence que l'image négative, i.e maximiser la différence entre (A-N) et (A-P). Cette approche considère ainsi les distances de manière *relative*. En effet, sa référence est basée sur la différence entre (A-N) et (A-P) et non des distances *absolues*. Cette métrique est donc plus *laxiste* car elle ne forcera pas la réduction de distance qui peut

¹⁴⁶Approche très utile dans le cadre d'un apprentissage d'un ChatBot

être trop exigeante. Par exemple, dans le cas de la *Contrastive Loss function*, l'apprentissage va chercher à diminuer au maximum la distance entre deux entités similaires alors que dans le cas de *Triplet Loss*, l'apprentissage se limitera à garantir une bonne séparation entre les entités similaires et dissimilaires. La condition est donc moins stricte.

Supposons un triplet (A,P,N). Nous voulons faire en sorte que l'image positive soit plus proche de la référence que l'image négative. Supposons f, la fonction de transformation des sous-réseaux. Ainsi, nous obtenons:

$$\underbrace{\|f(A) - f(P)\|_2}_{D(A,P)} \leq \underbrace{\|f(A) - f(N)\|_2}_{D(A,N)}$$

$$\underbrace{\|f(A) - f(P)\|_2 - \|f(A) - f(N)\|_2}_{D(A,P) - D(A,N)} \leq 0 \quad (1)$$

Cette approche présente une faiblesse majeure. En effet, la condition tolère le cas où:

$$D(A, P) \rightarrow 0$$

$$D(A, N) \rightarrow 0$$

Pour contrer ce phénomène, on instaure une *marge* qui forcera un éloignement des projections des entités négatives. Ainsi, nous obtenons:

$$D(A, P) + \alpha \leq D(A, N)$$

$$D(A, P) - D(A, N) + \alpha \leq 0 \quad (2)$$

Supposons un exemple. Si on utilise l'équation (1), si $D(A,P)=0.3$ et $D(A,N)=0.33$, alors la condition est remplie. Néanmoins, la discrimination est très faible et le risque de faux-positif et faux-négatif élevé à cause de cette proximité. Supposons l'équation (2) si $D(A,P)=0.3$ et $\alpha = 0.3$ alors il faut que $D(A, N) \geq 0.6$ pour que la condition soit satisfaite, ce qui favorise une discrimination importante.

Il reste à formaliser cette condition afin de la rendre exploitable par un réseau de neurones. Nous pouvons réécrire l'équation (2) telle que pour un triplet (A,P,N):

$$\mathcal{L}(A, P, N) = \max(\underbrace{\|f(A) - f(P)\|_2}_{D(A,P)} - \underbrace{\|f(A) - f(N)\|_2}_{D(A,N)} + \alpha, 0)$$

$$\mathcal{L}(\text{minibatch}_{(A,P,N)}) = \sum_{i=1}^n \mathcal{L}(A^{(i)}, P^{(i)}, N^{(i)})$$

Attention: Si les triplets (A,P,N) sont choisis aléatoirement, la condition peut être facilement satisfaite et nuire à la performance du modèle du fait de l'apprentissage trop "facile". Il est important de créer des triplets "difficiles" à entraîner en utilisant des entités qui se "ressemblent" !

10 Réseaux récurrents

10.1 Généralités théoriques

10.1.1 Description d'un RNN

Les architectures neuronales populaires (tels que les réseaux convolutifs ou les réseaux Full-Connected) ne possèdent pas une mémoire de leurs états internes. Ainsi, pour chaque donnée, ces réseaux ne considèrent pas le contexte de la donnée traitée et traitent chaque entrée indépendamment les unes des autres. En d'autres mots, ces réseaux reposent sur l'**hypothèse forte** que chaque donnée sont indépendantes.

Bien que possiblement vraie, dans les faits, cette condition est rarement vérifiée. Souvent négligeable, notamment dans le cadre de l'analyse d'image, elle est critique lorsque la donnée possède une forte liaison avec son contexte. C'est le cas pour toute donnée sérielle ou temporelle, tout particulièrement les données textuelles qui possèdent un fort potentiel applicatif (traduction automatique, reconnaissance vocale, commentaire/labellisation d'image...).

De plus, ces architectures imposent que la dimension d'entrée et de sortie soit fixe, ce qui est problématique dans le cadre d'analyse de données à dimension variable caractéristiques des données textuelles. Pour répondre à cette problématique, les *réseaux récurrents*[19] (RNNs - Recurrent Neural Network) ont été créées.

Contrairement aux autres architectures, un RNN possède une mémoire de ses états précédents. De ce fait, un RNN évalue une donnée à l'instant $t + 1$ selon ses caractéristiques et son contexte représenté par le comportement du réseau vis-à-vis de la donnée x_{-1}, \dots, x_{-t} à l'instant $t, \dots, 0$. La mémoire est caractérisée par une connexion entre chacune des cellules du RNN permettant la propagation de l'état caché qui représente le contexte informatif.

L'idée principale d'un RNN est de compresser l'information d'une séquence d'entrée en un vecteur $k \in R^d$ par récursivité. Ce vecteur est appelé *état caché* et résume l'information, à l'instant t , du signal traité (représenté par l'ensemble des données x_0, \dots, x_t analysées). Ainsi, à l'instant t , un *état caché* possède une connaissance liée à la donnée x_t et de son contexte défini par les données x_{t-1}, \dots, x_0 . L'*état caché* obtenu à l'instant t est ensuite transmis à la cellule $t + 1$ du RNN pour participer à la création de l'*état caché* $t + 1$ qui possédera l'information des données x_0, \dots, x_t complétée par celle de la données x_{t+1} . L'*état caché* à l'instant t est exploité pour réaliser la prédiction du RNN à l'instant t .

Il est **important** de retenir que chaque cellule d'un RNN sont strictement identiques. Les seules variables sont définies par les entrées i.e l'état caché et la

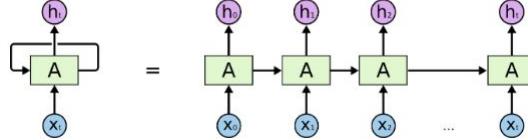


Figure 70: Représentation d'un RNN "déroulé" allant de l'instant 0 à t .

donnée de l'instant t . Une illustration d'un RNN est visible sur la Figure 70. On peut observer une version unitaire et déroulée. Cette double représentation est possible car il y a unicité de l'architecture et des vecteurs de poids de chaque cellule.

Remarque: Un état de l'Art avancé a été réalisé par Hojjat Salehinejad sur l'architecture RNN. Il est consultable via son article de recherche [94]. Cet article permet d'avoir une vision d'ensemble de l'évolution de l'architecture RNN, notamment de ces innovations les plus récentes.

10.1.2 Présentation théorique

Soit X une séquence, X_1, X_2, \dots, X_t les éléments de cette séquence tels que $X = (X_1, \dots, X_t)$. Soit h_t l'état caché du RNN à l'instant t et y_t , la prédiction du RNN à l'instant t alors:

$$h_t = \sigma_h(U_{hX}X_t + W_{hh}h_{t-1} + b_h)$$

$$y_t = \sigma_y(V_{yh}h_t + b_y)$$

Avec U , matrice de poids de dimension $h \times X^{147}$, W_{hh} matrice de poids de dimension $h \times h$, V , matrice de dimension $y \times h$, b_h et b_y , biais des couches et σ , fonction d'activation. Par convention, σ_h correspond à la fonction tanh ou ReLU et σ_y , fonction sigmoïde ou softmax selon le type de prédiction réalisée. Une illustration est visible sur la Figure 71.

La dimension de l'état caché est de dimension fixe à travers le réseau et ce dernier est initialisé comme vecteur nul par convention à l'étape $t = 0$.

Ce type de cellule est appelé *Réseau de Helman*. Il existe de nombreuses variantes de cellule notamment le *Réseau de Jordan* défini par:

$$h_t = \sigma_h(U_{hX}X_t + W_{hh}h_{t-1} + b_h)$$

$$y_t = \sigma_y(V_{yh}h_t + b_y)$$

Ces deux architectures sont très similaires. La différence repose sur le vecteur propagé entre les cellules. Dans le cas de Helman, il s'agit de l'état caché. Dans

¹⁴⁷Prends en entrée une donnée de dimension X et produit un résultat de dimension h

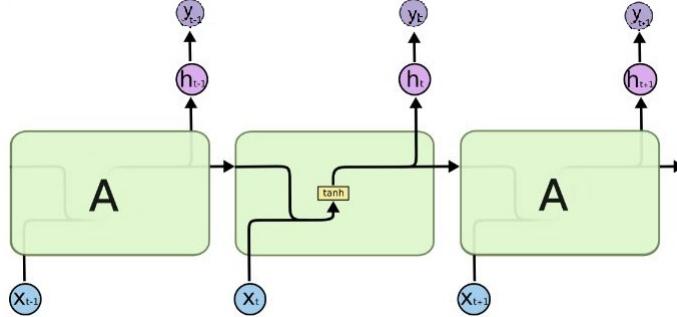


Figure 71: Exemple d'une cellule RNN.

le cas de Jordan, du vecteur de sortie de la cellule. L'approche de Jordan, en n'exploitant pas l'état caché, perd le contexte général de la séquence et se focalise exclusivement sur le contexte $t - 1$. En pratique, notamment les *Neural Machine Translation*, les deux vecteurs sont exploités afin d'obtenir un compromis entre contexte proche et contexte global.

10.1.3 Implémentation et représentation matricielle

En cours

10.1.4 Apprentissage d'un RNN

10.1.5 Problématiques d'un RNN Vanilla

Bien que théoriquement efficace¹⁴⁸, l'architecture RNN Vanilla présente des problèmes majeurs qui nuisent à son efficacité applicative.

10.1.5.1 Unilatéralité du contexte

Un RNN, à l'instant t , exploite l'information à l'instant $[0, \dots, t - 1, t]$. De ce fait, il ne possède qu'une vision unidirectionnelle pour réaliser sa prédiction. Ceci est très problématique car il biaise le contexte qui ne considère pas une direction informative.

Supposons une séquence $[x_0, \dots, x_t, \dots, x_n]$. A l'instant t , la prédiction réalisée dépendra de $[x_0, \dots, x_t]$. Les valeurs x_{t+1}, \dots, x_n ne sont donc pas considérées pour définir le contexte de la prédiction y_t . Ce défaut est critique dans le cadre du *Natural Language Processing* car le contexte d'une phrase (ou plus

¹⁴⁸Un RNN est une architecture Turing complete [100]

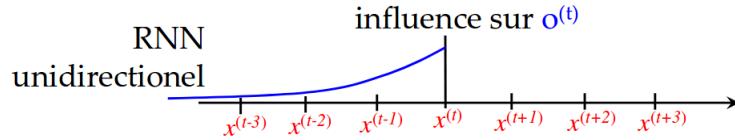


Figure 72: Influence de la donnée selon sa position dans la séquence

spécifiquement d'un mot) s'étend sur l'ensemble de la séquence considérée¹⁴⁹. Pour répondre à cette problématique, les *Réseaux récurrents bidirectionnels*¹⁵⁰ ont été proposés.

10.1.5.2 Mémoire court terme

Le vecteur de contexte (l'état caché) résume l'information de la séquence déjà observée. Dans les faits, l'information retenue est très peu uniforme et se concentre sur les dernières données observées. La Figure 72 illustre ce phénomène. Ainsi, un RNN possède essentiellement une mémoire à **court terme** et ceci est très néfaste dans le cadre d'analyse de séquences longues telles les textes ou encore les données temporelles. Pour corriger ce défaut, de nouvelles architectures de cellules ont été proposées, notamment les LSTM (*Long Short-Term Memory*) et les GRU (*Gated Recurrent Unit*).

10.1.5.3 Coût Calcul et représentativité de l'état caché

L'état caché d'une cellule RNN est de dimension fixée (supposons de dimension d) et constante. De ce fait, une hypothèse forte est faite en supposant qu'un espace de dimension d est capable de représenter le phénomène observé. Une dimension trop grande favoriserait un sur-apprentissage¹⁵¹ alors qu'une dimension trop petite ne permettrait pas la représentation de l'information utile de la donnée observée. Le choix de la dimension de l'état caché est un hyperparamètre qui peut être difficile à réaliser. Des valeurs *par défaut* ont été proposées dans le cadre d'études empiriques mais il reste difficile de *définir* une bonne valeur pour un problème donné.

Le problème de la dimension de l'état caché est directement lié au coût calcul d'une architecture RNN. Du fait du comportement séquentiel de ce type de réseau, ce dernier ne peut pas être efficacement parallélisé. Ainsi, les RNN sont des architectures coûteuses en temps machine (notamment pour l'apprentissage). Exploiter une dimension faible pour représenter l'état caché favorise une plus grande vitesse au risque d'une perte d'efficacité. Le choix de la dimension

¹⁴⁹Cette affirmation est générale mais dépend grandement de la langue étudiée. En effet, la structure grammaticale varie grandement d'une langue à une autre.

¹⁵⁰Bien qu'expérimentalement efficace, cette méthode est parfois considérée comme une méthode "cache-misère"...

¹⁵¹Apprendre par cœur chaque exemple, ce qui serait critique pour la généralisation du modèle

est donc un compromis entre *Représentativité*, *Généralisation* et *Optimisation calculatoire*.

10.2 Long Short-Term Memory (LSTM) - A FAIRE

En cours

10.3 Gated Recurrent Unit (GRU) - A FAIRE

En cours

10.4 BiRNN - A FAIRE

En cours

10.5 Architecture-type

Du fait du comportement sériel de l'architecture récurrente, plusieurs variantes sont réalisables:

- **One-to-One:** Cette architecture est la plus simple et correspond à un réseau récurrent composé d'une cellule uniquement. Ainsi, pour une entrée, le réseau produit une sortie.
- **One-to-Many:** Cette architecture prend en entrée une valeur unitaire et produit une sortie multiple. Elle est exploitée lors d'une tâche de prédiction d'un phénomène serial à partir d'une valeur donnée (série temporelle par exemple). Dans le cas où l'on veut prédire les valeurs $t+1, \dots, t+n$, il faudrait un réseau One-to-N.
- **Many-to-One:** Cette architecture est l'inverse du *One-to-Many*. À partir d'un ensemble serial, on cherche à prédire une unique valeur. On peut exploiter cette approche lorsque l'on veut prédire une valeur $t+1$ à partir d'un contexte formé des valeurs $t-n, \dots, t$.
- **Many-to-Many:** Cette architecture est caractéristique d'un *Encoder-Decoder*. L'objectif est de prédire une succession d'états à partir d'une succession d'états. La dimension de l'entrée peut être différente de la dimension de sortie (M-to-N). Cette architecture est à la base des modèles de *Natural Machine Translation* car elle permet de s'émanciper de la contrainte de la continuité dimensionnelle entre l'entrée et la sortie (i.e N=M).

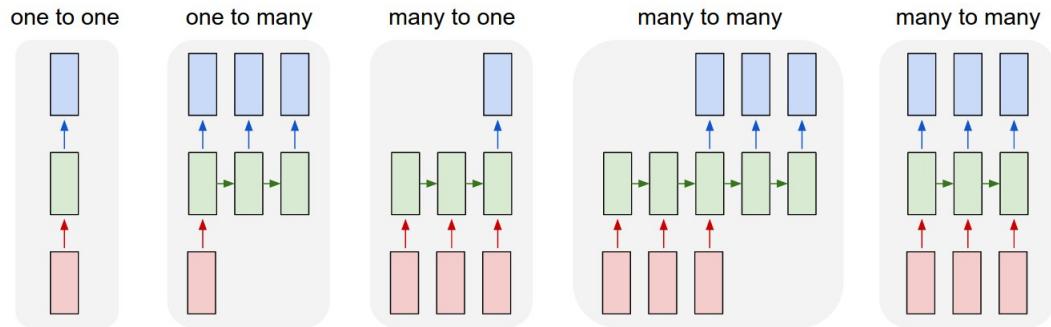


Figure 73: Architecture-type d'un réseau récurrent

- **Many-to-Many stricte:** Cette architecture est identique au *Many-to-Many* mais impose une continuité dimensionnelle entre le signal d'entrée et le signal de sortie. Cette approche est souvent représentée comme la version naïve du *Many-to-Many*, notamment pour les tâches de traduction.

Les différentes architectures sont illustrées sur la Figure 73. Elles sont indépendantes de la structure de la cellule choisie (LSTM, GRU ou RNN vanilla).

11 Deep Learning et Attention

11.1 Généralités

Attention: Cette partie va se reposer sur des architectures avancées de Deep Learning. Afin de comprendre le contexte, il est nécessaire d'avoir une connaissance générale de l'architecture des *Neural Machine Translation*, des *Encoder-Decoder* (Section 7) et des réseaux récurrents (Section 10), particulièrement les conventions d'écriture matricielle.

Pour réaliser une tâche, un réseau de neurones n'a pas nécessairement besoin de se focaliser sur l'intégralité de l'information pour produire sa décision. Par exemple, lorsqu'une traduction d'un texte est réalisée, le réseau doit se *concentrer* sur le mot qu'il traduit et son contexte uniquement. Dans le cas contraire, le risque majeur est qu'il n'arrive pas à isoler la partie discriminante nécessaire à sa prédiction et de ce fait, que sa prédiction soit mauvaise.

Ce comportement est représentable en Deep Learning grâce au procédé nommé **Attention**. L'*Attention* permet au réseau de se focaliser sur un sous-ensemble de l'information afin de mieux cibler l'information nécessaire à la tâche qu'il réalise. L'*Attention* est responsable d'une amélioration significative des réseaux de neurones, notamment pour la traduction automatique où il apporte une solution élégante à la contrainte dimensionnelle du réseau traducteur. En effet, il a été montré que plus un texte est volumineux, plus le réseau se doit d'être volumineux[11][105], ce qui est très problématique pour les contraintes matérielles actuelles. La qualité de la prédiction suit la courbe décrite par la Figure 74 lorsqu'un réseau de taille fixe voit la dimension de sa donnée d'entrée augmenter. L'*Attention* permet de conserver une performance stable malgré l'augmentation de la dimension de la donnée d'entrée pour une architecture fixée et permet ainsi, de s'émanciper de la condition de proportionnalité entre dimension du réseau et dimension de l'entrée.

Formellement, l'*Attention* consiste à construire dynamiquement une *information de contexte* (noté $c(t)$) à partir d'un ensemble de contextes locaux issus d'une donnée source (notés $h_{i \in [1, n]}$). Ainsi, $c(t) = f \circ g(h_{i \in [1, n]}, q(t))$ avec f , fonction d'*Attention*, g , fonction de proximité et $q(t)$, contexte d'état auquel les différents contextes locaux sont comparés. La détermination de $c(t)$ peut être issue d'une approche déterministe ou probabiliste.

L'*Attention* est un procédé très populaire depuis 2017 et voit son évolution rapide. Néanmoins, deux grandes familles d'architecture sont discernables: les approches *Hard Attention* et *Soft Attention*.

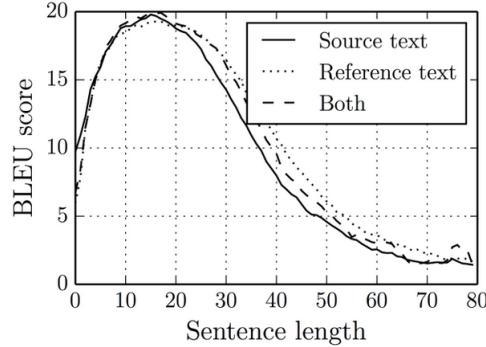


Figure 74: Performance de la traduction selon la dimension du texte d’entrée

11.2 Soft Attention

Les approches *Soft Attention* (Figure 75) reposent sur une hypothèse déterministe qui exprime le *vecteur de contexte* $c(t)$ sous la forme d’une somme pondérées des différents contextes locaux $h_{i \in [1, n]}$. Ainsi,

$$c(t) = \sum_{i=1}^n \alpha_i h_i$$

$$\alpha_j(t) = align(h_j, q(t)) = \frac{\exp(e_j)}{\sum_{j' \in [1, n]} \exp(e_{j'})}$$

$$e_j = score(h_j, q(t))$$

Les coefficients α_i sont obtenus par application de la fonction *Softmax* afin de normaliser leurs valeurs et de rendre possible leurs interprétations probabilistes. La fonction *score(.)* est la fonction qui évalue la proximité entre le vecteur référence et les contextes locaux. De nombreuses méthodes¹⁵² ont été proposées à ce jour.

Cette méthode est différentiable. De ce fait, le mécanisme d’*Attention* apprend par *rétropropagation du gradient* comme le reste du réseau. Elle est donc évolutive, souple et permet une approche *End-To-End*. Néanmoins, son impact sur le temps d’apprentissage est non négligeable. De plus, son hypothèse initiale impose que l’*Attention* puisse être convenablement représentée par une combinaison linéaire. Du fait de sa simplicité d’apprentissage et d’implémentation, *Soft Attention* est plus répandue que *Hard Attention*. Néanmoins, les performances sont variables selon les données exploitées.

¹⁵²Nous en étudierons certaines par la suite

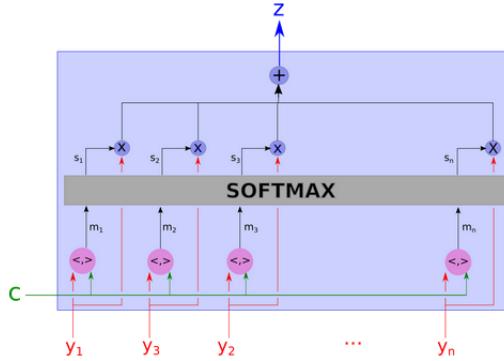


Figure 75: Schématisation de l'approche Soft Attention

11.2.1 Fonction de proximité

La fonction de proximité, appelée *fonction d'alignement*, permet de calculer la proximité entre le vecteur référence et les différents contextes locaux. De nombreuses méthodes sont proposées mais les plus répandues sont:

1. **Produit scalaire:** $score(h_j, q(t)) = h_j^T q(t)$
2. **Produit scalaire pondéré:** $score(h_j, q(t)) = h_j^T W_a q(t)$
3. **Produit scalaire normé**^[113]: $score(h_j, q(t)) = \frac{h_j^T q(t)}{\sqrt{n}}$
4. **Couche Feed-Forward**^[154]: $score(h_j, q(t)) = v_c^T \tanh(W_c[h_j; q(t)])$

11.3 Hard Attention

Les approches *Hard Attention* (Figure 76) reposent sur une méthode de *sampling* stochastique. De ce fait, elles sont probabilistes. Ces approches extraient un ^[155] vecteur local selon une densité de probabilités. Ainsi $Z_i \sim h_i, \alpha_i$.

Du fait de son hypothèse initiale, les valeurs des gradients sont estimées par des *méthodes de Monte-Carlo* et ne peuvent exploiter les gradients rétropropagés car non dérivables. Les méthodes *Hard Attention* sont donc plus difficiles à implémenter et à exploiter au sein du réseau neuronal. L'*Apprentissage par Renforcement* est très utilisé dans le cadre de ce type d'approches. Sur la Figure 77, nous pouvons observer le comportement général des deux types d'*Attention*. Alors que *Soft Attention* a une focalisation diffuse caractéristique d'une somme

^[153]Le facteur de normalisation est pertinent en cas de données à forte dimension afin de limiter l'impact d'un gradient faible, ce qui est nuisible à l'apprentissage

^[154] v_c^T est utilisé afin d'obtenir une valeur unitaire

^[155]Voir deux parfois selon la méthode

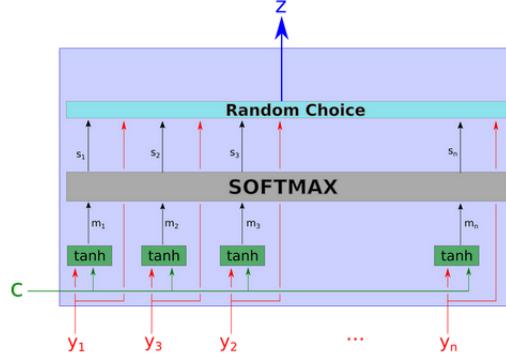


Figure 76: Schématisation de l'approche Hard Attention

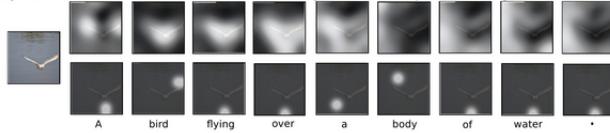


Figure 77: Visualisation du comportement de Soft Attention et Hard Attention

pondérée, *Hard Attention* se focalise sur une zone spécifique due à la sélection unitaire d'un contexte local. Ce type d'approche est moins exigeant en temps de calculs, ce qui la rend utile pour les tâches temps-réel.

11.4 Réseaux récurrents

Un réseau récurrent est caractérisé par une structure temporelle où la mémoire conservée d'un évènement x à l'instant t tend à diminuer lorsque t augmente. De même, la projection¹⁵⁶ d'une séquence dans un espace de dimension R^d impose une destruction d'informations qui peut être critique pour la conservation de son contexte. Pour palier à ce défaut, une méthode simpliste (LSTM Bidirectionnelle) a été de lire les données temporelles pour t croissant et t décroissant. Néanmoins, cette solution favorise grandement les données en début de séquence et en fin de séquence. Pour les séries de grande dimension typique des textes, cette méthode est insuffisante pour garantir une prédiction de qualité. L'exploitation de l'*Attention* est, à ce jour, la solution la plus performante pour combler cette problématique (Figure 78).

11.4.1 Approche Bahdanau

L'architecture d'*Attention* selon *Bahdanau*[5] est une méthode de *Soft Attention* démocratisée dans le cadre des *Neural Machine Translation*. Ainsi, les vecteurs

¹⁵⁶En exploitant l'état caché des cellules du RNN

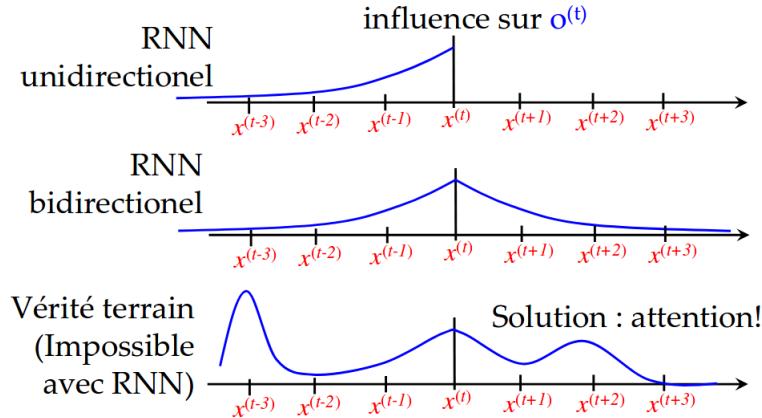


Figure 78: Attention et réseaux récurrents

de contexte locaux correspondent aux sorties intermédiaires de l'Encoder et le vecteur référence correspond à l'état caché à l'instant t du Decoder.

L'approche Bahdanau repose sur l'architecture standard d'un NMT. Ainsi, contrairement à une cellule RNN classique qui considère uniquement l'état caché précédent et l'entrée actuelle, une cellule associée à l'*Attention* selon Bahdanau recevra, en plus, un vecteur de contexte $c(t)$ **dynamiquement déterminé**¹⁵⁷. La méthode de détermination du vecteur de contexte $c(t)$ est identique à celle décrite dans la partie 11.2. Ainsi, nous avons:

$$h_{i,\text{classic}} = \phi_\theta(h_{i-1}, x_i)$$

$$h_{i,\text{bahdanau}} = \phi_\theta(h_{i-1}, x_i, c_i)$$

Afin d'uniformiser l'entrée, il est nécessaire de ramener à deux entrées uniquement. La méthode traditionnelle est de concaténer deux entrées afin de former un unique vecteur. Néanmoins, l'approche est peu détaillée dans les articles de recherche et le choix des vecteurs est variable selon les sensibilités de chacun.

Ce mécanisme est illustré sur la Figure 79. On peut constater que l'attention appliquée à l'instant t est calculé à partir de l'état caché précédent soit à l'instant $t-1$. Il n'y a donc pas de considération de l'état actuel (donc du dernier mot à l'entrée du réseau) pour déterminer le contexte.

11.4.2 Approche Luong

L'architecture d'*Attention* selon *Luong*[74] est très similaire à l'approche *Bahdanau*. La spécificité se situe au niveau de la relation entre le vecteur de contexte

¹⁵⁷Dans un NMT classique, le vecteur de contexte est fixe et correspond au dernier état caché de l'Encoder.

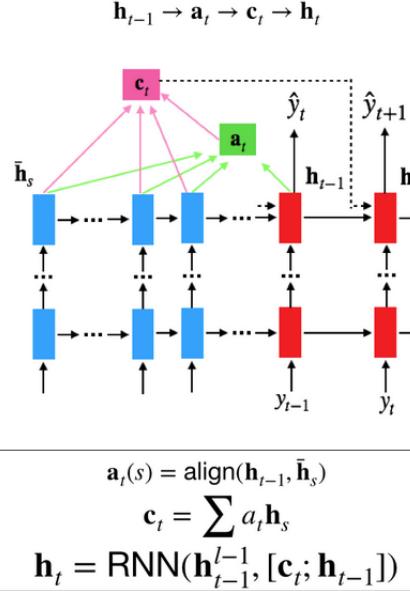


Figure 79: Attention selon Bahdanau

et l'état caché du Decoder du réseau dans le cadre de la prédiction de la sortie.

Supposons un *Neural Machine Translation*. Nous définissons \hat{h}_s , vecteurs de contexte locaux issus de l'Encoder et h_t , vecteur de référence défini par l'état caché du Decoder à l'instant t. Le vecteur $c(t)$ est le vecteur de contexte obtenu par la somme des vecteurs \hat{h}_s pondérés par les poids d'attention α_s .

Supposons y_t , prédiction du réseau à l'instant t soit un mot dans le cadre d'un NMT. Dans son architecture standard, $y_t = \text{softmax}(W_{dict}h_t)$. L'approche Luong rajoute une couche Full-Connected entre la sortie h_t et y_t qui appliquera l'attention. Ainsi:

$$y_t = \text{softmax}(W_{dict}\tilde{h}_t)$$

$$\tilde{h}_t = \tanh(W_c[c_t; h_t])$$

Ce mécanisme est illustré sur la Figure 80. On peut constater que l'attention appliquée à l'instant t est calculé à partir de l'état caché actuel soit à l'instant t. Il y a donc considération de l'état actuel, i.e le dernier mot en entrée du réseau, pour déterminer le contexte.

11.4.2.1 Attention globale et locale

L'Attention est dite *globale* lorsqu'elle exploite l'intégralité des vecteurs de contexte. Dans le cadre de vecteurs de grande dimension et/ou de séquences de données de grande taille, la contrainte calculatoire peut être trop importante

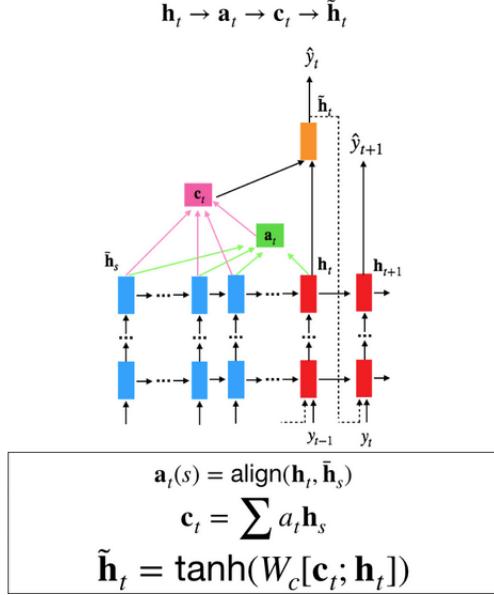


Figure 80: Attention selon Luong

pour permettre l'exploitation de cette approche, notamment pour le (quasi) temps-réel comme la traduction automatique.

Au contraire, l'*Attention locale* constitue un compromis entre *Soft Attention* (tous les vecteurs) et *Hard Attention* (un seul vecteur) en ne considérant qu'un sous-ensemble des vecteurs de contexte. Ainsi:

$$c(t) = \phi_\theta(\hat{h}_{i,i \in [p_t - D; p_t + D]})$$

avec D , hyperparamètre défini empiriquement¹⁵⁸ et p_t , référentiel inféré par le modèle pour chaque mot en entrée du Decoder.

Luong propose deux approches pour déterminer p_t : l'approche *monotonic* et *predictive*.

L'approche *monotonic* repose sur l'hypothèse que la position du mot-cible du Decoder est grossièrement alignée avec le mot source de l'Encoder. Ainsi, $p_t = t$ et constitue une constante du réseau. Cette méthode est assez peu efficace dans le cadre de la traduction automatique du fait des différences sémantiques et syntaxiques des langues.

¹⁵⁸Cette détermination peut être dure à réaliser et constitue une faiblesse importante de cette méthode

L'approche *predictive* apprend à prédire la position du référentiel afin de cibler le sous-ensemble le plus pertinent dans le cadre du mot-cible en entrée du Decoder. Supposons S , dimension de la séquence d'entrée du Decoder. Il est logique que p_t soit dans $[0, S]$. Ainsi, nous avons:

$$p_t = S \cdot \text{sigmoid}(\varphi(h_t))$$

φ est la fonction qui interprète l'état caché actuel pour définir la dimension de la fenêtre déterminée par p_t . Pour cela, une couche Full-Connected est utilisée. Nous obtenons donc:

$$\varphi(h_t) = v_p^T \tanh(W_p h_t)$$

avec v_p et W_p , paramètres du modèle à inférer durant l'apprentissage.

Afin de favoriser l'alignement autour de p_t , les poids associés à l'*Attention* sont modifiés. Ainsi, plus la position du mot est éloignée du référentiel, plus son importance sera diminuée¹⁵⁹. Supposons α_i , poids d'*Attention* du i-ème vecteur de contexte local alors:

$$\alpha_t(j) = \text{align}(h_j, q(t)) \cdot \exp\left(-\frac{(j - p_t)^2}{2\sigma^2}\right)$$

avec σ empiriquement défini à $\frac{D}{2}$, p_t nombre réel et j, entier appartenant à la fenêtre centrée en p_t .

11.4.3 Généralisation pour l'exploitation d'image: Image Captionning

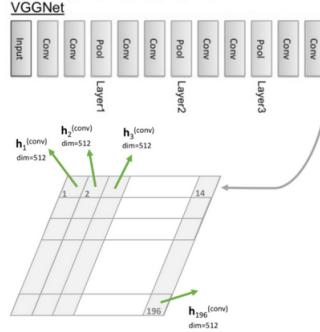
La notion d'*Attention* est applicable à toute forme de données structurées. Ainsi, une image, un texte, un signal sonore ou même quelconque peuvent être exploités par une architecture exploitant l'*Attention*. La différence se situe au niveau de la détermination des vecteurs de contexte obtenue par le réseau extracteur.

Dans le cas d'un réseau récurrent exploité comme extracteur (notamment pour les *Neural Machine Translation*, les vecteurs de contexte peuvent être associés aux vecteurs d'états cachés des cellules RNN¹⁶⁰. Néanmoins, dans le cas d'un réseau convolutif (exploité pour l'*Image Captionning* par exemple), il n'y a pas de "structure vectorielle" explicite. Il est donc nécessaire de proposer une méthode d'isolation des différents vecteurs de contexte.

Une image, à travers un réseau convolutif, est explicitée à travers les *feature map* créées par les couches de convolution. De ce fait, l'information d'une image conserve la structure de la donnée initiale soit une forme matricielle. Une sortie d'une couche convulsive est un ensemble de *feature map*. Une même localisation spatiale sur les différentes *feature map* correspond à un résultat d'analyse

¹⁵⁹Néanmoins, un mot éloigné peut avoir un poids important si son alignement est très bon malgré une distance importante.

¹⁶⁰Ou LSTM, GRU...



La dimension du vecteur de contexte est 512 car la profondeur de la sortie de la 4ième couche d'un réseau VGGNet est de 512. La valeur est différente selon le réseau et la couche convulsive exploitée.

Figure 81: Extraction des vecteurs de contexte à partir d'une feature map

d'une même partie spatiale de la donnée en amont. De ce fait, elles sont associées à une même entité alors que deux localisations distinctes sur une même *feature map* correspondent à deux analyses de deux parties distinctes de l'image en amont.

Cette spécificité permet d'isoler la structure vectorielle nécessaire à la création des vecteurs de contexte. Supposons une sortie d'une couche de convolution de profondeur D et dont les *feature map* sont *carrés*. Nous avons donc une sortie de la forme M^*M^*D . Nous obtenons ainsi M^*M vecteur de contexte de dimension D. Cette séparation permet ainsi de *découper* une image en un ensemble de vecteurs représentatifs de ses caractéristiques. Une contrainte importante est le *choix* de la couche convulsive à exploiter¹⁶¹. Le même comportement est applicable sur un signal quelconque car le comportement des *feature map* est indépendant de la nature de la donnée d'entrée. La Figure 81 illustre cette approche.

11.5 Réseaux convolutifs

L'*Attention* a été popularisée grâce à sa gestion des *vecteurs de contexte*. Cependant, il est possible d'appliquer ses concepts sur d'autres architectures neuronales, notamment les réseaux convolutifs.

Dans le cadre des RNN, les entités porteuses d'informations sont les vecteurs de contextes obtenus par les prédictions de cellules (souvent LSTM ou GRU) du réseau. L'équivalent dans un réseau convolutif sont les *feature map* obtenues en sortie des couches de convolution. Appliquer l'*Attention* à ce type de réseau

¹⁶¹Il est logique de favoriser les dernières couches avec un haut pouvoir d'abstraction

revient donc à pondérer l'importance de chacune des *feature map* produites durant l'apprentissage.

11.5.1 Spatial Transformer

Les données d'apprentissage (et à prédire) peuvent présenter des particularités géométriques variables. Ainsi, l'entité discriminante peut être décentrée au fil des images, son alignement peut être variable selon un angle θ ou encore, être représentée à une profondeur variable par rapport au référentiel du cadre de l'image (premier plan et second plan par exemple). Cette variabilité pose de grande difficulté d'apprentissage au modèle tout en forçant l'augmentation de sa complexité pour considérer les géométries variables des images. Afin de résoudre cette problématique, *Spatial Transformer*[47] (STN) propose une approche élégante.

Au lieu de traiter les données *brutes*, *Spatial Transformer* va réaliser une transformation géométrique de la donnée d'entrée (i.e *feature map*) afin de la standardiser. Il réalise donc une action comparable à un pré-traitement qui facilite grandement le travail de discrimination du modèle. Ce module n'a que peu d'impact sur la vitesse du modèle.

Son comportement peut être grossièrement comparé à l'action d'une couche de Pooling. Néanmoins, *Spatial Transformer* possède des avantages qui rend cette approche bien plus robuste. En effet, elle est:

- **Modulable:** Elle est applicable à toute architecture avec facilité.
- **Apprenante:** Elle peut être entraînée par rétropropagation du gradient. Son utilisation garantit donc l'aspect End-To-End du réseau.
- **Evulsive:** La transformation réalisée par STN est dynamique selon la donnée d'entrée. Son comportement est donc adaptatif et bien plus robuste qu'une approche statique comme le Pooling par exemple.

11.5.1.1 Transformation d'image

Afin de comprendre le fonctionnement du STN, la notion de *transformation linéaire* doit être abordée.

Supposons un point X de coordonnées (x,y). Matriciellement, les coordonnées peuvent ainsi être décrites par:

$$X = \begin{bmatrix} x \\ y \end{bmatrix}$$

Soit M, une matrice 2*2 définie par:

$$M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

La transformation linéaire T est définie par la matrice $X' = T(X) = MX$.

Ainsi, supposons:

$$M = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Alors:

$$X' = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} = X$$

Nous pouvons observer que la transformation réalisée est la *transformation identité*. En modifiant les valeurs de la matrice M , nous pouvons réaliser différentes transformations du vecteur représenté par la matrice X . Ainsi, par exemple:

$$\text{Scaling}^{162}: X' = \begin{bmatrix} p & 0 \\ 0 & q \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} px \\ qy \end{bmatrix}$$

$$\text{Rotation}: X' = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \cos \theta - y \sin \theta \\ x \sin \theta + y \cos \theta \end{bmatrix}$$

Il est possible de cumuler différentes transformations sur un même vecteur. Supposons 3 transformations définies par les matrices Q , R et S . Nous avons donc $X' = Q[R(SX)] = MK$, $M = QRS$.

Attention: L'ordre des transformations est significatif car le produit matriciel n'est pas une opération commutative¹⁶³!

Supposons que nous voulons réaliser une *translation* vectorielle. Cette transformation pose problème car, avec le modèle décrit précédemment, chaque dimension du vecteur transformé correspond à une fonction linéaire des composantes du vecteur initial. Or, dans le cadre d'une translation, nous ne réalisons pas une transformation linéaire. La translation est une transformation standard et très exploitée. Corriger ce défaut est une nécessité dans le cadre du SNT.

Pour résoudre ce dilemme, une solution consiste à exploiter une nouvelle dimension de projection vectorielle pour représenter le vecteur à transformer. Ainsi, supposons X dans R^2 alors nous présenterons X dans R^3 tel que:

$$X = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

De ce fait, nous pouvons construire une matrice représentant une transformation T telle que:

$$M = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

¹⁶²On parle d'*isotropic scaling* lorsqu'un même coefficient k est appliqué sur chaque dimension du vecteur.

¹⁶³C'est à dire que $A * B \neq B * A$

Afin de réaliser une translation, nous effectuons donc:

$$X' = \begin{bmatrix} a & 0 & e \\ 0 & d & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + e \\ dy + f \\ 1 \end{bmatrix}$$

Bien que fonctionnelle, cette approche impose la conservation de la troisième dimension qui ne sert que dans le cadre d'un jeu d'écriture. Afin de corriger ce défaut, il est nécessaire de modifier la matrice M. Supposons M telle que:

$$M = \begin{bmatrix} a & 0 & e \\ 0 & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

Alors:

$$X' = \begin{bmatrix} a & 0 & \Delta \\ 0 & d & \Delta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + \Delta \\ dy + \Delta \\ 1 \end{bmatrix}$$

Nous observons que la translation selon Δ a bien été réalisée et que l'unicité des dimensions est respectée. Nous avons donc généralisée notre modèle de transformation en permettant la réalisation de *transformations affines*.

11.5.1.2 Interpolation bilinéaire

Le modèle décrit dans le paragraphe précédent permet la réalisation de transformation vectorielle via l'aide d'un produit matriciel. Néanmoins, la structure d'une image a une contrainte importante: les coordonnées doivent être entières car un pixel possède des coordonnées entières. Or, suite à une transformation, les coordonnées obtenues peuvent être décimales. Comment pouvons nous "cibler" le pixel correspondant à des coordonnées non entières ?

Pour résoudre ce problème, l'*interpolation bilinéaire* est utilisée. Elle exploite les 4 pixels les plus proches afin de définir une valeur pour le pixel "abstrait". Le résultat est ainsi plus lisse et permet la création d'image plus réaliste.

Supposons un point P de coordonnées (x,y) et 4 points de références tels que $Q_{11} = (x_1, y_1)$, $Q_{21} = (x_2, y_1)$, $Q_{12} = (x_1, y_2)$, $Q_{22} = (x_2, y_2)$. Ces points sont représentés sur la Figure 82.

Schématiquement, nous pouvons considérer P comme un point présent sur la surface d'un carré dont les sommets sont pondérés par une valeur propre. Afin d'approximer P, il est donc nécessaire d'évaluer sa distance par rapport à chacun de ces sommets et de réaliser une moyenne pondérée de leurs valeurs selon les distances. Pour cela, nous définirons R_1 et R_2 , comme intensité du pixel intermédiaire entre deux pixels successifs selon l'axe des abscisses. Par la suite, nous définirons l'intensité de P comme valeur d'une moyenne pondérée de R_1 et R_2 . Cette approche permet donc de considérer la position spatiale de P et

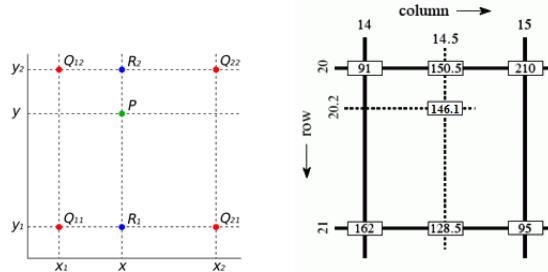


Figure 82: Illustration de l'interpolation bilinéaire

d'approximer sa valeur.

Nous définissons::

$$R_1 = \frac{x_2 - x}{x_2 - x_1} Q_{11} + \frac{x - x_1}{x_2 - x_1} Q_{21}$$

$$R_2 = \frac{x_2 - x}{x_2 - x_1} Q_{12} + \frac{x - x_1}{x_2 - x_1} Q_{22}$$

Et ainsi:

$$P = \frac{y_2 - y}{y_2 - y_1} R_1 + \frac{y - y_1}{y_2 - y_1} R_2$$

11.5.1.3 Le module SNT

Afin de réaliser une transformation affine d'une image, 3 étapes sont nécessaires:

- **Création du grille d'échantillonnage:** La grille est composée de vecteurs de coordonnées (x,y) telles que x_{max}, y_{max} égales aux dimensions de l'image initiale. En d'autres mots, il y a création d'un *meshgrid* de même dimension que l'image initiale.
- **Application de la matrice de transformation:** La matrice de transformation est appliquée sur les différents vecteurs du *meshgrid* obtenu lors de l'étape précédente.
- **Correction de la grille:** Afin de pouvoir obtenir une image valide, il est nécessaire de corriger les pixels "non entiers". Pour cela, une méthode d'interpolation est nécessaire.

Le module SNT est composé de deux parties: *Localisation Network* et *Grid generator*. Une illustration de son architecture est visible sur la Figure 84.

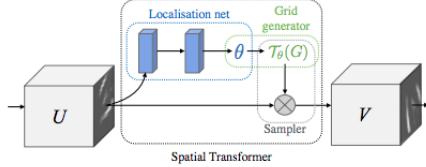


Figure 83: Illustration du module SNT

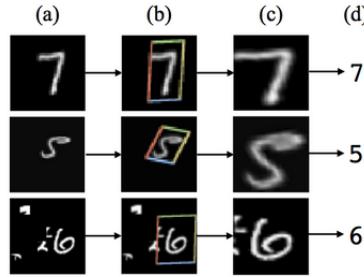


Figure 84: Illustration d'une transformation par SNT avec le dataset MNIST

L'objectif de *Localisation Network* est de produire la matrice de transformation θ , i.e une matrice $(2,3)^{164}$. Pour cela, un réseau Full-Connected (ou éventuellement convolutif) est exploité.

Ce réseau reçoit la *feature map* en entrée et il produit une sortie de dimension $(6,)$ correspondant aux paramètres de la matrice de transformation. Ce modèle est capable d'apprentissage du fait de son architecture. Il apprend à réaliser la *méilleure* transformation à appliquer sur une donnée d'entrée afin de favoriser la meilleure qualité prédictive. Il est **important** de comprendre que la transformation réalisée est dynamique et propre aux spécificités de la donnée d'entrée. Le module applique donc différentes transformations selon la donnée. Le modèle n'apprend donc pas à appliquer une transformation unique pour toutes les données mais quelle transformation réaliser selon la donnée. Bien que rien n'impose l'application d'une transformation particulière, nous avons observé expérimentalement que le comportement du réseau est comparable au comportement humain, i.e centrer l'entité discriminante et uniformiser son inclinaison selon les caractéristiques du jeu d'apprentissage. Par exemple, sur la Figure 84, nous pouvons observer le type de transformation dans le cadre du jeu de données MNIST.

La seconde partie correspond au *Grid generator*. Il est chargé de réaliser la transformation affine de la donnée d'entrée selon la matrice de transformation obtenue par *Localisation Network*.

¹⁶⁴Voir la partie précédente pour justifier l'utilisation d'une telle matrice.

En d'autres mots, il va produire le *meshgrid* et réaliser le produit matriciel avec la matrice de transformation. Afin de permettre des transformations affines (et le produit matriciel avec une matrice (2,3)), un ajout interne d'une dimension aux vecteurs du *meshgrid* est réalisé¹⁶⁵. L'action réalisée est donc:

$$\begin{bmatrix} x^s \\ y^s \end{bmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{bmatrix} x^t \\ y^t \\ 1 \end{bmatrix}$$

Cette transformation peut produire des valeurs non entières pour x^s, y^s . Le module SNT doit donc *uniformiser* les valeurs obtenues. Comme nous l'avons vu précédemment, l'application de *l'interpolation bilinéaire*¹⁶⁶ permet de corriger ce problème. Il est important de savoir qu'il existe d'autres méthodes¹⁶⁷ d'interpolation utilisables. Néanmoins, afin de conserver la capacité d'apprentissage du module (et des couches du réseau en amont), il est nécessaire que la méthode choisie soit pleinement **differentiable** pour permettre la transmission du gradient !

Remarque: Au lieu de réaliser un *meshgrid* de même dimension que la donnée d'entrée, il est possible d'augmenter ou de diminuer sa dimension. Cette particularité permet ainsi de faire du *Upsampling* ou du *Downsampling*. Du fait de sa grande efficacité et de son comportement dynamique, SNT est une alternative très sérieuse au Pooling. Néanmoins, SNT est difficilement utilisable dans le cadre d'analyse de données autres que des images. Le Pooling reste donc la référence pour l'analyse du texte ou du signal.

11.5.2 Squeeze-and-Excitation Networks (SENet)

Le modèle SENet[38](2017) est le vainqueur du ILSVRC 2017. Cette approche est extrêmement prometteuse du fait de la simplicité du concept utilisé et des bénéfices qu'il apporte.

Les *feature map* retroussent les informations obtenues après l'application d'une couche de convolution sur une donnée d'entrée. Toutes les *feature map* obtenues sont pondérées identiquement. L'idée de SENet est de les pondérer selon leurs capacités explicatives. L'approche utilisée par ce réseau est très simple:

- **Etape Squeeze:** Contraction d'une *feature map* en une seule valeur numérique représentative de cette dernière (via Global Pooling par exemple) et création d'un vecteur combinant l'ensemble des valeurs des features maps. Pour une entrée de profondeur P, le vecteur de sortie sera dans R^P .

¹⁶⁵Voir la partie précédente pour une explication détaillée de cette particularité

¹⁶⁶Cet algorithme est pleinement différentiable !

¹⁶⁷Etudier l'interpolation bi-cubique peut être intéressant.

- **Etape Excitation:** Le vecteur obtenu lors de l'étape Squeeze sera interprété par un modèle Feed-Forward (2 couches¹⁶⁸) dont la sortie sera de même dimension que l'entrée. Ce vecteur correspondra au vecteur d'excitation des *feature map*. Les nouvelles *feature map* sont obtenues par multiplication de leurs valeurs par le coefficient associé (1 par *feature map*).

La Figure 85 schématisse la différence entre un module ResNet standard et un module ResNet complété par une structure Squeeze-and-Excitation¹⁶⁹. Le coût de calcul de cette modification est inférieur à 1%, ce qui est considéré comme équivalent. De plus, cette modification peut se généraliser à tout modèle ou structure de bloc. Les auteurs ont montré qu'avec un réseau ResNet-50 avec des SE-blocs, on peut obtenir les mêmes performances qu'un modèle ResNet-101, ce qui est considérable dans le cadre d'une diminution de paramètres et donc de l'optimisation d'un modèle afin de le rendre plus rapide et léger.

11.5.2.1 Concurrent Spatial and Channel Squeeze and excitation

La recherche est active autour de cette architecture. Une amélioration significative, nommée *Concurrent Spatial and Channel Squeeze and excitation* a été proposée par [93]. L'approche initiale pondère les *feature map* les unes par rapport aux autres uniquement. *Concurrent Spatial and Channel Squeeze and excitation* propose une approche réalisant une pondération selon les *feature map* et selon les dimensions internes des *feature map*. Pour cela, deux blocks ont été introduits: sSE et scSE.

La version initiale réalise un GlobalPooling. De ce fait, chaque *feature map* est résumée par une valeur unique. Cette approche considère l'importance d'une *feature map* dans sa globalité et non l'importance de la localisation spatiale des valeurs qui la constituent. Le block sSE (Channel Squeeze and Spatial Excitation) propose de corriger ce défaut en pondérant l'importance de la localisation spatiale au sein des *feature map* selon l'information portée sur la profondeur d'une même position spatiale (différentes valeurs d'une même position spatiale à travers les différentes *feature map*). Pour cela, l'utilisation d'une convolution 1×1 est exploitée au lieu d'un GlobalPooling. On obtient donc une matrice de même dimension que les *feature map*¹⁷⁰ de profondeur 1 au lieu d'une valeur unitaire. Néanmoins, cette approche ne considère plus l'importance d'une *feature map* dans sa globalité mais uniquement l'importance de l'information portée sur une localisation spatiale des *feature map*. Pour palier à ce nouveau problème, le block scSE (Concurrent Spatial and Channel Squeeze and Channel Excitation) a été créé. Ce block est comparable à un modèle inception liant le block

¹⁶⁸La sortie de la première couche possède un nombre de sortie minoré par un ratio afin de limiter la taille du réseau

¹⁶⁹Il est important de noter que l'addition avec l'entrée est réalisée après application de Squeeze and Excitation

¹⁷⁰De même dimension que la source d'entrée

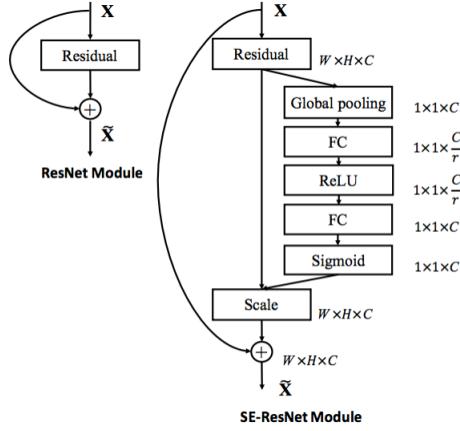


Figure 85: Comparaison entre un module ResNet standard et un module SE-Resnet

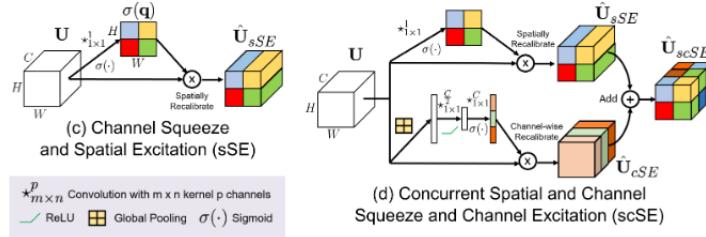


Figure 86: Illustration des blocs sSE et scSE

original (nommé cSE par [93]) et le block sSe. Les *feature map* pondérées selon les dimensions ou la généralisation sont par la suite additionnées pour obtenir les *feature map* finales. Une illustration de ces deux blocks sont visibles sur la Figure 86.

Remarque: Cette approche peut s'appliquer pour tout type de convolution (1D, 2D ou plus). Néanmoins, l'efficacité n'a pas véritablement été démontrée pour une autre donnée que l'image. Il peut être intéressant d'approfondir son analyse dans le cadre du texte, spécialement pour la tâche de classification. En effet, l'analyse d'image exploite des réseaux souvent profonds alors que la classification de texte utilise des réseaux "peu" profonds mais très larges. Cette différenciation est importante car nous ne connaissons pas le lien entre la profondeur et l'efficacité de la méthode Squeeze-and-Excitation.

12 L'analyse d'image et ses formes

12.1 Les différentes thématiques de l'analyse d'image

L'analyse d'image est un domaine d'application vaste et dont la recherche est très active. Les applications industrielles sont très riches et très diversifiées telles que l'imagerie médicale, l'automatisation des transports (véhicules autonomes) ou encore la robotique.

Les principales thématiques de l'analyse d'image¹⁷¹ sont:

- **Classification:** Cette thématique est la plus classique. L'objectif est de catégoriser les images selon l'entité qu'elle représente. La classification est non-absolue, i.e l'image est catégorisée comme positive à une classe si une entité caractérisant cette classe est représentée. Néanmoins, d'autres entités peuvent être représentées sur cette image aussi. Par exemple, une image représentant une voiture en ville peut être labellisée "voiture" malgré la présence d'autres entités comme un autre type de véhicule, une personne etc... Généralement, le label définissant l'image est le label défini comme le plus probable. Néanmoins, il est possible d'assouplir cette limitation en ne considérant pas la classe la plus fiable mais les différentes classes dont la prédiction est supérieure à une valeur seuil. Cette tâche est la plus basique et la plus exploitée aujourd'hui. Le prestige des compétitions de classification sont comparables au 100m en athlétisme et constitue la vitrine de l'analyse d'image. En effet, elle mettent en avant les innovations d'architectures neuronales et pose les fondations des évolutions permanentes qui alimenteront par la suite, les autres problématiques. Les réseaux décrites dans la Section 6 sont les solutions State-of-the-art.
- **Object Detection:** Cette thématique cherche à localiser spatialement une classe d'entité sur une image. L'objectif est donc de délimiter une sous-partie de l'image représentant la classe à détecter. Ce type d'analyse est binaire, i.e une surface peut être classée comme positive ou négative. Il n'y a donc qu'un type d'entité détectable possible. La surface classée positivement est souvent délimitée par un rectangle. L'un des problèmes les plus étudiés correspond à la reconnaissance faciale (*Face Recognition*).
- **Object Recognition:** Cette thématique généralise *Object detection*. Elle cherche à détecter de multiples entités correspondant à différentes classes dans une image et à déterminer leurs localisations. Cette méthode est très utilisée par les outils de mesure et de surveillance notamment par les drones et les caméras de surveillance.
- **Object Tracking:** Cette problématique est associée aux contraintes de mouvements et de temps réel. L'objectif est de repérer une (des) entité(s)

¹⁷¹Les noms des thématiques sont relativement flous et inconstants dans la littérature. Il est possible que certaines personnes en utilisent d'autres voire associent des noms à un autre problème. Retenez le contexte des problèmes plutôt que les noms !

et de réaliser un suivi en temps réel en gardant l'attention sur ces objets. Cette problématique est associée au support vidéo et peut être vue comme une généralisation de *Object recognition* avec une contrainte de temps-réel. Néanmoins, la considération de la dimension temporelle soulève plusieurs difficultés notamment la définition d'un phénomène temporel défini sur plusieurs images successives ou la variation d'une entité à suivre comme un individu qui porte puis enlève un chapeau. La classification de l'entité n'est pas impératif. Sous contraintes de vélocité, de nombreux algorithmes de *Tracking* néglige la classification. En effet, la présence du phénomène de *tracking* est associée à la présence d'une entité "de valeur"¹⁷², ce qui est une information souvent suffisante à l'exploitation métier. Le *Tracking* se découpe en deux problématiques principales qui sont le *Tracking* monocible et le *Tracking* multi-cible.

- **Semantic Segmentation:** Cette approche cherche à analyser une image au niveau du pixel. C'est-à-dire qu'elle cherche à assigner un pixel à une classe. La discrimination spatiale des entités sur l'image est ainsi bien plus importante. Néanmoins, la différenciation se fait au niveau sémantique et non des instances, de ce fait, seulement une différenciation de classe est réalisée et non des entités-même¹⁷³.
- **Instance Segmentation:** Cette approche est similaire à *Semantic segmentation* mais la discrimination se fait au niveau des entités et non des classes. Ainsi, en reprenant l'exemple d'une foule, l'objectif de *Instance segmentation* est de discriminer l'ensemble des individus de manière unitaire et non un ensemble correspondant à une classe "individus". Cette méthode est très utilisée en imagerie médicale notamment dans l'analyse de cellules cancéreuses.
- **Object Proposal:** Cette approche correspond à une analyse exploratoire. Elle cherche à isoler des sous-parties d'une image susceptible de contenir une entité de valeur.
- **Text Detection:** Cette thématique est associée à la détection de texte dans une image. Elle a pour objectif de détecter, localiser et extraire les entités textuelles présentes sur une image. Cette problématique est souvent associée à la transcription d'un texte manuscrit (en photo) ou de fichiers non éditables caractérisant un texte tels qu'un scan sous un format d'image (.jpeg ou .png par exemple). Une thématique connexe (*Handwriting recognition*) est souvent associée et consiste à unir un texte à un auteur selon les spécificités du style d'écriture¹⁷⁴.

¹⁷²On ne sait pas de quelle classe est l'entité précisément mais on sait que c'est une entité d'importance.

¹⁷³Supposons une foule d'individus dense, Semantic segmentation ne sera pas capable de différencier chacun des individus et généralisera à une surface globale classifiée comme "individu".

¹⁷⁴Cette approche se base quasi-exclusivement sur l'analyse du style graphique d'écriture. Le style syntaxique n'est pas traité.

- **Landmark Detection:** Cette problématique correspond à la détermination de la position d'un point de référence d'une entité et de son orientation par rapport à un système de coordonnée (par exemple, l'étude de points spécifiques déterminant la posture d'un individu (*Pose Estimation*)). Cette problématique est très utilisée par les systèmes autonomes afin d'avoir une meilleure compréhension de son environnement local et des interactions à réaliser. L'exemple populaire de ce type d'outils sont les filtres de photo/vidéo d'applications mobiles telles que Snapchat ou Instagram.
- **Image Captionning:** Cette problématique est associée à la génération d'une description textuelle d'une image. Elle unit l'analyse d'image (Computer Vision) et le Traitement du langage naturel (Natural Language Processing).

Un exemple illustratif de ces différents problèmes est visible sur la Figure 87.

12.2 Généralités théoriques

12.2.1 Théorie - Object Detection, un problème de Régression

Supposons une problématique d'*Object Detection*. L'objectif est donc de détecter la classe d'une image et de localiser l'entité de valeur sur l'image. Pour isoler l'entité de valeur, nous chercherons à réaliser un rectangle qui délimitera sa surface.

Un rectangle peut être défini de plusieurs manières¹⁷⁵. Dans notre configuration, nous définirons un point qui déterminera le centre (b_x, b_y) de l'entité de valeur et (b_h, b_l) , hauteur et largeur du rectangle. Une illustration est visible sur la Figure 88.

L'objectif du réseau prédictif est donc de prédire une classe d'image et 4 coordonnées spatiales (b_x, b_y, b_h, b_l) . Supposons un réseau qui doit discriminer 3 classes: (1) chien, (2) chat, (3) voiture. La première problématique est de définir la structure de la sortie du réseau (et donc du label des données). Pour rappel, il est nécessaire d'avoir les différentes coordonnées du rectangle délimiteur (bounding boxe) et le label de la classe d'image sachant qu'une image peut ne pas représenter une classe d'objet. Une représentation peut donc être:

Soit y , la sortie du réseau prédictif tel que:

$$y = [p_c, b_x, b_y, b_h, b_l, c_1, c_2, c_3]$$

- p_c : valeur binaire - Si une classe est représentée, $p_c=1$ sinon 0.
- b_x, b_y, b_h, b_l : valeur numérique - Coordonnées du rectangle délimiteur où b_x, b_y , coordonnées du point central de l'entité.

¹⁷⁵La manière de représenter le rectangle est déterminée par la méthode employée dans le jeu d'apprentissage. Cette méthode doit être identique pour toutes les données !

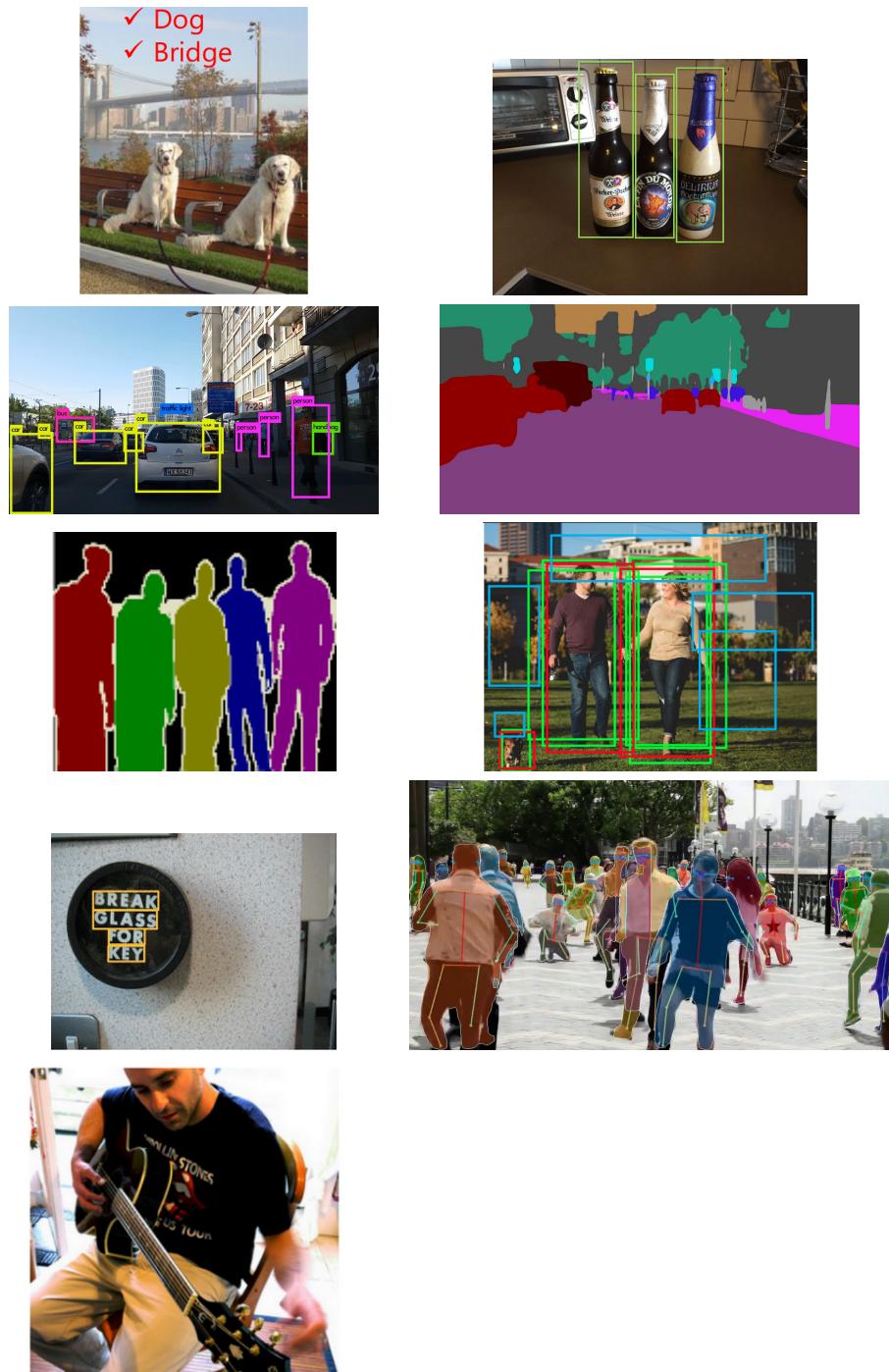


Figure 87: Exemple d'analyse d'image¹⁹⁶: 1) Classification, 2) Object detection, 3) Object recognition, 4) Semantic segmentation, 5) Instance segmentation, 6) Object proposal, 7) Text Detection, 8) landmark Detection (Pose estimation), 9) Image Captionning

- c_1, c_2, c_3 : valeur binaire - Si la classe i est représentée, $c_i=1$ sinon 0. Seule une classe peut être représentée sur une image.

Supposons une image représentant un chien, alors la sortie du réseau prédictif sera de la forme:

$$y = [1, b_x, b_y, b_h, b_l, 1, 0, 0]$$

De même, supposons une image ne représentant aucun classe, nous obtiendrons:

$$y = [0, b_x, b_y, b_h, b_l, 0, 0, 0]$$

Il est **important** de savoir que dans le cas d'une image non classée, le contenu du vecteur de sortie ne présente **aucune importance** en terme de valeurs exceptée la valeur de p_c . Les valeurs du réseau peuvent donc être arbitraires, de même que celles du vecteur référence pour l'apprentissage.

Il reste à définir une fonction de coût. Observons que nous avons 3 types de problématiques: le problème bi-classe (p_c), le problème de régression (b_x, b_y, b_h, b_l) et le problème multi-classe (c_1, c_2, c_3). Il est donc nécessaire d'utiliser une fonction de coût qui soit capable de considérer ces trois problèmes. De même, elle doit être capable de considérer la particularité de l'absence de classe sur une image.

Définissons y comme le label d'apprentissage et y' , label prédit. Une solution peut être définie par:

Si $y_{p_c} = 1$:

$$\mathcal{L}(y, y') = \mathcal{L}_{binary}(p_c, p'_c) + \mathcal{L}_{regression}(\sum b, \sum b') + \mathcal{L}_{Nclasse}(\sum c, \sum c')$$

Si $y_{p_c} = 0$:

$$\mathcal{L}(y, y') = \mathcal{L}_{binary}(p_c, p'_c)$$

\mathcal{L}_{binary} doit être capable de mesurer l'erreur d'un problème binaire. *Cross entropy loss* est donc une possibilité. Pour $\mathcal{L}_{regression}$, *Mean Squared error* est une possibilité et pour $\mathcal{L}_{Nclasse}$, *Negative Logarithmic Likelihood*¹⁷⁶. Il est possible d'utiliser d'autres fonctions tant qu'elles respectent les particularités de chaque problématique.

12.2.2 Théorie - Landmark detection, un problème de Régression

Cette approche est similaire à une généralisation de l'*Object Detection*. Supposons un problème de détection de points de références pour une analyse de visage par exemple. Ces points peuvent déterminer la position des yeux, des lèvres etc... Un exemple est visible sur la Figure 89.

Au lieu de déterminer un rectangle, cette problématique cherche à déterminer la localisation de différents points soit des couples de la forme (pr_x, pr_y) . Ainsi,

¹⁷⁶Il ne faut pas oublier le danger du $\log(0)$!

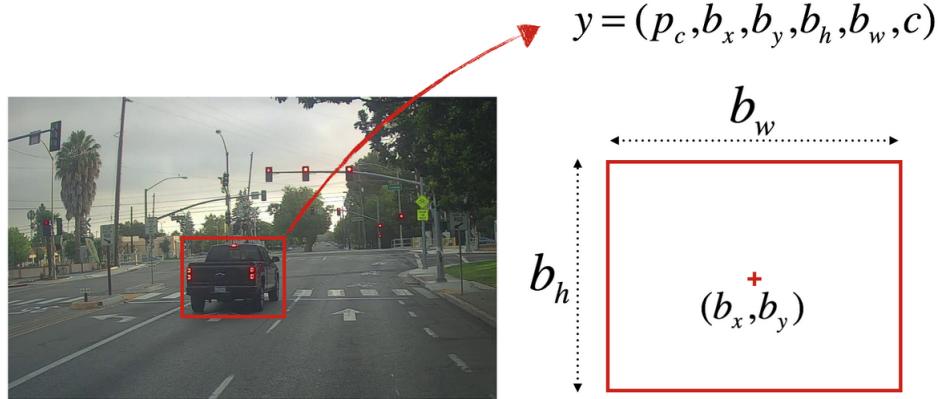


Figure 88: Exemple - Object Detection



Figure 89: Exemple - Landmark Detection

supposons la présence de 128 points de références, il faudra donc 128 couples (pr_x, pr_y) soit 256 valeurs. Il est important de considérer la possibilité de présence ou non de la classe qu'on cherche à déterminer. On supposera un système qui reconnaît qu'une classe spécifique. Il faudra donc une variable binaire comme vu dans la section précédente. Ainsi, le vecteur de prédiction sera de la forme:

$$y = [p_{classe}, \sum_{128} (pr_{x,i}, pr_{y,i})]$$

12.2.3 Sliding Windows

Object Recognition se distingue de *Object Detection* par le nombre d'entités détectables sur une image. En autorisant la détection d'un nombre indéterminé d'entités, la dimension de sortie des réseaux neuronaux n'est plus déterminable à l'avance. Il est donc nécessaire de proposer une approche qui permet de résoudre des problèmes de multi-détection en se ramenant à un problème mono-détection. Une solution (historique) s'appelle *Sliding Windows*.

Supposons une image I_{raw} de dimension $N \times N$. L'algorithme du *Sliding Windows* consiste à définir une *fenêtre* (window) de dimension $M \times M$ telle que $M \leq N$ qui

isolera une sous-partie de l'image initiale que nous appellerons $I_{crop,i,j}$ avec i,j coordonnées du point de référence¹⁷⁷ de I_{raw} . Nous supposerons que ce point de référence est indiqué par le coin supérieur gauche de la fenêtre. Dans notre exemple, il y a N^2 points de référence. Nous souhaitons que notre fenêtre soit intègre, i.e qu'elle soit contenu dans l'image initiale. Il y a donc $(N-M+1) \times (N-M+1)$ position possible pour notre fenêtre. Ainsi, nous faisons *glisser* notre fenêtre sur les $(N-M+1) \times (N-M+1)$ positions possibles. Une illustration est visible sur la Figure 90.

Nous obtenons ainsi un ensemble d'images caractérisant l'image initiale. Chacune des images est analysée par un réseau prédictif (convolutif) et une classification est réalisée. Le réseau prédictif doit déjà être entraîné sur la tâche à réaliser et peut être binaire (classification mono-classe) ou multi-classe.

Ce cycle est réalisé avec différentes dimensions de fenêtre (grande et petite) afin de réaliser une discrimination efficace de l'image. De même, le pas de glissement de la fenêtre peut être différente qu'unitaire afin de balayer plus grossièrement l'image à traiter. Une particularité est **importante**. Du fait de la variation de la dimension de la fenêtre, la dimension de l'image à analyser par le réseau diffère. Un réseau de neurones peut traiter une donnée avec une dimension fixe uniquement. Il est donc important de s'assurer que le réseau reçoit une donnée de dimension unique. Deux méthodes sont possibles: exploiter autant de réseaux distincts qu'il y a de dimension possible aux fenêtres. Cette méthode est sans doute la plus efficace mais particulièrement gourmande en temps d'apprentissage. La seconde approche est de réaliser un redimensionnement de l'image avec des méthodes de *Downsampling* ou *Upsampling*.

Dans les faits, cette méthode n'est pas exploitable avec des approches neuronales du fait du temps calculs bien trop élevé. Le nombre d'étape *Forward* est bien trop important à cause de la quantité d'images obtenues par le fenêtrage. Cette méthode est, cependant, encore employée avec des outils de classification plus rudimentaires et donc rapides mais la qualité de la prédiction est diminuée. Malgré tout, l'idée du *Sliding Windows* est encore au coeur des algorithmes à l'état de l'art. Seule son implémentation diffère afin de le rendre exploitable avec des approches neuronales. De plus, *Sliding Windows* présente un problème majeur: la position de ses fenêtres. En effet, les fenêtres sont de tailles fixes et le déplacement de la fenêtre est fixé. Il est donc probable que de nombreuses entités ne puissent être parfaitement localisées. Un exemple de cette limitation est visible sur la Figure 91. Ce type d'approche est donc peu efficace en cas d'entités aux dimensions variables. Il est, cependant, possible d'exploiter l'algorithme du *Sliding Windows* avec différentes dimensions de fenêtre mais le coût en calcul devient rapidement trop important du fait de la multitude de dimension nécessaire pour envisager une détection viable.

¹⁷⁷On peut associer un point de référence à un pixel de la matrice d'image

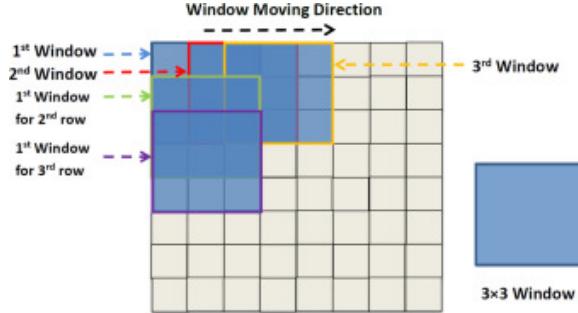


Figure 90: Exemple - Sliding Windows

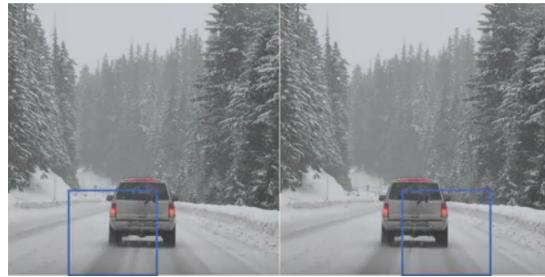


Figure 91: Limitation du Sliding Windows

12.2.4 Sliding Windows et réseau convolutif

Dans la Section 6.4, nous avons vu comment convertir une couche Full-Connected en couche de convolution. La méthode *Sliding Windows* est trop gourmande en temps machine comme vu dans la Section précédente. Une implémentation intégralement convulsive est possible pour limiter le coût de calcul de cet algorithme.

Observons la Figure 92. Le réseau du haut présente un réseau convolutif standard. L'image d'entrée est de 14×14 (on négligera la profondeur pour l'exemple) et la sortie 1×1 . L'intégralité de l'information portée par l'image d'entrée est donc résumée par la sortie de ce réseau.

Supposons maintenant une image de dimension plus grande (16×16). En gardant la même configuration de réseau que celui décrit précédemment, on observe une sortie de la forme 2×2 . Du fait du comportement de fenêtre glissante des couches de convolutions, l'action de *Sliding Windows* est réalisée par le calcul-même des convolutions. Observons le rectangle rouge de la donnée d'entrée, il correspond à une matrice 14×14 (similaire à la matrice d'entrée de l'autre réseau). Nous observons au fil du réseau que l'information portée par cette surface de la matrice est expliquée par le carré supérieur droit de la sortie. En

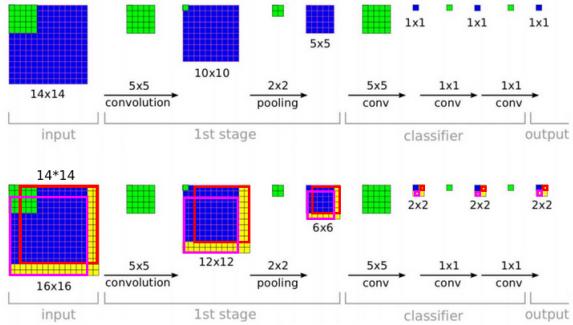


Figure 92: Exemple - Convulsive Sliding Windows

prenant l'exemple du rectangle mauve, il va s'agir du carré inférieure gauche. Nous pouvons donc constater que ce réseau est similaire à l'action du *Sliding Windows* pour une fenêtre carrée de taille 14 et de stride 2 (réalisée par l'étape du Pooling). Cependant, il n'a réalisé qu'une étape Forward pour réaliser ces prédictions. Un gain de temps majeur est donc réalisé en limitant la redondance de calculs.

12.2.5 Region Proposal

Au lieu de se limiter à l'optimisation de calcul du *Sliding Windows*, une approche visant à optimiser le nombre de fenêtres (de dimension **non fixe**) à analyser serait intéressante. Cette approche est exploitée par les algorithmes *Region Proposal*.

L'objectif de ces algorithmes est de déterminer des régions d'intérêts sur une image, i.e déterminer des *bounding boxes* susceptibles de contenir une entité. Il est **important** de comprendre que ces algorithmes ne réalisent pas de classification. Ils isolent des parties de l'image qui pourraient contenir une entité sans analyser la nature de l'entité isolée. La condition principale de ces algorithmes est d'avoir un grand *Rappel*. En effet, il est possible de filtrer des faux positifs avec les algorithmes de classification. Par contre, si une entité n'est pas isolée par une *bounding box*, elle ne pourra *jamais* être détectée par l'algorithme d'*Object Detection*. Il est donc nécessaire de ne pas être trop sévère sur la création des fenêtres. Ainsi, les *bounding boxes* créées peuvent être bruitées, se superposer mais il est probable que l'une d'elles isolent l'entité de manière satisfaisante.

Region Proposal repose sur la notion de **segmentation** d'image. La *segmentation* permet de grouper différentes régions de l'image selon des critères de similarité tels que l'analyse de couleur, de texture¹⁷⁸... Ces régions unies dis-

¹⁷⁸Ces analyses relèvent de l'analyse d'image traditionnelle. Elles ne seront pas approfondies dans ce cours

criminent des surfaces d'intérêts et permettent de définir des *bounding boxes* de dimension variables et spécifiques à ces zones d'intérêts tout en étant moins nombreuses qu'une recherche exhaustive comme réalisée par *Sliding Windows*.

Il existe différents algorithmes de *Region Proposal*. Le plus populaire et utilisé est **Selective Search**, notamment du fait d'être la méthode employée par des méthodes d'*Object Detection* parmi les plus performantes¹⁷⁹ (R-CNN et Fast R-CNN).

12.2.5.1 Selective Search

Selective Search est un algorithme de *Region Proposal* dans le cadre de la détection d'objet. Son objectif est d'avoir un fort *Rappel* et d'être rapide. La segmentation de l'image est réalisée par une méthode *graph-based* développée par Felzenszwalb and Huttenlocher[21]¹⁸⁰. Un exemple d'application de cette méthode est visible sur la Figure 93.

Selective Search exploite une image segmentée et réalise une procédure itérative en deux étapes:

- Création des *bounding boxes* correspondant aux différentes surfaces segmentées
- Union des surfaces segmentées qui présentent une similarité élevée

Cette procédure est répétée jusqu'à ce que l'image segmentée ne soit plus modifiée par l'étape d'union ou que l'image présente une unique catégorie. Un exemple est visible sur la Figure 94.

La mesure de similarité de *Selective Search* est une combinaison linéaire de 4 autres métriques sous-jacentes: similarité par couleur, par taille, par texture et par forme. Nous ne détaillerons pas ces métriques, veuillez vous référer à l'article associé.

12.2.6 Anchor Boxes

Bien qu'efficace, *Selective Search* est un point faible de plusieurs algorithmes de l'état de l'art. Plus lent que l'étape Forward d'un réseau convolutif, il est le facteur limitant à la vitesse des systèmes actuels (R-CNN et Fast R-CNN). Une solution alternative a été créée: **Anchor boxes**.

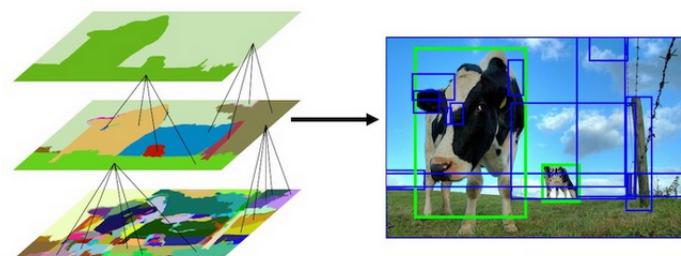
Anchor Boxes définit des fenêtres prédefinies avec des tailles différentes (64*64, 128*128 par exemple) et des ratios différents (1:1, 2:1, 1:2 par exemple) centrées

¹⁷⁹Ces méthodes tendent à devenir obsolètes avec les dernières avancées

¹⁸⁰Nous ne détaillerons pas cette méthode. Il faut juste retenir que c'est une technique de segmentation d'image. Vous pouvez lire l'article de recherche indiqué si vous souhaitez approfondir son étude.

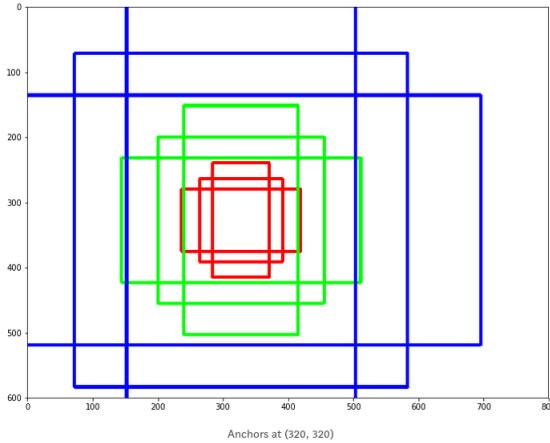


Figure 93: Exemple - Graph-based Segmentation



La suppression des bounding boxes doublons ou inexacts n'est pas réalisée par Selective Search

Figure 94: Exemple - Selective Search



3 tailles: 128x128, 256x256, 512x512
3 ratios: 1:1, 1:2, 2:1

Figure 95: Exemple - Anchor Boxes

en un même centre. Une illustration est visible sur la Figure 95.

Supposons l'image d'entrée comme une surface quadrillée selon chaque pixel. Une image 200*200 serait donc équivalent à un quadrillage 200*200. Nous définissons nos *anchors* selon l'exemple décrit sur la Figure 95. Il y a donc 9 *anchors* distincts. Un centre d'*anchors* est positionné sur chaque case du quadrillage. Il y aura donc $200*200*9=360\ 000$ *anchors* distincts. Dans les faits, il n'est pas nécessaire d'imposer une aussi grande sensibilité¹⁸¹. Il est donc possible d'appliquer un stride sur le positionnement des centres. Par exemple, supposons un stride de 20. Il y aura donc $11*11$ centres soit 1089 *anchors* distincts. Bien que ce nombre puisse impressionner, il est faible (ou équivalent selon le cas) comparé aux nombre de fenêtres obtenues par *Sliding Windows* et *Region Proposal*. Grossièrement, nous pouvons donc dire que *Anchor Boxes* réalise le même travail (détermination de *bounding boxes*) que *Region Proposal* excepté qu'il produit des *bounding boxes* de dimensions prévues à l'avance. Ce type d'approche est efficace car il est plus facile d'adapter les dimensions d'une fenêtre que de les prédire intégralement. Ainsi, on peut créer des *anchors* dont la forme est grossièrement adaptée à la forme de l'entité à observer - par exemple, une fenêtre haute et peu large pour un individu - et le réseau s'occupera d'affiner les dimensions pour coller au mieux à l'objet observé.

12.2.7 RoI Pooling

La création de *bounding boxes* soulève une problématique importante: la dimension variable des fenêtres. Cet aspect est critique car un réseau neuronal

¹⁸¹Un pixel ne "sépare" pas deux entités distinctes en général...

ne peut accepter qu'une entrée à valeur fixe. Il est donc important de proposer une méthode d'uniformisation des dimensions des *bounding boxes*.

Une autre condition d'optimisation est caractéristique de cette méthode. En effet, il est tout à fait envisageable d'extraire chaque sous-partie d'une image discriminée par une *bounding boxes* et de réaliser un redimensionnement (cropping, warping...). Bien que parfaitement fonctionnelle, si on suppose qu'il y a 1000 *bounding boxes* créées, cette technique va imposer 1000 étapes Forward aux couches de convolution suivantes pour extraire les attributs de l'image¹⁸² analysée. En effet, avec cette approche, on considère une sous-partie de l'image d'origine, pas une *feature map* issues d'un réseau convolutif qui a traité cette image. Ceci est très coûteux en temps de calcul. En effet, il est probable que de nombreuses *bounding boxes* se chevauchent. Calculer leurs sorties d'un réseau convolutif reviendrait à réaliser de la redondance calculatoire en réalisant les mêmes calculs plusieurs fois sur une même surface de la matrice de l'image d'origine. *RoI Pooling* propose une approche qui permet de s'émanciper de l'image initiale brute et d'exploiter les *feature map* qui lui sont associées uniquement. Cette particularité est très importante car elle permet de réaliser l'extraction d'attribut via un réseau convolutif **avant** le redimensionnement et non après.

L'algorithme *RoI Pooling* considère une entrée composée des *feature map* obtenues lors de l'extraction d'attributs par les couches de convolution en amont et d'une matrice de dimensions $N \times 5$ ¹⁸³ qui déterminent les *bounding boxes* définies indépendamment (avec *Selective Search* par exemple).

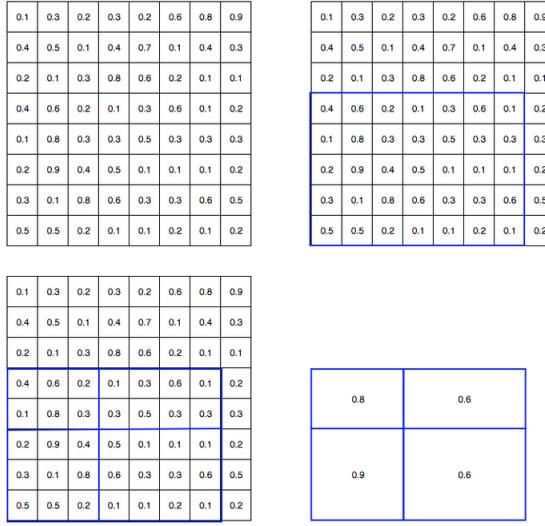
La méthode se divise en 3 parties:

- Isolation d'une sous-partie de la *feature map* en accord avec la *bounding boxes*.
- Division de la surface délimitée par la *bounding boxes* en parties équivalentes en accord avec la dimension de sortie souhaitée. Par exemple, pour obtenir une sortie de dimension 2×2 , il faudra réaliser 4 groupes. Il est possible que ce ne soit pas toujours possible, il n'y a donc pas de condition d'égalité stricte des groupes. La dimension correspondante est donc un **arrondi**. Cette spécificité provoque donc de légers décalages qui n'ont pas de réelles influences pour les problèmes d'*Object Detection* mais nuisent **grandement** pour les problèmes de *Segmentation* qui évalue une entité au pixel-près¹⁸⁴.
- Sur chaque sous-ensemble, on réalise un Max-Pooling, i.e extraire la valeur la plus élevée.

¹⁸²Sous-partie de l'image de départ délimitée par une *bounding boxes*

¹⁸³N correspond au nombre de *bounding boxes* prédites

¹⁸⁴Une approche plus conservatrice a été développée pour la Segmentation sous le nom de RoI-Align - Voir Section ??



Input feature map (top left), output feature map (bottom right), blue box is the ROI (top right).

Figure 96: Exemple - RoI Pooling

Supposons que nous voulons uniformiser les dimensions de sortie à 2×2 sachant que la dimension des *feature map* est à 8×8 et cela, pour toutes *bounding boxes* possibles. Un exemple de cette application de *RoI Pooling* est visible sur la Figure 96.

12.2.8 Intersection over Union (IoU)

Afin d'évaluer la performance des prédictions réalisé dans le cadre des thématiques vues précédemment, il est nécessaire de définir une métrique de performance. En effet, les métriques standards telles que l'Accuracy ou la Précision sont inefficaces pour juger de la validité des *bounding boxes*. Afin de résoudre ce problème, la métrique *Intersection over Union* est utilisée. Une illustration est visible sur la Figure 97.

Supposons une image d'apprentissage. Une *bounding boxes* de référence est définie et l'objectif de notre modèle est de la reproduire. Ainsi, si la *bounding boxes* prédite est strictement identique à la *bounding boxes* d'apprentissage, alors la prédiction peut être jugée comme *parfaite*. Deux *bounding boxes* identiques signifient que leurs coordonnées sont identiques, et de ce fait, il en va de même pour la surface qu'elles englobent. Cette particularité est à l'origine de la métrique IoU.

Considérons l'intersection de deux *bounding boxes*. Une intersection entre la *bounding boxes* prédite et d'apprentissage de même dimension que cette dernière

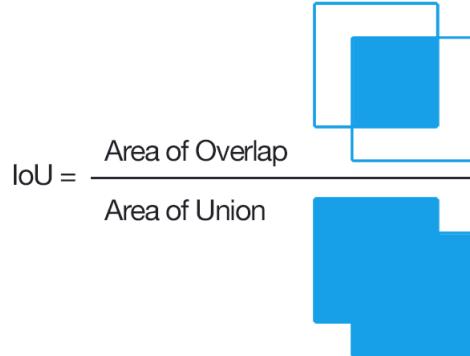


Figure 97: Illustration de la métrique Intersection over Union (IoU)

signifie que l'entité a été entièrement détectée. Néanmoins, ce résultat possède une faiblesse majeure car elle ne considère pas la précision de la *bounding boxes* prédite. Ainsi, supposons une image quelconque. Si la *bounding boxes* prédite englobe l'intégralité de l'image, alors l'intersection entre les deux surfaces sera de la dimension de la *bounding boxes* d'apprentissage mais le résultat sera très mauvais. Il faut donc un critère discriminant la précision de la *bounding boxes* prédite. Ce problème est corrigé par la considération de la surface d'union des deux *bounding boxes*. En effet, si les deux *bounding boxes* sont identiques, alors la surface d'union et d'intersection seront identiques. Au contraire, si elles diffèrent, la surface obtenue sera supérieure à la valeur de la *bounding boxes* d'apprentissage.

On obtient donc une relation permettant de considérer ces deux spécificités:

$$IoU(s, s') = \frac{s \cap s'}{s \cup s'}$$

Si les deux surfaces sont identiques, alors $IoU(s, s')=1$ et tend vers 0 lorsqu'elles diffèrent. Il s'agit donc d'une métrique normalisée, ce qui facilite son exploitation. Il est commun de considérer une *bounding boxes* prédite comme de qualité si $IoU(s_{ref}, s_{pred}) \geq 0.5$. C'est la valeur de référence pour la plupart des compétitions officielles exploitant cette métrique.

12.2.9 Non-Max Suppression

Lors de la prédiction des *bounding boxes*, une même entité peut être associée à plusieurs *bounding boxes*. Ce phénomène est illustrer sur la Figure 98. Il est donc nécessaire de filtrer ces différentes prédictions afin de ne garder que les meilleures et associer à une entité, une unique *bounding boxes*. Un algorithme de tri existe et s'appelle *Non-Max Suppression*.

Supposons un problème d'*Object Detection* à 3 classes. Une prédiction de sortie est donc de la forme:

$$y = [p_c, b_x, b_y, b_h, b_l, c_1, c_2, c_3]$$

Cet algorithme s'effectue en deux étapes. La première est la suppression de toute *bounding boxes* dont la confiance de la détection est inférieure à une valeur-seuil définie (supposons 0.7) donc si $p_c < 0.7$. La seconde étape est une procédure itérative qui réalise la sélection de la meilleure *bounding boxes* pour chaque entité-cible.

Supposons tout d'abord un problème mono-classe (il n'y a donc pas de variable c_i dans le vecteur de prédiction). L'objectif est de supprimer les *bounding boxes*-doublons qui localisent une même entité. Il est donc probable qu'il y est des chevauchements de surface entre chacune de ces *bounding boxes*. L'idée est donc de supprimer les *bounding boxes* qui partagent une trop grande surface soit définir une valeur-seuil pour la valeur de l'IoU. Ainsi, cette partie se découpe en deux phases. Tout d'abord, on choisit la *bounding boxes* avec la confiance la plus importante puis on supprime toutes *bounding boxes* dont l'IoU avec la *bounding boxes* ciblée est supérieure à la valeur-seuil choisie (disons 0.5). Cette étape est ainsi répétée itérativement jusqu'à ce que toutes les *bounding boxes* soit traitées (suppression ou préservation).

Dans le cas d'un problème multi-classe, le cycle est séparé selon la classe prédite par la *bounding boxes*. Ainsi, si un IoU entre deux *bounding boxes* est supérieur à la valeur-seuil (0.5) mais que la classe prédite diffère, alors il n'y aura pas de suppression de *bounding boxes*. On réalise donc le cycle en ne considérant que la classe 1, puis on recommence avec la classe 2 etc... Cette spécificité permet de mieux traiter le cas d'entités de nature différente mais proche spatialement¹⁸⁵. Un exemple concret est visible sur la Figure 99

Néanmoins, une faiblesse majeure existe. En effet, une hypothèse est faite que deux *bounding boxes* qui se chevauchent et de même classe identifient la même entité. Bien que vrai dans de nombreux cas, il y a d'autres cas où c'est problématique, notamment les nuées denses telles qu'une foule d'individus, une nuée d'oiseaux ou d'abeilles par exemple.

12.2.10 Feature Pyramid

Feature Pyramid[68] (FPN) propose une nouvelle méthode d'extraction d'attribut d'une image. En effet, un dilemme existe entre résolution de la *feature map* et l'information sémantique produite. Plus les couches de convolution se succèdent, plus l'information sémantique se développe au détriment de la résolution qui diminue¹⁸⁶. Ceci est problématique car ça favorise les entités de grande

¹⁸⁵Par exemple, un cycliste où l'on localise le vélo et l'individu.

¹⁸⁶Pas de Padding et application de Downsampling récurrente



Figure 98: Illustration de la problématique de la superposition de bounding boxes

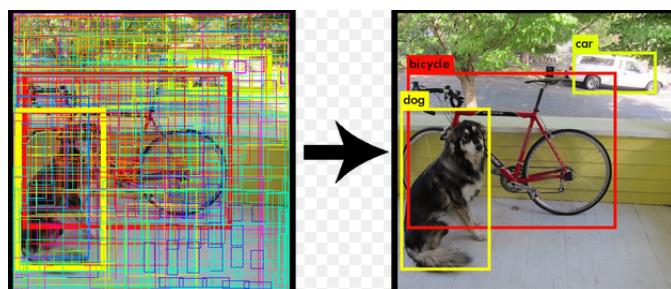


Figure 99: Illustration de Non-Max Detection

taille sur l'image au détriment des plus petites qui seraient "noyées" par la perte de résolution. Il est donc nécessaire d'apporter une solution afin de conserver l'information portée par les *feature map* de grande résolution.

Feature Pyramid Networks propose une approche pyramidale. Elle se décompose en deux étapes: l'étape *Bottom-up* et l'étape *Top-down*.

- **Bottom-up:** Cette étape est comparable à une succession de couches convolutives standards dont le *stride* augmente de manière à diviser par deux la dimension des *feature map* en sortie. La valeur sémantique des *feature map* augmente au fil des étages de la pyramide mais sa résolution diminue du fait de l'impact des *stride*. Chaque étage peut produire une ou plusieurs *feature map*.
- **Top-down:** Cette étape réalise un *Upsampling* sur les *feature map*. La résolution de l'image est grossière¹⁸⁷ mais l'information sémantique est, quant à elle, forte. Le premier étage de la pyramide (première couche de convolution) n'est pas considéré durant cette étape car trop gourmande en mémoire (et temps de traitement). Chaque étage issu de *Bottom-up* ou *Top-down* traitent des données de même dimension deux à deux.
- **Connexions latérales:** L'étape *Top-down* souffre du *Upsampling* qui ne peut retranscrire la résolution de l'image avec fidélité. Pour corriger ce défaut, des connexions latérales sont produites. Une connexion relie deux étages respectivement, i.e l'étage 3 de *Bottom-up* est lié à l'étage 3 de *Top-down*. Une liaison extrait les *feature map* de l'étage correspondant, réalise une convolution 1×1 afin de contrôler la profondeur et additionne la sortie obtenue avec les *feature map* correspondant sur le cycle *Top-down*.
- **Sortie prédictive:** Les *feature map* issues de l'étape *Top-down* sont soumises à une autre couches de convolution (filtre 3×3) avant d'être exploité par les couches de détection et classification.

Une illustration de cette architecture est visible sur la Figure 100. *Feature Pyramid* n'est pas un réseau de détection d'entité mais un extracteur d'attributs exploitable par d'autres architectures, notamment par le modèle RetinaNet.

12.2.10.1 Amélioration de l'architecture du réseau central

12.2.10.2 A faire

[65] <https://arxiv.org/pdf/1804.06215.pdf>

12.3 Object recognition: méthodes State-of-the-art

Attention: Cet aperçu de l'état de l'art n'est qu'indicatif et propose quelques approches présentant une efficacité reconnue ou une originalité novatrice (et

¹⁸⁷Le *Upsampling* ne peut garantir une mise à l'échelle précise

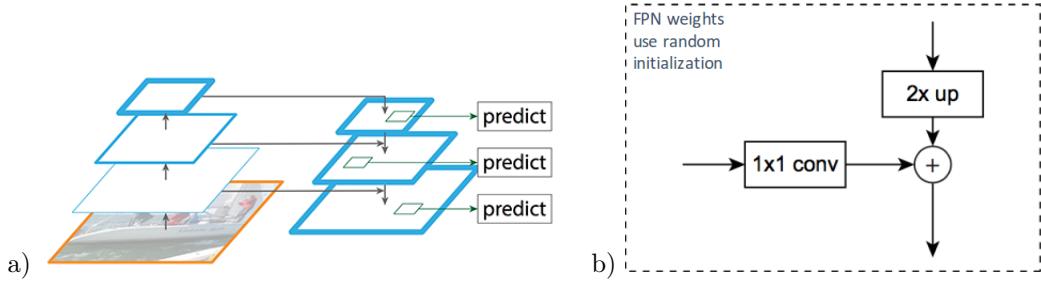


Figure 100: Illustration de l'architecture Feature Pyramid: a) Architecture générale b) Architecture des connexions latérales

intéressante à étudier). Il est certain qu'il doit exister d'autres méthodes aux performances au moins similaires.

Il existe deux grandes familles d'algorithmes d'*Object Detection*: *Region based object detectors* et *single shot object detectors*. Ces deux familles ont des algorithmes performants à l'état de l'art bien que leurs approches diffèrent légèrement.

En effet, *Region based object* est caractérisé par le fait qu'un même calcul sur une image puisse être réalisé plusieurs fois au sein d'une prédiction. Par exemple, une même *feature map* peut être exploitée (en partie) plusieurs fois au sein du réseau (lors de la création de la *bounding boxe* et de la classification de l'entité isolée par exemple). Cette tolérance à la redondance de calculs permet de favoriser une précision élevée du réseau mais porte préjudice à sa vitesse. De plus, cette famille d'algorithme est caractérisée par une approche en *bloc*. C'est-à-dire qu'elle accepte l'exploitation d'algorithmes extérieurs et ou de structures parallèles au sein d'un même réseau (par exemple, deux réseaux de neurones). La méthode finale est donc une combinaison de méthodes distinctes. Cette tolérance rend plus difficile l'apprentissage du réseau du fait de la nature différente de ses composants.

Single shot object, au contraire, cherche la simplicité et l'unicité. Elle n'exploite une donnée qu'une fois uniquement, ce qui permet une vitesse bien plus importante grâce à la non-redondance de calculs. L'objectif est donc d'unir le réseau de classification et de *Region Proposal* au sein du même réseau. Néanmoins, cette approche tend à rendre le modèle moins robuste bien que les modèles récents s'approchent grandement des performances des *Region based object*. Cette approche tend à donner des réseaux plus *simples* et modulables et semble gagner la bataille en terme de popularité grâce à cette simplicité qui laisse penser qu'elle possède une plus grande marge de progression.

Les méthodes *Region based object* sont principalement *R-CNN*, *Fast R-CNN*,

Faster R-CNN et *R-FCN*. Les approches *Single shot object* sont *YOLO* et *SSD*.

12.3.1 R-CNN - Algorithme précurseur

R-CNN[25] est un algorithme d'*Object recognition* développé en 2014. Son fonctionnement repose sur des méthodes décrites dans la section précédente.

Le réseau se partage en 3 parties indépendantes:

- Création de *bounding boxes*: Cette étape est réalisée par une méthode de *Regional Proposal*. *Selective Search* a été choisi par les créateurs de R-CNN. Il y a une création d'environ 2000 *bounding boxes*.
- Extraction d'attributs: Les sous-images discriminées par les *bounding boxes*¹⁸⁸ sont analysées par un réseau convolutif déjà pré-entraîné. L'entrée de ce type de réseau est de taille fixe. Il est donc nécessaire de s'assurer que chaque image possède la même dimension. Pour redimensionner les images, une approche par *Warping* ou *Cropping* est utilisée.
- Prédiction et affinement: Cette étape réalise 2 actions: Prédire la classe de l'entité présente dans la *bounding boxes* observée¹⁸⁹ et un affinement de la dimension de la *bounding boxes*.

La classification est réalisée par un modèle indépendant. Dans l'architecture originale, un SVM (*Support Vector Machine*¹⁹⁰) est utilisé. Le raffinement de la *bounding boxes* est réalisée par un réseau Full-Connected qui est chargé de prédire une nouvelle dimension pour la *bounding boxes*. Ce réseau réalise donc une "correction d'erreur" de la dimension par défaut obtenue par l'algorithme de *Region Proposal*. Un exemple illustratif de ce réseau est visible sur la Figure 101.

Un graphique récapitulatif de ce réseau est visible sur la Figure 102. Ce réseau possède des résultats remarquables en comparaison des autres méthodes du moment. Néanmoins, du fait des composants indépendants de son architecture, son apprentissage n'est pas aisé et ses prédictions longues à réaliser. Par conséquent, il ne peut pas être exploité dans le cadre du temps-réel.

12.3.2 Spatial Pyramid Pooling - Algorithme précurseur

R-CNN souffre de sa lenteur de prédiction. Cette lenteur est essentiellement due au réseau convolutif appliqué sur chaque *bounding boxes*. En effet, étant donné le recouvrement de nombreuses *bounding boxes*, il y a une redondance de calcul réalisé lors des créations de *feature map*¹⁹¹. La proposition de *Spatial Pyramid Pooling Network*[33](SPPnet) est de réaliser l'extraction des *feature map* **avant**

¹⁸⁸Elles représentent une partie de l'image brute initiale

¹⁸⁹Dans la majorité des cas, aucune classe n'est représentée car il n'y a pas d'entité présente!

¹⁹⁰C'est un algorithme de Machine Learning utilisé pour des problèmes de Classification

¹⁹¹Le filtre de convolution va analyser les mêmes pixels plusieurs fois d'où la redondance

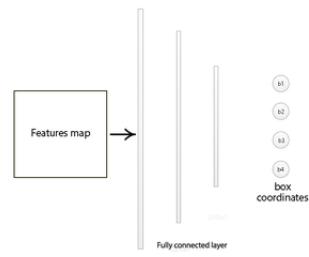


Figure 101: Illustration d'un réseau de prédiction de bounding boxes selon R-CNN

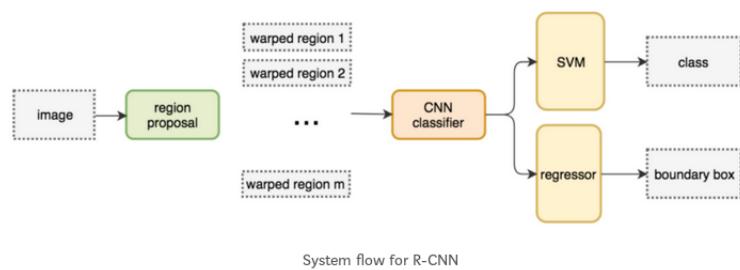


Figure 102: Illustration d'un réseau R-CNN

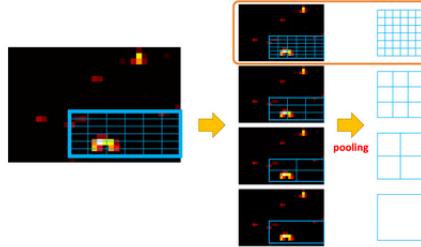


Figure 103: Illustration du Spatial Pyramid Pooling

l’application de l’algorithme de *Region Proposal*. Ainsi, l’algorithme de *Region Proposal* va s’appliquer sur la *feature map* de l’image et non l’image originale, permettant ainsi de réaliser l’extraction d’attributs une fois uniquement.

La problématique d’uniformisation des dimensions reste présente. Afin d’uniformiser les *feature map* issues de l’algorithme de *Region Proposal*, SPPnet introduit *Spatial Pyramid Pooling*. Cette méthode exploite une architecture pyramidale basée sur une action de Pooling où chaque étage de la pyramide réalise un Pooling de dimensions différentes sur la *feature map* initiale. Cette méthode permet ainsi d’avoir une sortie de dimension $N * K_i$ avec N , nombre d’étages de la pyramide et K_i , nombre de valeurs obtenues par le Pooling. Cette valeur varie pour chaque étage, la dimension de sortie n’est donc pas uniforme selon l’axe de N . Un exemple est visible sur la Figure 103.

L’architecture de prédiction du réseau est similaire à l’architecure R-CNN. L’apport principal de ce réseau est le gain de temps important par rapport à R-CNN pour une qualité de prédiction similaire. Le réseau convolutif doit être appris en amont. L’architecture de SPPnet ne permet pas son apprentissage. La Figure 104 résume l’architure de SPPnet.

12.3.3 Fast R-CNN - Algorithme précurseur

Fast R-CNN[24], comme son nom l’indique, est une optimisation de R-CNN pour minimiser son temps d’exécution et faciliter son apprentissage. Ces deux améliorations reposent sur 2 spécificités.

Comme SPPnet, Fast R-CNN réalise l’extraction d’attributs par réseau convolutif avant l’exploitation d’un algorithme de *Region Proposal*. L’uniformisation des dimensions est réalisée par *RoI Pooling* (voir Section précédente). Cette uniformisation est comparable à un cas particulier de *Spatial Pyramid Pooling* où la pyramide possède qu’un étage uniquement (voir Figure 105).

La plus-value véritable comparée à SPPnet est le remplacement du SVM par

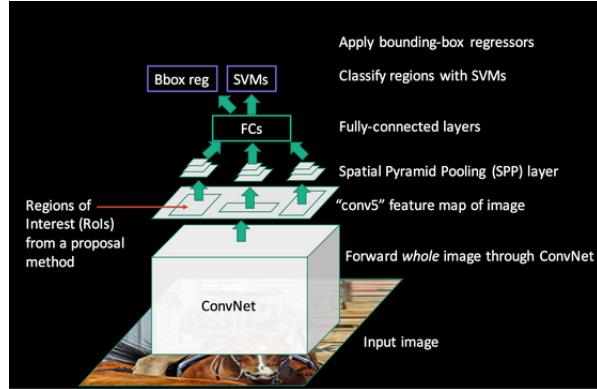


Figure 104: Illustration du Spatial Pyramid Pooling Network

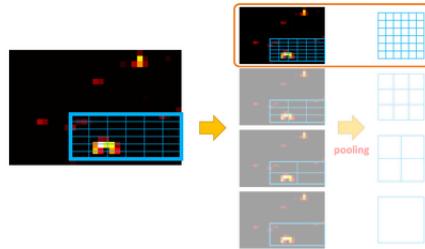


Figure 105: Analogie de RoI Pooling avec Spatial Pyramid Pooling

un réseau Full-Connected qui réalisera la classification. Cette modification permet ainsi de réaliser une mise à jour intégrale du réseau (CNN+FC) par rétro-propagation du fait de l'architecture intégralement neuronale. Sans cette modification, le réseau convolutif doit être appris en amont indépendamment du réseau. Seules les couches FC pourraient être entraînées par le biais de la prédiction du correctif des *bounding boxes*. Ce système peut donc exploiter une méthode d'apprentissage unitaire et faciliter son exploitation en unifiant les entités du réseau. L'architecture du réseau et sa structure d'apprentissage sont visibles sur la Figure 106.

12.3.4 Faster R-CNN

Faster R-CNN[91] est une amélioration de Fast R-CNN. L'idée derrière cette architecture est de s'émanciper de l'algorithme de *Region Proposal* qui était indépendant du réseau jusque-là (utilisation de l'algorithme *Sliding Search*). Cette dépendance est très problématique car il est le facteur limitant à la vitesse du système. Faster R-CNN propose donc une approche qui intègre dans son réseau-même, une méthode de *Regional Proposal* nommée *Region proposal network*.

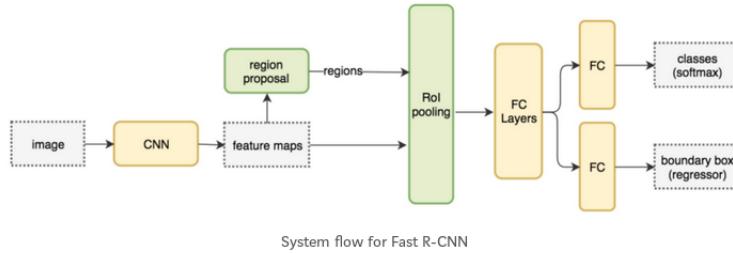


Figure 106: Illustration du Fast R-CNN

work(RPN). Cette architecture repose sur l'approche par *Anchor Boxes* (voir Section précédente) et est intégralement convolutif. La sortie de RPN correspond à la confiance en la présence d'une entité ou non sur l'*anchor* et des coordonnées corrigées de l'*anchors* utilisée. Si il y a 9 *anchors*, il y aura $K*9*(N+M)$ valeurs de sortie où K, nombre de case du quadrillage, N nombre de variable pour représenter la confiance¹⁹² et M, caractéristiques de la fenêtre¹⁹³.

La partie prédictive du réseau est similaire à Fast R-CNN (RoI Pooling + classification par réseau Full-Connected). Une autre plus-value du système est de créer une méthode de *Region Proposal* propre à un jeu de données. Du fait que RPN apprend sur le jeu de données d'apprentissage de tout le réseau et non indépendamment, il sera parfaitement adapté et spécialisé à la nature des données observées. Il est intéressant de noter qu'il y a un double ajustement de la dimension des *bounding boxes*: lors de l'ajustement de l'*anchor* par le RPN et lors de la classification après l'uniformisation des dimensions par RoI.

Une illustration de ce réseau est visible sur la Figure 107. Il est un modèle avec l'une des meilleures précisions de l'état de l'art. Il reste néanmoins très lourd et lent, ce qui l'empêche d'être exploitable en temps-réel. Néanmoins, Faster R-CNN a été une amélioration remarquable de l'approche proposée par R-CNN. Un comparatif de performance est visible sur la Figure 108. D'autres approches sont préférées aujourd'hui car bien plus rapide et avec une qualité de résultat satisfaisante (telles que YOLO ou SSD par exemple). De plus, une faiblesse de ce type d'approche est la séparation des tâches. La séparation entre le réseau qui produit les *bounding boxes* et le réseau prédictif favorise un coût de calculs important lié à la redondance de calculs entre la phase de création des fenêtres et de la classification (les deux réseaux observent la même feature map) d'où les limites de vitesse de ce type de réseau.

¹⁹²En général, il n'y a qu'une valeur en sortie

¹⁹³En général, il y a 4 valeurs en sortie

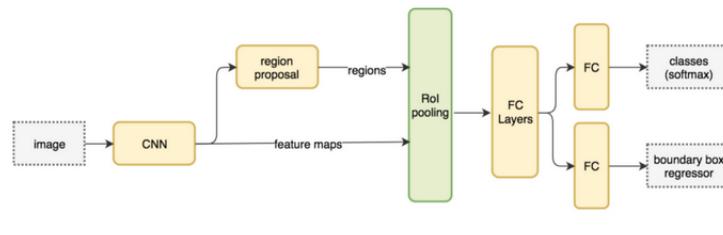


Figure 107: Illustration du Faster R-CNN

	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image (with proposals)	50 seconds	2 seconds	0.2 seconds
(Speedup)	1x	25x	250x
mAP (VOC 2007)	66.0	66.9	66.9

Figure 108: Comparatif de performance entre R-CNN, Fast R-CNN et Faster R-CNN

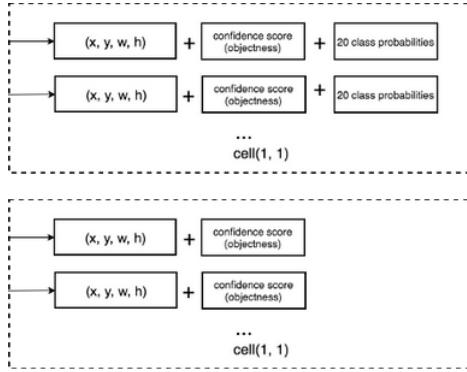


Figure 109: Différence de la sortie des couches convolutives entre les approches Single shot et Region based

12.3.5 Différence principale entre Single shot et Region based

Faster R-CNN, du fait de son architecture entièrement basée sur des réseaux de neurones, se rapprochent de l'architecture *Single shot*. Néanmoins, une différence majeure est encore présente et repose sur son approche de l'algorithme *Anchors Box*.

Supposons la création de 9 *anchors*, La sortie du RPN associée à Faster R-CNN sera de la forme $K*9*(N+M)^{194}$. N (souvent égal à 1) et M (souvent égal à 4) sont associés à la confiance en la présence d'une entité et à la dimension de la *bounding box*. La classification de l'entité est donc complètement ignorée, la sortie se limitant à dire sa confiance en la présence d'une entité sans discrimination. Cette sortie est donc envoyée à un autre "sous-réseau" pour la classification.

Au contraire, dans le cas d'un algorithme *Single shot*, la sortie serait de la forme $K*9*(N+M+C)$ où C, probabilités de classification pour une classe donnée. Cette sortie est donc associée à la fin de la prédiction et donc du réseau prédictif. Il n'y a pas de transfert à un autre sous-réseau de classification. Supposons une *feature map* de dimension $10*10$ où 20 classes sont à discriminer avec l'aide de 9 *anchors*. On obtiendra donc pour Faster R-CNN, une sortie de dimension $10*10*9*(1+4)$ et pour un algorithme *Single shot*, $10*10*9*(1+4+20)$. Une illustration est visible sur la Figure 109.

Cette différence sur l'aspect prédictif implique une différence majeure: les modèles *Single shot* ont une valeur finie (et identique) de *bounding boxes* définies pour chaque image (en accord avec le quadrillage de l'image¹⁹⁵) alors que les approches *Region based*, de part la séparation des sous-réseaux, permettent la

¹⁹⁴Voir section 12.3.4 pour plus de détails

¹⁹⁵Cette particularité est expliquée dans les sections qui vont suivre

création d'un nombre indéfini de *bounding boxes*. Cette spécificité des modèles *Region based* permet généralement d'obtenir un meilleur taux de détection car le *Rappel* est plus important. Néanmoins, ces modèles sont plus lourds, plus lents et plus sensibles aux problèmes de distributions fortement déséquilibrées entre les classes (notamment entre les classes associées à une entité et la classe générale qui traduit l'absence d'une entité).

12.3.6 You Only Look Once (YOLO)

Les méthodes précédentes séparent le réseau de *Region Proposal* du classifier. Une autre approche proposée par YOLO repose sur l'idée de pouvoir obtenir les *boundary boxes* et les classes directement à partir des *feature map* en un réseau unique. YOLO a été très populaire lors de sa sortie grâce à sa structure *simple* et souple d'utilisation. Aujourd'hui, deux versions ont été réalisées afin d'améliorer les performances de la version initiale. De plus, des versions dites *tiny* du réseau ont été créées pour diminuer l'impact mémoire du réseau et ses exigences matérielles pour réaliser les prédictions. Il s'agit donc de versions idéales pour des systèmes portables et embarqués.

12.3.6.1 YOLOv1

La première version de YOLO[88] repose sur un réseau convolutif finalisé par deux couches Full-Connected. Le réseau va reproduire le comportement suivant:

- L'image d'entrée est divisé selon un quadrillage de dimension $S \times S$. Si le *centre* d'une entité de l'image (défini par le centre d'une *bounding boxes*) est présent dans une des cases du quadrillage, alors cette case est responsable de déterminer la classe de l'objet.
- Chaque case prédit K *bounding boxes*, i.e ses coordonnées spatiales (centre(x,y), hauteur et largeur) et son degrés de confiance en la présence d'un objet au sein de cette *bounding boxes*. Le degrés de confiance n'exprime que la probabilité qu'une entité quelconque soit présente. Il n'y a pas de discrimination de la nature de l'entité à ce niveau. Cette valeur, dans le cas idéal, devrait être exprimé selon:

Si aucun objet est présent:

$$conf(B_i) = 0$$

Si un objet est présent (même partiellement):

$$conf(B_i) = Pr(entite) * IoU_{Pred, True}$$

Avec $Pr(entite)$, probabilité qu'une entité soit présente et $IoU_{Pred, True}$, IoU entre la *bounding box* créée par le réseau et la vraie à obtenir. Ainsi, cette valeur permet d'évaluer la présence ou non d'un objet et la surface de l'objet isolé. En effet, une entité peut être détectée mais que partiellement ou au contraire, la surface peut être trop grande.

- Chaque case calcule C probabilités de classe $Pr(Classe_i|entite)$. Ainsi, toute *bounding boxes* dont le centre est délimité par cette case sera considérée comme représentative de la classe dont la probabilité est la plus élevée et ce, **indépendamment** du nombre de *bounding boxes*. Supposons 5 *bounding boxes* dont le centre est dans la case C_i représentant la classe i alors ces 5 *bounding boxes* seront classées comme de la classe i.
- La sortie du réseau est donc de la forme: $S * S * (B * 5 + C)$ avec S*S, nombre de cases du quadrillage, B, nombre de *bounding boxes* et C, nombre de classes discriminées. Dans la version originale, le quadrillage est de 7*7 et chaque case est associée à 2 *bounding boxes* (B=2). 20 classes sont discriminées. La sortie du réseau sera donc de la forme 7*7*(2*5+20).
- Les *bounding boxes* sont conservées si la valeur de confiance est supérieure à un seuil. Les *bounding boxes* maintenues sont ensuite filtrées par *Non-Max Suppression*. Une illustration récapitulative est visible sur la Figure 110.

Important: Les valeurs prédites du centre de la *bounding boxes* sont normalisées. Cette normalisation est **importante** et nécessaire au bon fonctionnement de YOLO. Le centre de la *bounding boxes* n'est pas prédite de manière *absolue* mais relative à la case à qui elle est associée. Ainsi, les valeurs (x,y) sont comprises entre 0 et 1 relatif à un point de référence défini sur le sommet supérieur gauche par convention. Par exemple, supposons la case (5,5) et supposons un centre idéalement positionné à (5.4, 5.1). La prédiction du réseau ne sera pas (5.4, 5.1) mais (0.4, 0.1). Cette spécificité est capitale pour conserver l'intégrité de l'approche YOLO. En effet, si on réalise une prédiction absolue, il est possible que le centre "quitte" la case associée (par exemple (5.5,6.1)). Il y a donc un conflit entre la nouvelle position du centre et la case responsable de la prédiction. Afin de réaliser cette normalisation, les prédictions suivent la relation suivante:

$$\begin{aligned}x &= \sigma(b_x) + c_x \\y &= \sigma(b_y) + c_y\end{aligned}$$

Avec c_x, c_y position du sommet supérieur gauche de la case considérée, σ , fonction sigmoïde¹⁹⁶, (x,y) position réelle du centre et (b_x, b_y) , position relative du centre (valeurs prédites).

Le réseau de YOLO est un réseau convolutif inspiré d'une architecture de réseau *Inception* finalisée par deux couches Full-Connected (24+2 couches dans la version originale). Un récapitulatif du réseau est présent sur la Figure 112 mais pour une présentation détaillée du réseau, veuillez vous référer à l'article de recherche[88]. Nous n'approfondirons pas le réseau convolutif qui ne présente pas de particularité notable mais il est intéressant d'étudier les couches Full-Connected du réseau. Comme on peut le voir, la première couche possède 4096

¹⁹⁶Elle donne une valeur entre 0 et 1

neurones. Or, la sortie finale doit être de dimension $7*7*30$, i.e 1470 valeurs¹⁹⁷. Il est donc nécessaire de redimensionner la couche. Pour cela, la seconde couche de Full-Connected doit être de dimension 1470 et un redimensionnement de la sortie de cette couche devra être fait afin d'obtenir l'architecture $7*7*30$. Une illustration des couches Full-Connected est visible sur la Figure 111

Une des particularités de YOLOv1 est sa fonction de perte. Elle se présente sous la forme d'une combinaison linéaire de *Squared error* basée sur différentes données. Cette fonction considère la sortie du réseau prédictif. De ce fait, l'intégralité des *bounding boxes* seront considérées¹⁹⁸. Plusieurs spécificités sont à considérer:

- Afin d'augmenter la précision des *bounding boxes*, il est utile de pondérer l'erreur associée à la prédictions de la *bounding boxes* par rapport à la classification de l'entité associée. On va donc créer un coefficient λ_{coord} (supposons $\lambda_{coord}=5$) qui va pondérer l'erreur de dimension par rapport à l'erreur de classification laissée à 1.
- Du fait du nombre important de *bounding boxes* qui ne contiennent pas d'objet, il est important de minorer leurs impacts sur la fonction de coût au risque de voir l'importance des prédictions de *bounding boxes* contenant un objet fortement minorée. En effet, Si l'essentiel des *bounding boxes* sans objet sont bien prédites (c'est souvent plus facile de à réaliser car la discrimination est plus faible pour les *bounding boxes* sans objets) alors une erreur sur une *bounding boxes* contenant un objet sera moins préjudiciable sur le résultat de la fonction de coût car "compensée", ce qui est très problématique. Pour cela, un coefficient noté λ_{noobj} est crée et minorera l'erreur correspondant aux *bounding boxes* sans objet. Supposons λ_{noobj} égal à 0.5.
- Etant donné qu'il n'y a pas de filtrage de *bounding boxes* pour le calcul de l'erreur, il est nécessaire d'isoler la *bounding boxes* qui servira de "référence" pour la prédiction. Afin de la déterminer, on gardera, pour chaque objet identifiée sur l'image d'apprentissage, la *bounding box* possédant le meilleur IoU avec la fenêtre de référence définie sur l'image d'apprentissage. Ainsi, pour 3 entités présentes sur une image, seulement 3 *bounding boxes* seront considérées comme responsable de la prédiction et de ce fait, délimiteur d'un objet. Toutes les autres seront considérées comme négatives. Il y a, bien sûr, un risque important de faux positifs et faux négatifs durant les premières itérations d'apprentissage du réseau.
- La classification d'un objet est pertinent si une *bounding box* isole un objet. Si une *bounding box* n'est pas "responsable" de l'isolation d'un objet, la classe prédite par cette *bounding box* n'a pas d'influence sur la fonction

¹⁹⁷En effet, la notion de quadrillage est représentée par la dimension de la sortie

¹⁹⁸L'étape de sélection des *bounding boxes* et l'application de Non-Max Suppression ne sont pas considérées

de perte. Il en va de même pour la dimension de la fenêtre définie. Ainsi, pour une *bounding box* non responsable de la détection d'une image, seule la valeur de confiance en la présence d'un objet est considérée alors que pour une *bounding box* qui est responsable de la détection d'un objet, la dimension de la fenêtre et la classification sont considérées.

- *Square Error* est convexe et présente donc une caractéristique intéressante dans le cadre d'une optimisation. Néanmoins, elle favorise la correction des grandes erreurs et non des petites du fait de la puissance. Cet aspect est problématique car elle favorisera la correction des erreurs associées à de grandes *bounding boxes* et tolérera plus facilement les erreurs sur des petites car l'échelle de la taille des erreurs sera "différente". Cet aspect est partiellement corrigé en considérant la racine carré de la valeur à comparer (hauteur et largeur) et non la valeur réelle. Cette imperfection participe à la faiblesse de YOLOv1 à la détection d'entité de petite dimension.

En considérant ces caractéristiques, la fonction de coût est donc de la forme¹⁹⁹:

- *Bounding boxes responsables* de la détection d'un objet:

– **Dimension:** Centre et Hauteur/Largeur:

Pour $\mathbb{1}_{ij}^{obj} = 1$ si la *bounding box* est **responsable** de la détection d'un objet (0 sinon):

$$\begin{aligned}\mathcal{L}_1 &= \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\ \mathcal{L}_2 &= \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]\end{aligned}$$

– **Classification:** Confiance + probabilités de classe:

$$\begin{aligned}\mathcal{L}_3 &= \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 \\ \mathcal{L}_4 &= \sum_{i=0}^{S^2} \mathbb{1}_{ij}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2\end{aligned}$$

- *Bounding boxes non responsables* de la détection d'un objet:

– **Classification:** Confiance:

Pour $\mathbb{1}_{ij}^{noobj} = 1$ si la *bounding box* n'est **responsable** de la détection d'un objet (0 sinon):

$$\mathcal{L}_5 = \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2$$

¹⁹⁹La forme peut impressionner mais elle est facilement compréhensible. Seules les notations sont lourdes !

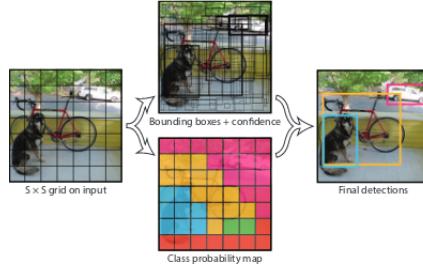


Figure 110: Fonctionnement général de l'algorithme YOLOv1

- **Fonction de perte totale:**

$$\mathcal{L}_{total} = \mathcal{L}_1 + \mathcal{L}_2 + \mathcal{L}_3 + \mathcal{L}_4 + \mathcal{L}_5$$

Cette approche présente trois défauts significatifs. Une *bounding boxes* dont le centre est dans une case sera associée à la classe prédictive par cette case. Cette contrainte spatiale forte est très préjudiciable en cas d'objets de différentes natures proches. En effet, si deux objets distincts sont proches et donc, que le centre des *bounding boxes* associées sont au sein d'une même case, la prédiction de la classe ne sera pas capable de prédire deux classes distinctes, ce qui produira une erreur de prédiction. De même, chaque case définit B *bounding boxes*. De ce fait, seulement B *bounding boxes* peuvent avoir un centre dans une case, ce qui provoque une limite spatiale au nombre d'entités détectables (S^*S^*B). Ce modèle est donc peu robuste pour détecter les entités (de même nature) proches et en masse telles que les nuées d'oiseaux ou une foule d'individus par exemple. Pour finir, cet algorithme est très sensible aux dimensions de *bounding boxes exotic*²⁰⁰. Il est très dépendant des *bounding boxes* du jeu d'apprentissage, ce qui peut être dangereux avec un jeu de données faiblement représentatif. Ses capacités de discrimination, du fait de l'architecture du réseau choisi (beaucoup de *downsampling*), tend à être grossières. Un risque de faible généralisation est donc probable, de même que des approximations de calculs de dimensions des *bounding boxes*. Une dépendance au jeu de données d'apprentissage semble donc très (trop) importante malgré une efficacité très correcte de l'algorithme.

12.3.6.2 YOLOv2 - YOLO9000

Afin d'améliorer le modèle initial de YOLO, une version 2 a été créée et appelée (sobrement) YOLO9000[89]. Cette amélioration rend le modèle plus rapide, plus précis et généralise la détection à 9000 classes au lieu de 20.

²⁰⁰YOLO possède de grandes difficultés à détecter des entités de petites tailles

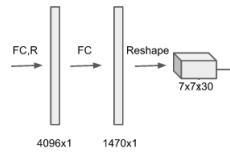


Figure 111: Architecture des couches Full-Connected de l'algorithme YOLOv1

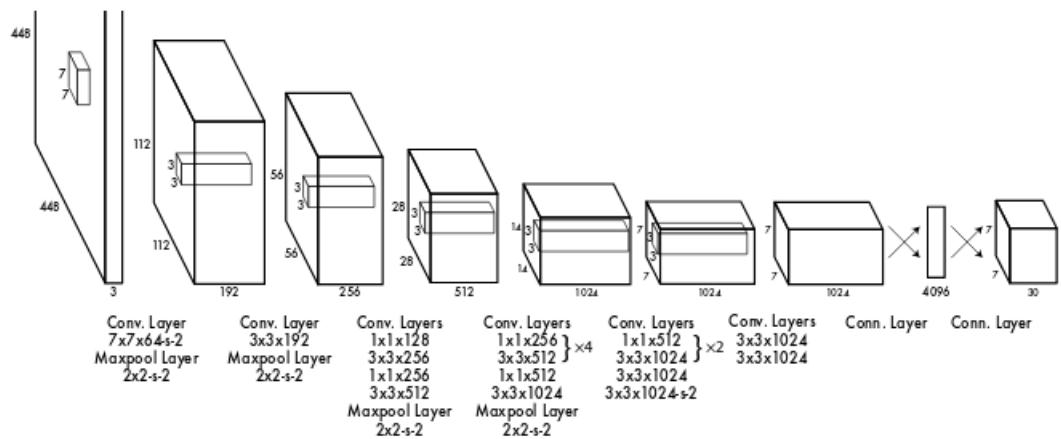


Figure 112: Architecture de l'algorithme YOLOv1

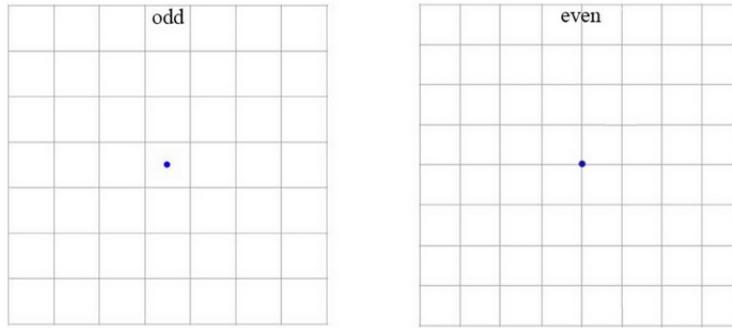


Figure 113: Différence de centre entre un quadrillage pair et impair

YOLOv1 réalise un nombre significatif d'erreur de localisation durant ses prédictions. De même, son *Rappel* (Recall) est trop faible comparé aux méthodes de *Region Proposal* classiques²⁰¹. Afin d'améliorer le modèle, il a donc été préféré d'améliorer la localisation et la détection d'entité au détriment de la classification.

Afin d'améliorer ces caractéristiques, différents ajouts ont été faits. YOLO9000 utilise *Batch Normalization* après chaque couche de convolution. Afin d'être plus robuste dans la détection, un pré-apprentissage²⁰² à de multiples résolutions est réalisé (224*224 puis 448*448). Ceci permet d'améliorer le modèle sur la gestion d'images à "haute" résolution. En effet, YOLOv1 réalisait un apprentissage sur des images 224*224 avant de se mettre à l'échelle soit 448*448 mais cette approche ne permettait pas une évolution efficace des extracteurs d'informations des couches convolutives. Un double apprentissage avec YOLO9000 compense cette faiblesse. De plus, la dimension du quadrillage passe de 7*7 à 13*13, ce qui permet une bien meilleure discrimination des entités, notamment groupées. Une approche un peu *tricky* a été faite. En effet, 13 étant impair, le quadrillage possède une case centrale. En générale, une image a tendance à posséder une entité en son centre (ou proche). Il est donc intéressant de pouvoir la détecter sans courir le risque d'un décalage associé à un problème de centre d'*anchor* légèrement décalé. une illustration présente le problème sur la Figure 113.

Pour compenser le problème de faible *Rappel*, YOLO9000 va exploiter l'approche par *Anchor boxes* proposée par Faster R-CNN. Cette approche est très bénéfique car elle permet une classification d'entités variées au sein d'une même case du quadrillage. En effet, chaque *anchor* est indépendante et peut prédire une classe distincte. Ainsi, si 9 *anchors* sont présentes sur une même case, alors 9 entités peuvent avoir leur centre localisé sur une même case et être classées

²⁰¹Notamment utilisées par Faster R-CNN

²⁰²Il s'agit de l'entraînement des couches convolutives d'extraction d'informations avant de faire l'apprentissage spécifique de détection

selon des classes différentes. Ainsi, il n'y a plus de prédictions "directes" des dimensions de la *bounding box* mais des prédictions d'*offset* d'une dimension pré-établie. Ce travail est plus facile à apprendre pour le réseau de neurones mais diminue la précision du modèle. La diminution est négligeable alors que le *Rappel* augmente significativement. Dans le cadre de YOLOv2, il y a 5 *anchors* par case.

Tout comme pour YOLOv1, le centre est relatif à la case associée. Par contre, la dimension de la *bounding box* sera relative à la dimension de l'*anchor*. On obtient donc:

$$\begin{aligned}x &= \sigma(b_x) + c_x \\y &= \sigma(b_y) + c_y \\h &= p_h e^{t_h} \\l &= p_l e^{t_l} \\Pr(\text{entité}) * IoU(\text{box}, \text{entité}) &= \sigma(t_0)\end{aligned}$$

Avec c_x , c_y position du sommet supérieur gauche de la case considérée, σ , fonction sigmoïde²⁰³, (x,y) position réelle du centre et (b_x, b_y) , position relative du centre (valeurs prédictées). p_h et p_l correspondent respectivement à la hauteur et à la largeur de l'*anchor*. t_h et t_l sont les prédictions associées aux offset des dimensions de l'*anchor*. t_0 est la valeur prédictée de la confiance d'une *anchor* en la présence d'une entité. Un exemple est visible sur la Figure 114.

Les *anchors* sont, à la base, prédéfinies manuellement. Il peut être difficile de définir des dimensions capables de discriminer l'essentiel des entités qu'on cherche à détecter. Bien que des dimensions préférées aient déjà fait leurs preuves dans divers contextes, YOLOv2 fait le choix de définir des dimensions en adéquation avec le jeu de données d'apprentissage. Pour cela, il va utiliser l'algorithme *K-means*[51]. Pour éviter la problématique de tolérance associée aux faibles dimensions, la métrique euclidienne n'est pas exploitable. Une autre a donc été créée pour corriger ce défaut et repose sur IoU. Elle est définie par:

$$d(\text{box}, \text{centroid}) = 1 - IoU(\text{box}, \text{centroid})$$

Afin de considérer la possibilité de présence d'entités de tailles différentes, YOLOv2 va *additionner* des *feature map* à différents niveaux du réseau afin de considérer différentes échelles de dimensions. Ainsi, un comportement similaire à l'addition à la fin d'un block *ResNet* est réalisée. Un exemple illustratif de cette approche est visible sur la Figure 115. Il est **important** de constater que les couches Full-Connected en fin de réseau ont été remplacées par des couches convolutives, ce qui favorise la vitesse du système.

Par ailleurs, YOLO9000 propose une nouvelle approche afin d'unir différents jeux de données. Dans le cadre de YOLO9000, il s'agit d'ImageNet et COCO.

²⁰³Elle donne une valeur entre 0 et 1

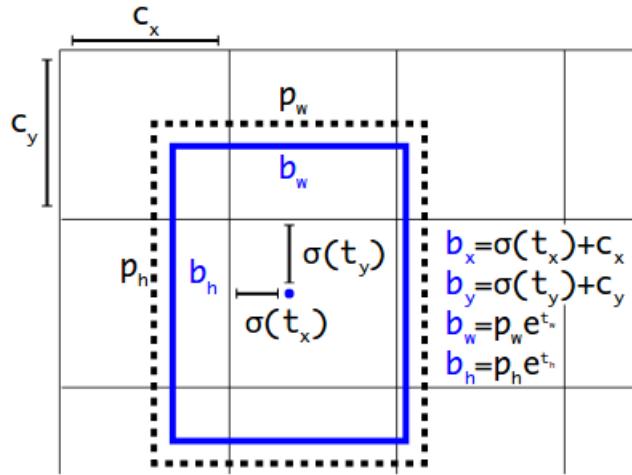


Figure 114: Conversion des prédictions du réseau YOLOv2 selon l'anchor associée

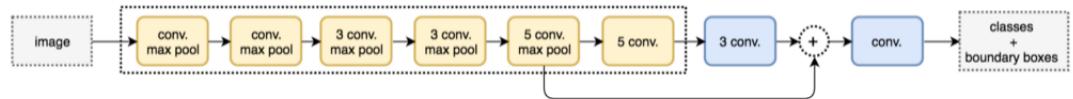


Figure 115: Architecture du réseau convolutif YOLOv2

Nous ne détaillerons pas cette méthode qui sort du cadre de ce cours. Pour plus d'informations sur cette méthode, référez-vous à l'article associé [89]. YOLOv2 et YOLO9000 sont strictement identiques si ce n'est la différence associée à la méthode d'union des deux jeux de données et de l'apprentissage associé qui s'en voit légèrement modifié.

12.3.6.3 YOLOv3

La version la plus récente est YOLOv3[87]. Cette amélioration vise à améliorer les performances de prédiction du réseau mais la rapidité de prédiction est diminuée à cause de la complexité grandissante du réseau²⁰⁴.

Bien que YOLOv2 ait des résultats meilleurs sur cette thématique, la détection des entités de petite taille reste problématique. Cela est due aux *Downsampling* du réseau qui extrait la sémantique de l'image mais nuit à la "résolution" de l'image, ce qui accentue la perte des informations "discrètes" associées à des

²⁰⁴La logique de simplicité et d'autonomie initiale du réseau semble ne plus être un critère décisif des chercheurs

entités de petites tailles.

Pour corriger ce défaut, plusieurs modifications ont été faites:

- **Architecture interne du réseau approfondie:** Le réseau YOLO était *primaire*. Relativement peu profond, il n'utilisait pas de composantes caractéristiques des réseaux de l'état de l'art. Avec YOLOv3, cette faiblesse n'est plus d'actualité: ajout de *résidus* par addition et concaténation (*skip-layer*), ajout de *Upsampling*, ajout de profondeur (passage de 53 à 106 couches²⁰⁵). Ces particularités apportent différentes plus-values:
 - **Profondeur du réseau:** L'ajout de couches de convolution permet d'améliorer l'extraction d'attributs des images et, de ce fait, la qualité de la prédiction. Néanmoins, l'impact sur le temps de prédiction est à considérer²⁰⁶.
 - **Skip-layer et résidus:** Dans les premières couches de convolutions, des *résidus* sont transférés afin de maintenir la bonne cohérence des données dans les différentes couches successives²⁰⁷. De même, la présence de *Skip-layer* permet de conserver un bon équilibre entre l'information sémantique obtenue par *Downsampling* et l'information détaillée des *feature map* issues des couches précédentes.
 - **Upsampling:** Le réseau exploite des *feature map* de différentes dimensions. L'utilisation de *résidus* et de *skip-layer* peut être difficile dans les cas d'opérations sur des *feature map* de dimensions différentes. Afin de les mettre à la même échelle, les *feature map* reçoivent un *Upsampling*. La méthode de *Upsampling* n'est pas explicitée²⁰⁸ dans l'article de recherche. Il est donc nécessaire de la choisir selon votre sensibilité personnelle.
- **Prédiction sur 3 échelles:** Alors que YOLOv1/2 réalisait la détection sur un set unique de *feature map*, YOLOv3 le fait sur 3 sets distincts à des échelles différentes (quadrillage appliqué à l'image de différentes dimensions). Plus le *Downsampling* est réalisé au cours du réseau, plus la sémantique de l'image est extraite. Cependant, on perd les détails de l'image et de ce fait sa capacité de détection des petites entités. Afin de corriger cela, une extraction de *feature map* sur 3 sections différentes du réseau permet de réaliser une détection à 3 échelles différentes et ainsi détecter (en simplifiant grossièrement) les entités de petites, moyennes et grandes tailles. Le fonctionnement suit l'idée apportée par les *Feature Pyramid Networks*²⁰⁹ (FPN).

²⁰⁵D'où la plus grande lenteur du réseau comparé à YOLOv2

²⁰⁶Tout n'est que jeu de compromis...

²⁰⁷Voir ResNet pour plus de détails sur la notion de résidus

²⁰⁸A confirmer par une lecture très minutieuse

²⁰⁹Pour plus d'informations, voir Section 12.2.10

- **Anchor boxes:** 9 *anchors* sont utilisées par YOLOv3. Ces *anchors* sont réparties sur les 3 sorties prédictives. Ainsi, pour chaque sortie, 3 *anchors* sont réparties et appliquées respectivement²¹⁰²¹¹. Malgré le fait qu'il n'y ait que 3 *anchors* (5 pour YOLOv2), la présence de 3 sorties prédictives à des échelles bien plus précises (dimension du quadrillage supérieure à 13*13) provoque la création d'un nombre bien plus important de *bounding boxes*. YOLOv3 prédit environ 10x le nombre de *bounding boxes* prédit par YOLOv2.
- **Multi-label:** Il est possible qu'une entité ait plusieurs labels. Par exemple, une femme peut être labellisée *Individu* et *femme*. Avec YOLOv1 et YOLOv2, la classe d'une entité est unique, ce qui est très limitant. Cette limite est due à l'utilisation de la fonction d'activation *Softmax* en sortie prédictive, ce qui produit une prédiction unique (prédiction exclusive). Pour corriger ce défaut, chaque classe est prédite selon une régression logistique²¹², ce qui permet un multi-label pour la classification.
- **Fonction de coût:** La fonction de coût est modifiée pour ses composantes probabilités (confiance et probabilité conditionnelle de classe). L'approche par *Squared error* n'est pas très performante dans le cadre de comparaison de probabilités. Pour corriger ce défaut, l'utilisation de la *Cross-Entropy* permet d'améliorer les performances de cette fonction.

Une illustration du réseau YOLOv3 est visible sur la Figure 116. En considérant la métrique de validation à un IoU de 0.5²¹³, YOLOv3 a des résultats qui dépassent SSD et qui rivalisent avec RetinaNet. Cependant, si le seuil du critère de l'IoU augmente, YOLOv3 perd grandement en performances, ce qui traduit que ce réseau manque de précision dans la prédiction de ses *bounding boxes*. De plus, dans les versions initiales, YOLO avait du mal avec les entités de petites tailles. Avec YOLOv3, les performances pour les entités de petites tailles a augmenté mais la prédiction des entités de tailles intermédiaires et grandes a significativement diminué. Un travail important reste à faire sur la précision de la prédiction. Néanmoins, le réseau reste remarquablement rapide comparé à ses concurrents, ce qui fait de YOLOv3, une alternative parfaitement viable et exploitable²¹⁴ pour les situations nécessitant une vitesse de prédiction élevée comme le *Tracking*.

²¹⁰Les 9 anchors ne sont donc pas appliquées sur toutes les feature map

²¹¹On peut considérer qu'il y a 3 anchors fondatrices qui sont mises à l'échelle de différentes manières

²¹²Comparable à la sigmoïde

²¹³Si l'Iou est supérieur ou égal à 0.5, la prédiction est jugée bonne. C'est une convention pour les compétition d'analyse d'images

²¹⁴D'un point de vue métier, les performances sont à la hauteur des attentes. Il est important de savoir que le référentiel n'est pas la performance applicative mais la performance absolue du modèle contre les autres existants, ce qui peut être contre-productif dans le cadre d'une production du modèle au niveau industriel. En effet, un modèle n'a pas à être forcément le "meilleur" au niveau des benchmarks pour être pertinent. De nombreux autres paramètres sont à considérer.

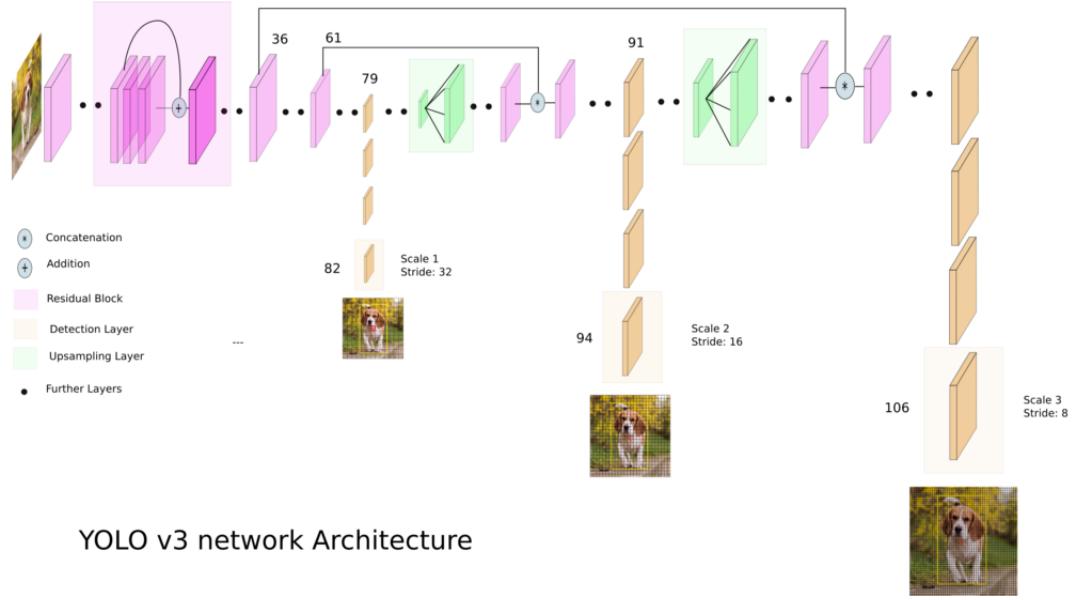


Figure 116: Architecture du réseau convolutif YOLOv3

12.3.7 Single Shot MultiBox Detector (SSD)

SSD est un concurrent de YOLO. Longtemps considéré comme plus performant, les dernières versions de YOLO tendent à relancer la concurrence entre les deux approches tant les performances sont du même ordre de grandeur.

12.3.7.1 SSD

Le nom complet du réseau SSD[71] traduit les spécificités de son architecture. En effet, SSD est un algorithme *Single Shot*, i.e il réalise la tâche de classification et de localisation sur une même étape *Forward*. La prédiction des *bounding boxes* repose sur l'approche *MultiBox*[110] et il réalise une tâche d'*Object Detection* d'où *Detector*.

La version originale de SSD repose sur l'architecture VGG-16 où les couches Full-Connected ont été supprimé. En effet, nous n'exploitons que les capacités d'extraction d'attributs de ce réseau et non sa capacité prédictive. SSD se démarque de YOLO par son approche de *détection multi-échelle*. En effet, au lieu de réaliser une prédiction sur un seul set de *feature map*, SSD va extraire des *feature map* à différentes échelles et réaliser une prédiction distincte pour chacun de ces sets. L'architecture du modèle prédictif (convolutif) est différente pour chaque set de *feature map*, ce qui contourne la problématique de dimension des *feature map*. Une illustration du réseau SSD est visible sur la Figure 117.

La prédiction des *bounding boxes* suit l'architecture *MultiBox*. Contrairement aux approches standards, *MultiBox* propose une architecture par flux parallèles comparable à l'approche *Inception*. Ainsi, à partir d'un même set de *feature map*, plusieurs prédictions sont réalisées selon des méthodes d'extraction d'attributs variables (couches de convolutions différentes et distinctes). Cette méthode est ainsi plus robuste car elle permet une analyse plus fine des *feature map* et de ce fait, une meilleure prédiction. Un exemple illustratif de *MultiBox* est visible sur la Figure 118.

SSD repose sur l'approche *Anchor box*²¹⁵. Ainsi, chaque prédiction est composée de l'offset de la *bounding box* de référence (*l'anchor*) et des probabilités de classes soit $(c+4)*k$ avec c , nombre de classes (contenant la classe *background* qui correspond à aucune classe représentée) et k , nombre d'*anchor*. SSD utilise 6 *anchors* par case.

La fonction de perte de SSD est de la forme:

$$L(x, c, h, l) = \frac{1}{N} (L_{conf}(x, c) + \alpha L_{loc}(x, h, l))$$

Avec L_{conf} , fonction d'évaluation de la classification, L_{loc} , fonction d'évaluation de la localisation, N nombre de *bounding boxes* conservées et α , coefficient de pondération entre la classification et la localisation. C'est un hyperparamètre. L_{conf} repose sur la fonction *Cross-Entropy* pour évaluer la classification et L_{loc} sur *Smooth-L1* pour la localisation. Pour rappel, *Smooth-L1* est définie par:

$$smooth_{L1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

L'utilisation de *Smooth-L1* permet une plus grande souplesse d'apprentissage. En effet, il n'est pas nécessaire d'avoir une précision au pixel-près, ce que favorise la norme L2. Cette contrainte peut être trop forte et limiter l'apprentissage sans réel bénéfice pour l'oeil humain et l'exploitation métier. Pour plus de détails sur la fonction de perte de SSD, référez-vous à l'article de recherche [71]. Étant donné le nombre significativement plus grand du nombre de *bounding boxes* négatives, il est nécessaires de limiter leurs utilisations durant l'apprentissage pour conserver un équilibre entre *bounding boxes* positives et négatives. Pour corriger ce défaut, SSD trie les valeurs de la fonction de confiance (L_{conf}) pour chaque *anchor* par ordre décroissant et sélectionne ces valeurs de manière à conserver une ration 3:1 entre positifs et négatifs pour l'apprentissage. Cette approche est appelée *Hard Négative Mining*.

Les créateurs de SSD conseille l'utilisation de *Data Augmentation* pour renforcer l'apprentissage et rendre le modèle plus robuste. Leur algorithme de génération

²¹⁵Cette approche est récurrente. Nous ne la détaillerons pas ici. Elle est similaire à celle exploitée par Faster R-CNN

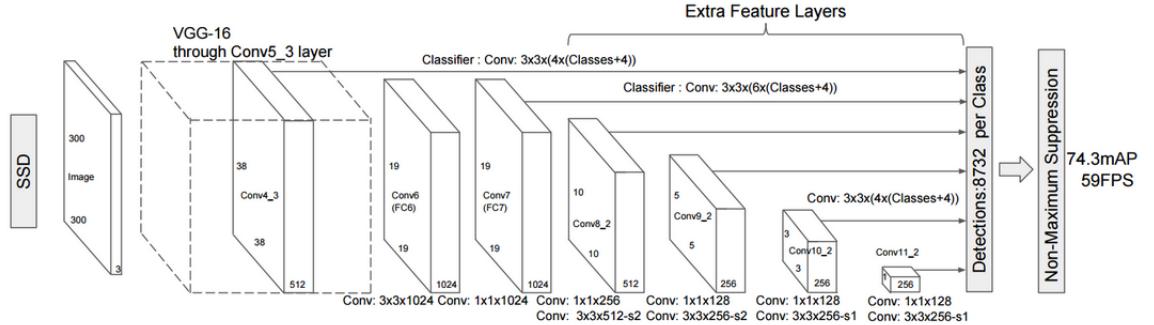


Figure 117: Architecture du réseau SSD

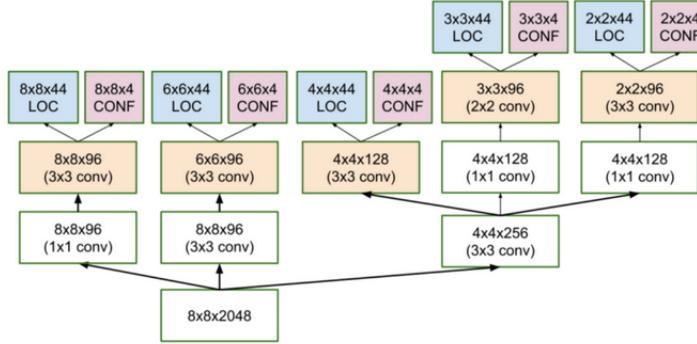


Figure 118: Architecture de Multibox

de données repose sur l'exploitation de la métrique IoU pour créer différentes sous-images conservant une valeur minimum d'IoU. Cette méthode peut être pratique pour apprendre à détecter des entités partielles, détériorées ou de caractéristiques légèrement différentes de celles présentes dans le jeu de données d'apprentissage.

12.3.7.2 Tiny SSD

Les modèles de Deep Learning sont lourds et exigeants en terme de matériels. Cette contrainte est une problématique récurrente pour implémenter ce type de modèle sur des supports de faibles capacités tels que l'embarqué. Pour répondre à cette contrainte, une version plus légère de SSD a été conçue: Tiny SSD: [120].

Ce modèle repose sur l'exploitation de l'architecture *SqueezeNet* et de ses *Fire*

*Module*²¹⁶. Hormis cette particularité, il n'y a pas de nouveauté significative. Pour plus de détails sur l'architecture du modèle, veuillez consulter l'article associée. L'architecture du réseau est le fruit d'une optimisation soutenue entre exploitation des *Fire Module* et des couches de convolution classiques. Il est donc pertinent de suivre à la lettre, le modèle décrit pour une future implémentation.

Ce modèle est remarquable pour l'embarqué du fait de ses performances et de sa taille. En effet, son concurrent direct (Tiny YOLO) est environ 30x plus lourd (60.5Mo contre 2.3Mo) tout en ayant une performance moins élevée (57.1%²¹⁷ contre 61.3%).

12.3.7.3 Deconvolutional Single Shot Detector (DSSD)

Deconvolutional Single Shot Detector[23] est une amélioration importante de SSD. Ses deux principales innovations sont le remplacement du modèle VGG par un ResNet (Residual-101) et l'utilisation de *déconvolutions* avec application de *skip-layer*.

Le réseau initial de DSSD est comparable au modèle SSD. Sur la dernière couche de convolution du modèle SSD, le *Downsampling* a permis de produire une image avec une forte explication sémantique mais une faible résolution de l'image. Comme décris précédemment, SSD réalise une extraction de *feature map* sur les dernières couches du réseau afin d'obtenir différentes échelles et ainsi améliorer la prédiction. Bien qu'efficace, cette méthode n'exploite pas les capacités explicatives obtenues par les dernières couches de convolution. En effet, les *feature map* sont extraites **avant** la (ou les) dernière couche de convolution du réseau. L'idée de DSSD est de ne pas perdre la capacité explicative obtenue lors des dernières couches de convolution. Il ne va donc pas extraire les *feature map* avant ces couches mais **après**. Pour cela, il est nécessaire d'appliquer une méthode pour redimensionner les *feature map* (*Upsampling*) tout en leur redonnant l'information associée à la résolution de l'image perdue durant l'extraction de la semantique de l'image. Pour cela, DSSD exploite donc la *Déconvolution*²¹⁸ pour remettre à l'échelle et des *skip-layer* pour réintroduire l'information de résolution de l'image.

L'action des *skip-layers* est d'additionner les *feature maps* obtenues par *Déconvolution* (porteur de la sémantique) avec les *feature maps* issues de couches précédentes avant *Downsampling* (porteur de la résolution de l'image). Les *feature maps* obtenues possèdent donc les deux informations au lieu d'un compromis que réalisait SSD selon la couche extraite. Une illustration de l'architecture générale est visible sur la Figure 119 et la déconvolution réalisée est détaillée

²¹⁶La description détaillée de cette architecture est disponible dans la Section 6.6.10.1

²¹⁷Métrique mAP sur VOC 2007

²¹⁸Ne pas oublier que ce nom est un abus de langage !

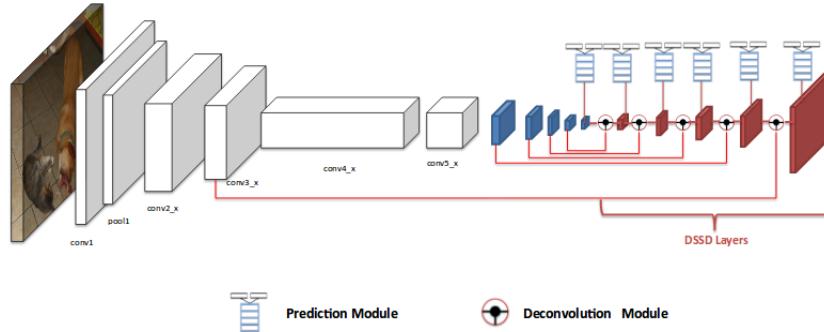


Figure 119: Architecture de Deconvolutional Single Shot Detector (DSSD)

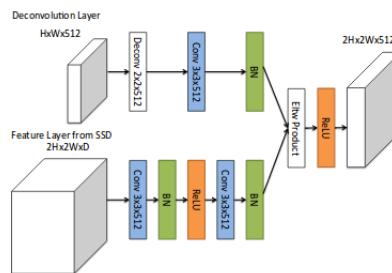


Figure 120: Architecture du Deconvolution module de DSSD

sur la Figure 120.

Les prédictions des *bounding boxes* possèdent toujours les mêmes spécificités que les autres modèles, i.e un problème de régression basé sur la prédiction de probabilités de classes et de dimensions (plus précisément offset) de la *bounding boxes*. La spécificité ajoutée par DSSD est d’exploiter l’architecture *résiduelle* dans son module prédictif. Plusieurs variantes sont proposées par les créateurs de DSSD et sont visibles sur la Figure 121.

Afin de supprimer les *feature map* redondantes, DSSD exploite l’algorithme *Non-Max Suppression* pour réaliser une sélection des *feature map* à conserver. De même, son approche pour la fonction de perte est comparable à DSD et réalise toujours un tri sur les *feature map* à exploiter pour l’apprentissage afin de garantir un ratio 3:1 entre positif et négatif (approche appelée *Hard example mining*).

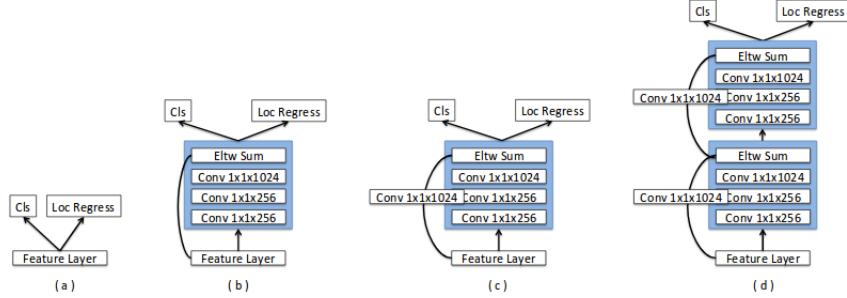


Figure 121: Architecture du module prédictif de DSSD

12.3.8 RetinaNet

Une des contraintes principales des modèles d'*Object Detection* est la différence de distribution entre les classifications positives et négatives du fait du nombre très importants de *bounding boxes* proposées pour chaque image. Deux approches sont exploitées aujourd’hui: exploiter une sous-parties des données prédites afin d’exploiter une distribution avantageuses (*Hard Negative Mining* par exemple) ou appliquer une pondération spécifique dans la fonction de coût. Le modèle *RetinaNet*[69] exploite la seconde approche en proposant une nouvelle fonction de perte: *Focal Loss*.

12.3.8.1 Focal Loss

Focal Loss est une nouvelle fonction de perte basée sur *Cross-Entropy*. Son approche propose de pondérer la valeur de l’erreur produite en fonction des probabilités issues des prédictions afin de favoriser l’expression des erreurs de prédiction. En effet, *Cross-Entropy* impose une erreur (même faible) pour toute prédiction non parfaite, i.e différente de la valeur de la données d’apprentissage (souvent 1 pour la classe correspondante). Cette imperfection du modèle n’a pas d’influence sur la performance du modèle. En effet, la prédiction reste inchangée que la probabilité soit de 1 ou de 0.7 par exemple. Cependant, pour la fonction de perte, cette différence est majeure car les vraies erreurs sont noyées par l’effet de bord produit par les bonnes prédictions "imparfaites". Afin de pondérer les vraies erreurs du modèle, *Focal Loss* propose une pondération de l’erreur selon sa probabilité prédite. Elle est définie par:

$$p_t = \begin{cases} p & \text{if } y_{\text{classe}, \text{ref}} = 1 \\ 1 - p & \text{otherwise} \end{cases}$$

$$FL(p_t) = -\alpha(1 - p_t)^\gamma \log(p_t)$$

Avec α et γ , hyperparamètres du modèle. En supposant $\gamma = 2$ et $\alpha = 1$, si $p_t = 0.9$ alors son erreur sera 100x plus faible que l’erreur produite par *Cross-*

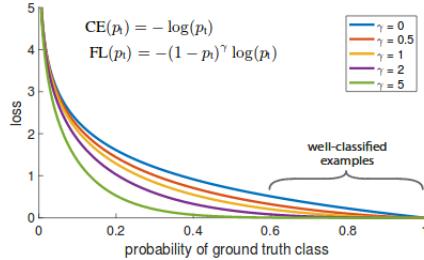


Figure 122: Erreur obtenue par Focal Loss selon γ

Entropy par exemple. Cette réduction permet de s'émanciper du déséquilibre entre les distributions de classes et d'éviter l'étape de *sampling* consistant à ramener le minibatch d'apprentissage à un ratio 3:1. Le comportement de *Focal Loss* est visible sur la Figure 122. Pour $\gamma = 0$, *Focal Loss* est identique à *Cross-Entropy*. Expérimentalement, les créateurs du modèle ont observé que les meilleurs résultats expérimentaux sont obtenus avec $\gamma = 2$. Avec cette configuration, on peut observer que l'erreur pour les probabilités supérieures à 0.6 est quasi-nulle. Néanmoins, cette approche pénalise la confiance du modèle pour ne considérer que la prédiction finale. Ainsi, d'un point de vue métier, le modèle favorise les bonnes prédictions au détriment de la confiance en ses prédictions. Par conséquent, bien qu'il y ait une amélioration de prédiction empiriquement, les résultats sont moins fiables. Il peut donc être délicat d'employer ce genre de métrique lorsque l'on recherche un résultat avec une forte probabilité de certitude.

12.3.8.2 Le modèle RetinaNet

L'architecture de *RetinaNet* ne présente pas de nouveautés notables et repose sur des architectures connues. Ainsi, *RetinaNet* exploite une architecture *Resnet* pour l'extraction d'attributs associée à un *Feature Pyramid Network*²¹⁹ (FPN) pour extraire les *feature map* de prédiction. L'approche *Anchor boxes* est utilisée pour définir les *bounding boxes*. La classification et la détermination des coordonnées de la *bounding boxes* sont définies par deux réseaux *Full Convolutionnal Network* indépendants, i.e ils ne partagent pas les mêmes paramètres. Une illustration de l'architecture de *RetinaNet* est visible sur la Figure 123.

L'utilisation de procédés ayant démontré leurs efficacités associée à une fonction de perte plus performante que les modèles précédents font de *RetinaNet*, l'un voire le modèle le plus performant actuellement. Seule sa vitesse de prédiction est préjudiciable, faisant de YOLO la seule alternative véritablement convaincante selon les circonstances.

²¹⁹Voir Section 12.2.10 pour plus de détails

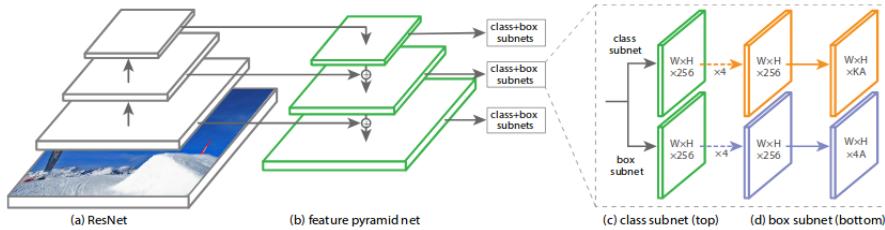


Figure 123: Architecture du réseau RetinaNet

12.3.9 Comparatif des modèles récents

Sur la Figure 124, nous pouvons observer un comparatif sur le jeu de données COCO selon la métrique mAP. Comment on peut le constater, les modèles *Retina* sont les plus performants que ce soit en temps de prédiction ou de précision. Seul YOLO est significativement plus rapide en contrepartie d'une précision moindre. Il est important de comprendre que ce comparatif est réalisé selon un mAP strict. Il n'est pas toujours pertinent d'avoir une prédiction avec une précision très importante. En effet, d'un point de vue métier, il n'est pas forcément utile d'être au pixel-près.

Sur la Figure 125, le même comparatif est réalisé avec un mAP placé à 0.5²²⁰. L'exigence de précision est donc plus faible mais reste suffisamment stricte pour garantir une exploitation métier satisfaisante. On observe que YOLov3 devient significativement meilleur que précédemment. Sa vitesse d'exécution est significativement plus rapide que les autres modèles et ce, pour une qualité de prédiction meilleure ou équivalente. Seule FPN FRCN est plus précis en contrepartie d'un temps de prédiction nettement plus important.

On peut donc conclure que *Retina* est le choix de référence pour la précision du modèle lorsqu'une forte exigence est requise. Au contraire, lorsque l'exigence est moindre, YOLov3 offre un compromis vitesse/précision remarquable et FPN FRCN, la meilleure précision malgré un temps de calculs élevé.

²²⁰C'est une métrique standard pour les compétitions de modèles

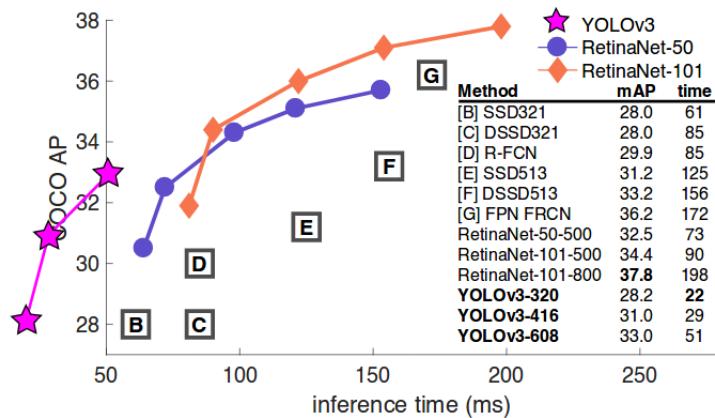


Figure 124: Comparatif des modèles d'Object Detection

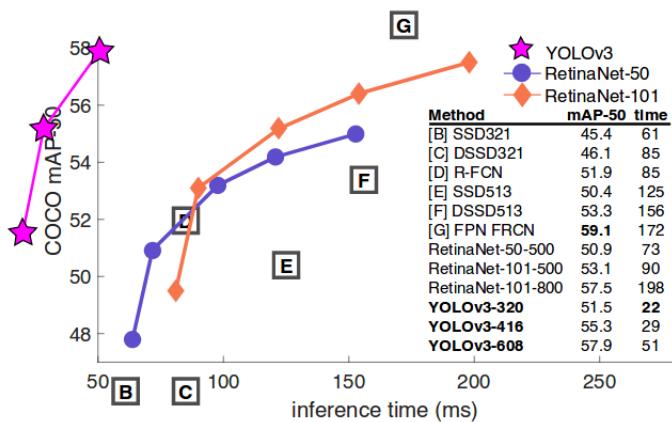


Figure 125: Comparatif des modèles d'Object Detection pour un mAP à 0.5 IoU

13 Application au traitement du langage écrit

13.1 Introduction

13.1.1 La loi de Zipf et Mandelbrot

Au 20^{ème} siècle, Zipf proposa une loi, nommée *Loi de Zipf* pour expliquer la fréquence d'apparition d'un mot au sein d'un texte (volumineux). Selon cette loi, si mot^n est le $n^{ième}$ mot le plus fréquent, alors sa fréquence d'apparition est définie par $Z(n) = \frac{K}{n}$ avec K, une constante. Supposons K=1, alors la fréquence de ce mot est de $\frac{1}{n}$.

Pour rappel, la Figure 126 illustre son comportement pour K=1. Nous pouvons observer que la répartition statistique des mots est très fortement déséquilibrée. Des mots sont omniprésents alors que d'autres sont quasi-absents. Ce comportement justifie des difficultés historiques du traitement du langage naturel car les modèles reposant sur des fondations statistiques, cette particularité constitue une contrainte importante. Une seconde interrogation se porte sur la nature des mots fréquents et peu fréquents. Il a été montré que les mots récurrents sont des mots de liaisons grammaticales (adverbes, pronoms, déterminants...) alors que les mots illustratifs (noms, adjetifs...) sont plus rares. Cette particularité accentue la difficulté d'analyse car l'information discriminante est *rare*.

Dans les faits, la loi de Zipf est incomplète et a été généralisée par Mandelbrot. En effet, Zipf a été vivement critiqué pour avoir exploité des mots non *lemmatisés*²²¹. La loi de Mandelbrot est ainsi définie par $M(n) = \frac{K}{(a+bn)^c}$ avec a,b,c et K des constantes.

Cette loi illustre des problématiques récurrentes en traitement du langage. Le comportement statistiques des données soulève des contraintes mathématiques notamment des biais pour manque de représentativité²²². La faible proportion de mots explicatifs nuit aux capacités de discrimination des modèles et les rend plus sensible au bruit des données d'apprentissage. Le traitement du langage naturel (*Natural Language Processing*) est, aujourd'hui encore, un problème partiellement résolu et la recherche toujours fortement active.

13.1.2 Pré-traitement d'un texte

Afin d'exploiter une donnée textuelle, deux catégories de pré-traitement sont nécessaires. La première consiste à *normaliser* le texte afin de considérer une forme qui facilite l'apprentissage et limite le bruit intrinsèque lié à une distribution de données peu exploitable. La seconde permet de représenter un texte dans un format compréhensible par un réseau neuronal. En effet, seules

²²¹Cette notion est explicitée dans la suite de cette introduction

²²²Des mots peuvent ne pas être représentés dans les données d'apprentissage par exemple, ce qui est très délicat à exploiter pour les approches probabilistes

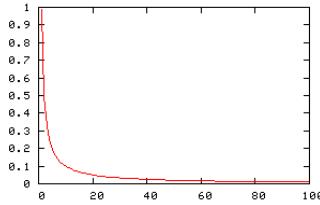


Figure 126: Loi de Zipf pour $K=1$

les valeurs numériques sont exploitables. Il est donc nécessaire de trouver une approche permettant de représenter une valeur textuelle (donc catégorielle) en valeur numérique.

13.1.2.1 Tokenization

Un réseau de neurones exploite des séquences de valeurs (série de pixels dans le cadre d'une image par exemple). Afin d'exploiter un texte, il est nécessaire de rendre son architecture sérielle, i.e transformer un texte en une suite d'entités de même nature. La Tokenization est le procédé qui permet de réaliser cette tâche. Lors de la Tokenization, une entité unitaire est appelée *Token*.

Important: Dans la suite de cette introduction, nous considérerons l'alphabet latin. Bien que le raisonnement soit identique pour toutes les langues, les conclusions peuvent variées selon les spécificités de chacune.

Considérons un texte quelconque. Quelles sont les entités qui le constituent ?

Source	Entité sous-jacente
Texte intégral	Phrases
Phrases	Mots
Mots	Syllabes
Syllabes	Lettres

Nous observons différentes catégories d'entités fondamentales. Dans les faits, un consensus a choisi le mots comme type de Token. Néanmoins, dans le cadre de certaines problématiques, plusieurs types de Tokenization peuvent être exploités²²³.

Bien qu'en apparence simple, la Tokenization présente des difficultés notables dont la solution relève essentiellement d'une sensibilité personnelle plus que d'une approche scientifiquement démontrable. Parmi ces difficultés, nous pouvons citer l'exploitation de la ponctuation et des symboles (y compris les jeux graphiques tels que les *émoticônes* par exemple), l'interprétation des mots composées et/ou à particules ou encore les ambiguïtés syntaxiques (notamment liées

²²³Ceci sera développé dans la suite de cette introduction.

à l'apostrophe). De même, pour l'anglais, l'interprétation des contractions pose des difficultés. Comment traiter *aren't* ? Par *aren't*, *arent*, *aren t*, *are n't* ?

13.1.2.2 Lemmatisation et racinisation - A FAIRE

En cours

13.1.2.3 Stopwords

Lors d'une tâche de classification (Analyse sentimentale, catégorisation de documents par exemple), il est nécessaire d'extraire l'information discriminante d'un texte afin de pouvoir le classifier. Pour cela, il est nécessaire de mettre en valeur, les mots porteurs d'informations.

Les mots les plus fréquents dans un texte sont des entités de liaisons grammaticales. Par exemple, nous pouvons citer les déterminants, pronoms ou encore certains adverbes. Ces mots n'ont pas de pouvoir de discrimination car leur expression est neutre et/ou sont trop représentés statistiquement pour être discriminant. Par exemple, le mot *le* n'a aucun pouvoir de discrimination alors que l'auxiliaire *être* est trop représentés pour être discriminant.

Ces mots sont définis comme *Stop Words*, i.e des mots au pouvoir de discrimination nul. Ils sont donc inutiles pour la tâche de classification qui repose sur des entités au fort pouvoir explicatif. Une approche classique de pré-traitement de texte est donc de supprimer ces mots des textes à analyser pour ne conserver que des mots pertinents.

Ainsi, par exemple, supposons les tokens suivants: [Cet, article, est, une, arnaque, car, la, qualité, semble, mauvaise !]. Après suppression des *Stop Words*, nous obtenons : [article, arnaque, qualité, semble²²⁴, mauvaise].

Cette approche supprime les mots sans incidence sur la décision du réseau. La prédiction est donc plus rapide car le volume de données traitées est grandement diminué²²⁵ et l'apprentissage plus performant grâce à la plus grande spécialisation des données d'apprentissage. Néanmoins, cette méthode ne conserve pas la sémantique du texte ni sa structure syntaxique.

13.1.2.4 Problématique et pré-traitement

Toutes les modifications précédentes (sauf Tokenization) détériorent l'intégrité sémantique et syntaxique d'un texte. Pour certaines tâches, cette détérioration est sans importance. Dans le cas d'une classification par exemple, cette perte a souvent des conséquences négligeables sur la qualité du résultat. Au contraire,

²²⁴Il est possible de discuter sur la pertinence ou non de ce mot.

²²⁵Dans le cadre d'analyse intégrale d'oeuvres ou de documents

pour les problématiques qui demandent de conserver une bonne structure sémantique et syntaxique, appliquer cette approche fera échouer l'apprentissage de manière critique. Il est donc important d'étudier les caractéristiques du problème à résoudre avant d'appliquer un pré-traitement sur les données.

Généralement, pour les tâches qui ne demandent pas de création de texte comme valeur de sortie, l'application de pré-traitement est envisageable. Nous pouvons citer l'analyse sentimentale, la Recherche d'informations, la désambiguïsation par exemple. Au contraire, lorsqu'un texte syntaxiquement cohérent doit être généré, utiliser un pré-traitement est dangereux. La traduction automatique, les systèmes Question-Réponse, les générateurs de texte ou encore l'Image Captioning sont des exemples classiques.

Néanmoins, les approches récentes de classification, notamment basées sur les RNN et les principes d'*Attention* imposent la conservation de la structure sémantique et syntaxique du texte. De ce fait, le pré-traitement des données tend à devenir de plus en plus "néfaste" même sur des tâches qui s'y prêtent traditionnellement.

13.2 Représentation vectorielle - A FAIRE

13.2.1 Projection Word-Based - A FAIRE

En cours

13.2.2 Projection Character-Based

En cours

13.3 Classification

La classification de texte est une thématique qui présente un fort intérêt d'un point de vue métier. Son objectif est, comme dans le cadre de l'image, de classer le texte dans une (ou plusieurs) catégorie qui correspond à ses spécificités. La classification des Tweets selon leurs "toxicités" (neutre, harcèlement, discrimination...) est un exemple classique d'application.

Cette thématique présente une grande diversité d'architecture exploitable. En effet, il est possible d'utiliser les réseaux convolutifs, les réseaux récurrents ou encore l'union de ces deux types de réseau. Nous verrons, dans cette partie, quelques unes des solutions standards de l'état de l'art et les spécificités théoriques observées dans le cadre de la classification de texte.

Aujourd'hui, il n'y a pas de consensus sur la meilleure famille d'architecture pour réaliser de la classification de texte. Néanmoins, les réseaux convolutifs sont plus rapides à apprendre et à réaliser des prédictions. Par contre, les réseaux

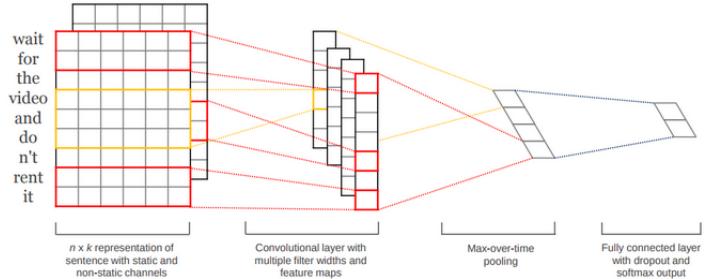


Figure 127: Architecture de CNN pour la Classification de Sentence

récurrents ont une plus grande capacité de compréhension de la sémantique du texte, ce qui, dans le cas de texte difficile ou ambiguë, peut permettre de meilleurs résultats que les réseaux convolutifs.

13.3.1 Classification par CNN

13.3.1.1 CNN for Sentence Classification

CNN for Sentence Classification[53] est un modèle précurseur créé en 2014. Ce modèle considère le texte comme une séquence de mots et analyse les N-Grams obtenus selon les spécificités du filtre de convolution choisi. Cette approche néglige le contexte sémantique global du texte pour se focaliser sur le comportement local des groupes de mots.

Le réseau utilisé présente une architecture simple. Les Words Embeddings correspondant aux mots du texte sont générés aléatoirement ou issus d'une librairie de vecteurs pré-entraînés. Dans le cas où les vecteurs initialement sont pré-entraînés, la valeur des vecteurs est figée durant l'apprentissage. Il est possible de cumuler plusieurs Words Embeddings pour un même mot en concaténant les vecteurs. L'architecture du réseau applique une approche *Inception*. Chaque flux exploite un kernel de dimension différente afin de produire différents N-Grams (le nombre de filtre est un hyper-paramètre). Il n'y a qu'une couche de convolution par flux. Le réseau est donc très peu profond mais large. Une couche de MaxPooling est appliquée sur chaque flux pour extraire les N-Grams les plus pertinents. Ces valeurs sont ensuite envoyées à une couche Full-Connected pour réaliser la prédiction. Le modèle est visible sur la Figure 127.

Ce modèle présente une efficacité convenable sur des textes simples et de petites tailles mais est insuffisant dans le cas contraire. Néanmoins, il mit en avant les capacités des réseaux convolutifs pour la classification d'un texte alors "dominé" théoriquement par les réseaux récurrents.

13.3.1.2 Améliorations notables

De nombreuses améliorations de *CNN for Sentence Classification* ont été proposées. Néanmoins, l'architecture générale reste très similaire.

- **Dynamic Convolutional Neural Network (DCNN)**[50]: Ce modèle exploite K-max Pooling pour capturer les relations courtes et longues distances dans le texte. Les kernels des couches de convolutions sont aussi plus larges et diminuent progressivement, de même que la dimension de sortie des couches de K-max Pooling. Son architecture est visible sur la Figure 128.
- **Multi Channel Variable size CNN (MV-CNN)**[125]: Ce modèle est très similaire à *Dynamic Convolutional Neural Network* mais exploite l'idée des Embeddings multi-sources. Ainsi, le modèle applique la même extraction d'attribut indépendamment pour chaque matrice initialisées par chacune des librairies d'Embeddings et réalise la prédiction à partir de la concaténation des résultats obtenus pour chaque flux²²⁶. L'intuition repose sur le fait que les Words Embeddings sont définies selon des méthodes différentes mais surtout des jeux de données différents. Selon la source d'apprentissage, le contexte global peut changer et de ce fait, des mots rares ou absents dans une librairie peuvent être significatifs dans une autre. Son architecture est visible sur la Figure 129. Une variante très proche, intitulé **Multi Group Norm Constraint CNN (MG(NC)-CNN)**[135], repose sur une architecture similaire mais simplifiée en plus d'un ajout de contraintes de régularisation pour compenser la simplicité du modèle.

13.3.1.3 Very Deep CNN

Les modèles précédents se basent sur le mot comme unité de référence et exploitent un réseau peu profond mais large. Au contraire, *Very Deep CNN* (VDNN)[14] exploite la lettre comme unité de référence et repose sur une architecture très profonde mais peu large. La ponctuation et les symboles sont considérés comme des lettres et utilisés par le réseau. Des projections vectorielles spécifiques aux lettres sont utilisées pour représenter chaque lettre. Afin de considérer la spécificité de l'unité choisie, le nombre de filtres est significativement augmenté mais la dimension du kernel reste dans le même ordre de grandeur (<5). Afin de progressivement diminuer la dimension des 130.

Du fait de la profondeur du réseau et du volume de données à analyser, ce modèle est significativement plus lent à apprendre et à réaliser ses prédictions.

²²⁶Si 2 librairies sont utilisées, il y aura deux flux.

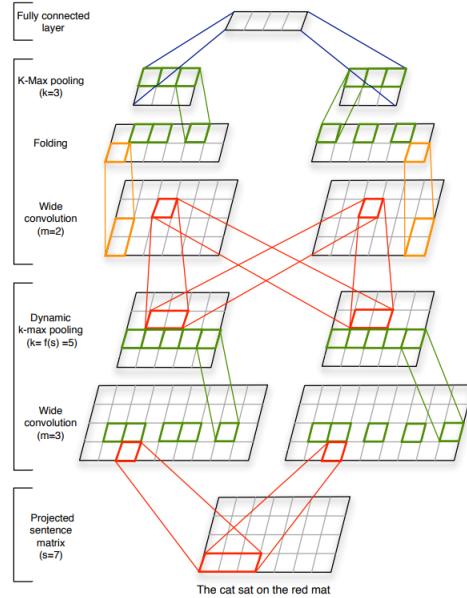


Figure 128: Architecture de Dynamic Convolutional Neural Network

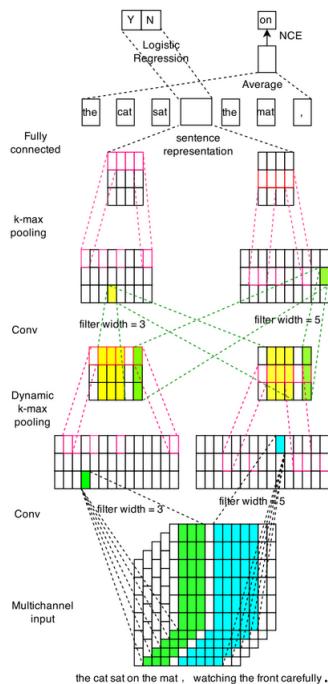


Figure 129: Architecture de Multi Channel Variable size CNN

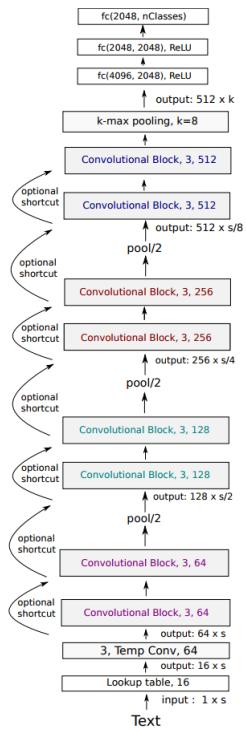


Figure 130: Architecture de Very Deep CNN

13.3.1.4 Comparatif expérimental sur les architectures CNN

Deux études expérimentales ont été réalisées afin d'étudier les différentes approches et définir laquelle semble être la plus performante.

La première, réalisée par Hoa T. Le[63], a montré que dans le contexte de la classification de texte:

- Les modèles larges et peu profonds sont meilleurs que les modèles très profonds. De plus, du fait de la complexité des modèles profonds, l'apprentissage de ce type de réseau est plus difficile et gourmand en temps et en ressources matérielles.
- L'utilisation de *Global Max Pooling* avec un modèle peu profond mais large semble aussi performant que l'utilisation du *Max Pooling* avec un modèle profond. La capacité très généralisante semble suffisamment discriminante dans le contexte de la classification.
- Les modèles exploitant le mot sont plus performants que les modèles basés sur la lettre. De plus, les modèles *character-based* imposent l'exploitation d'un réseau très profond et de ce fait, s'associe aux problématiques de l'apprentissage et de ressources.

Ainsi, les réseaux de classification de textes s'opposent aux dogmes des réseaux de classification d'images qui reposent sur l'idée que le réseau doit être très profond pour être efficace.

La seconde étude, réalisée par Ye Zhang[136], apporte une aide sur le paramétrage des hyperparamètres des modèles de classification de textes. Ainsi, plusieurs conseils ont été proposés:

- Durant la première approche, exploitez des Word Embeddings non statiques²²⁷ au lieu de *One-Hot vector*. Concaténer plusieurs Word Embeddings ne semble pas améliorer significativement les performances de la classification.
- Exploitez une dimension de kernel entre 1 et 10. Réalisez éventuellement un *GridSearch* autour de la meilleure valeur de kernel obtenue.
- Variez le nombre de filtres de 100 à 600. Durant la recherche, utilisez un DropOut (< 0.5) et une Max-Norm contrainte importante. Gardez en tête qu'il y a un compromis à faire entre nombre de filtres et temps d'apprentissage.
- Plus le nombre de *feature map* au sein du réseau augmente, plus les contraintes de régularisation doivent être fortes.

²²⁷Variables durant l'apprentissage et non figés.

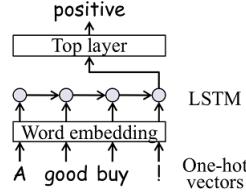


Figure 131: Architecture de LSTM Deep Sentence Embedding

- Ne pas négliger la Cross-Validation pour la recherche d'hyperparamètres pour éviter les biais d'apprentissage. Néanmoins, cette méthode est gourmande en temps de calculs.

13.3.2 Classification par RNN

Il existe des modèles de classification de textes basés sur les réseaux récurrents (spécialement avec des cellules LSTM ou GRU). Nous allons étudier plusieurs modèles de l'état de l'art reposant sur cette architecture.

13.3.2.1 LSTM Deep Sentence Embedding

LSTM Deep Sentence Embedding[80] est un réseau simple à réaliser et présente des performances satisfaisantes. Il est composé d'un réseau LSTM-RNN et d'une couche Full-Connected qui exploite le vecteur d'états cachés de la dernière cellule LSTM. Ce vecteur résume l'information utile du texte analysé et de ce fait, représente un Embedding du document. Il utilise le mot comme entité-unité. La Figure 131 illustre l'architecture de ce modèle.

La faible dimension du vecteur caché permet de s'émanciper des couches de Pooling (ou autres méthodes de *Downsampling*). Néanmoins, l'espace de représentation est de faible dimension et de ce fait, favorise la perte d'informations. De plus, bien que le LSTM corrige grandement le problème de mémoire à long terme des réseaux récurrents, la perte d'informations dans le cas d'analyse de textes volumineux n'est pas à négliger.

13.3.2.2 Discriminative RNN

Discriminative RNN[126] est très similaire à *LSTM Deep Sentence Embedding* mais propose une solution à la perte d'informations liée à la sélection du vecteur d'états cachés. Pour cela, au lieu d'exploiter le vecteur de la dernière cellule, l'Embedding du document sera égal à la moyenne des différents vecteurs d'états cachés du LSTM-RNN. Cette approche permet d'être moins sensible à la perte de mémoire liée à ce type de réseau. Son architecture est visible sur la Figure 132.

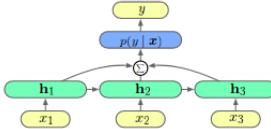


Figure 132: Architecture de Discriminative RNN

13.3.2.3 Hierarchical Attention Networks

Hierarchical Attention Networks[124] pose le postulat qu'un mot et qu'une phrase ont deux pouvoirs explicatifs dissociés. Ainsi, afin de considérer ces deux sources d'informations, le réseau présente une architecture à deux niveaux composée de deux réseaux Bi-LSTM imbriqués.

Le premier niveau analyse le texte en isolant chacune des phrases et en analysant l'importance des mots qui constituent la phrase. Le second niveau analyse le texte en fonction des phrases dont l'information est calculée à travers le premier réseau. La sortie de ce réseau sera exploitée par la couche prédictive composée de couches Full-Connected.

Supposons un texte composé de N phrases et chacune des phrases possède M mots. Le premier réseau considérera une entrée composée de M éléments successifs et réalisera N cycles prédictifs. Il produira donc $N \times M$ vecteurs d'états cachés (et N vecteurs d'états cachés finaux). Le second réseau considérera une entrée composée de N éléments successifs (chacun représentant l'information d'une phrase) et produira un cycle uniquement. Chaque entrée du second réseau correspond à la sortie obtenue à la fin d'un cycle du premier.

Cette architecture exploite aussi la notion d'*Attention* (Voir Section 11.2). Ainsi, chaque entrée du second réseau correspond à une moyenne pondérée des vecteurs de contexte du premier réseau liés à cette entrée. La pondération est dynamique et apprise durant l'apprentissage. Cette méthode est aussi appliquée à la sortie du second niveau. Ce réseau est illustré par la Figure 133.

13.3.3 Classification par CNN-RNN

Des architectures ont été développées en utilisant à la fois l'architecture CNN et RNN en se basant sur l'hypothèse que l'utilisation des deux architectures permettrait d'unir leurs points forts (compréhension sémantique du RNN et la détection de patterns locaux du CNN).

13.3.3.1 CNN-RNN

Le modèle *CNN-RNN*[10] est composé de deux parties:

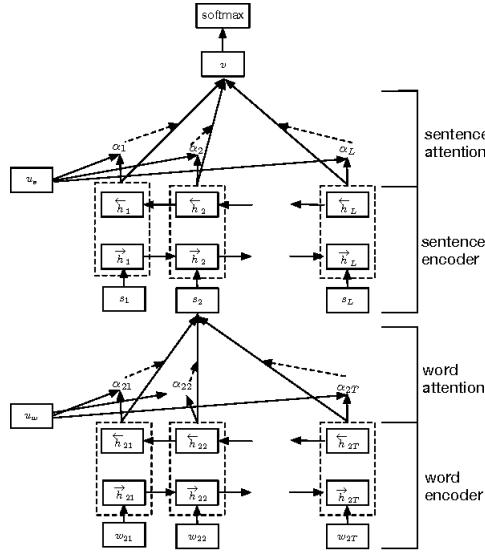


Figure 133: Architecture de Hierarchical Attention Networks

- **Extraction du vecteur de contexte:** Cette action est réalisée par un réseau convolutif (CNN). Ce réseau prend une séquence de *Word Embeddings* en entrée et produit un vecteur représenté par la sortie d'une couche Full-Connected²²⁸. Le réseau CNN présente une architecture peu profonde mais large comparable au réseau *NN for Sentence Classification*.
- **Prédiction du label:** Cette action est réalisée par un réseau récurrent (RNN). Ce réseau exploite le vecteur de contexte pour initier son état caché initial en plus d'être additionné aux autres entrées des cellules RNN. L'addition suit la relation suivante:

$$h^{(t)} = f_{activation}(W_h x_t + U_h h_{t-1} + W_T T)$$

Avec T , vecteur de contexte et $W_T \in R^{q*t}$ avec q , dimension de l'état caché de la cellule RNN et t , dimension du vecteur de contexte.

Une couche *softmax* est appliquée en amont de chaque sortie du RNN afin de prédire le label le plus probable selon la sortie de la cellule RNN ciblée.

La cellule RNN peut être remplacée par une cellule LSTM ou GRU afin de limiter l'impact de la perte de mémoire. Dans ce cas, le vecteur de contexte est ajouté à chacune des différentes *Gates* de la cellule. La cellule $n+1$ utilise la prédiction faite par la cellule $n-1$, i.e son tag, afin d'orienter sa

²²⁸La fonction d'activation de la couche est linéaire, i.e aucune fonction d'activation particulière n'est exploitée.

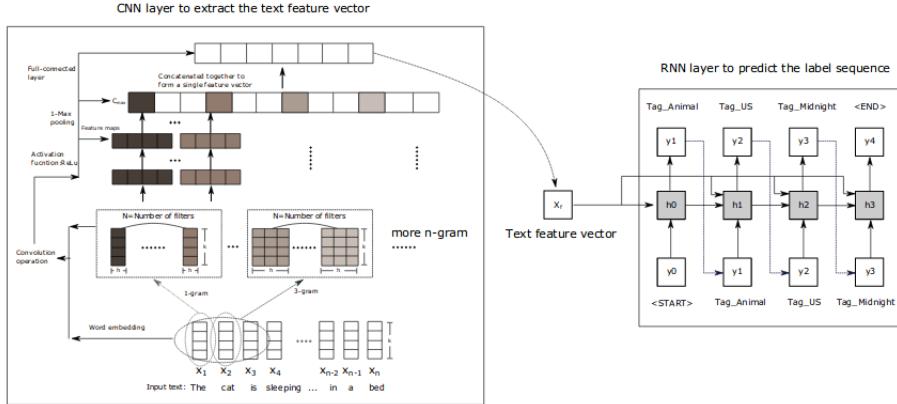


Figure 134: Architecture de CNN-RNN

prédition. Le réseau n'a connaissance du texte qu'à travers le vecteur de contexte proposé par le CNN. Il est donc important de s'assurer que la dimension de ce vecteur est suffisante pour représenter la séquence à prédire²²⁹. La Figure 134 illustre cette architecture.

13.3.3.2 C-LSTM

Le modèle *C-LSTM*[137] est composé de deux parties:

- **Extraction des N-Grams:** Le réseau convolutif applique n filtres (kernel identique) sur la séquence à prédire. Les n *feature map* obtenus sont concaténés selon l'axe des colonnes. Le réseau présente deux particularités:
 - Aucune méthode de DownSampling est utilisée (stride, Pooling...) afin de conserver la cohérence sémantique de la séquence²³⁰. Une alternative proposée par [116] reprend l'architecture du C-LSTM mais applique du Pooling afin de réduire la dimension des *feature map* en sortie du réseau convolutif et ainsi, augmenter la vitesse de prédiction et d'apprentissage du réseau (voir Figure 136).
 - Un seul kernel est exploité afin de permettre une bonne concaténation et la cohérence des données concaténées. Néanmoins, exploiter différents kernels est possible en s'assurant du respect des dimensions des *feature map* notamment via l'usage de *padding*. **Cette spécificité est incertaine et relève de mon interprétation du**

²²⁹Plus le texte est complexe et volumineux, plus la dimension du texte doit être grande. Néanmoins, l'utilisation de l'*Attention* peut aider à éviter cette proportionnalité dimensionnelle.

²³⁰Extraire des sous-séquences détruit l'intégrité de la continuité sémantique

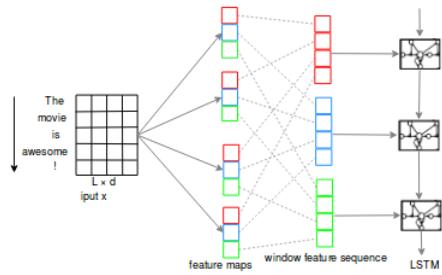


Figure 135: Architecture de C-LSTM

papier de recherche. Veuillez vous référer au papier associé afin de confirmer cette affirmation.

- **Extraction d'un vecteur de contexte:** Le réseau RNN (LSTM) est utilisé afin d'interpréter le comportement sémantique du texte à partir des N-Grams obtenus par le réseau convolutif. Ainsi, l'entrée séquentielle i du RNN correspond aux N-Grams concaténés à la position i des différentes *feature map*²³¹.

Une couche *softmax* prend en entrée le **dernier état caché** du LSTM afin de réaliser sa prédiction. Il n'y a donc qu'une prédiction réalisée (à partir du dernier état caché) en fin de réseau alors qu'avec le modèle *CNN-RNN*, la prédiction se fait *en continu* à partir des différents états cachés du RNN.

13.3.3.3 AC-BLSTM

Le modèle *symmetric Convolutional Bidirectional LSTM Networks* (AC-BLSTM) [66] propose une modification des couches convolutives nommée *Asymmetric Convolution* afin d'extraire l'information de la séquence à prédire.

Supposons un séquence S de dimension L donc le j -ième mot x_j est représenté par un *Word Embeddings* tel que $x_j \in R^d$. Nous avons donc:

$$S = [x_1, \dots, x_L]$$

$$x_i = [x_{i,1}, \dots, x_{i,d}]$$

Supposons k dimension du kernel du filtre d'une couche de convolution. Une convolution standard (sur une dimension) réalise donc une opération de dimension $k*d$, i.e analyse k mots successifs représentés par un vecteur de dimension d .

²³¹Dans les faits, les *feature map* sont représentées par une matrice unique de dimension $i*j$ avec i nombre de N-grams et j , nombre de filtres. Les entrées du RNN correspondent donc aux lignes de la matrice.

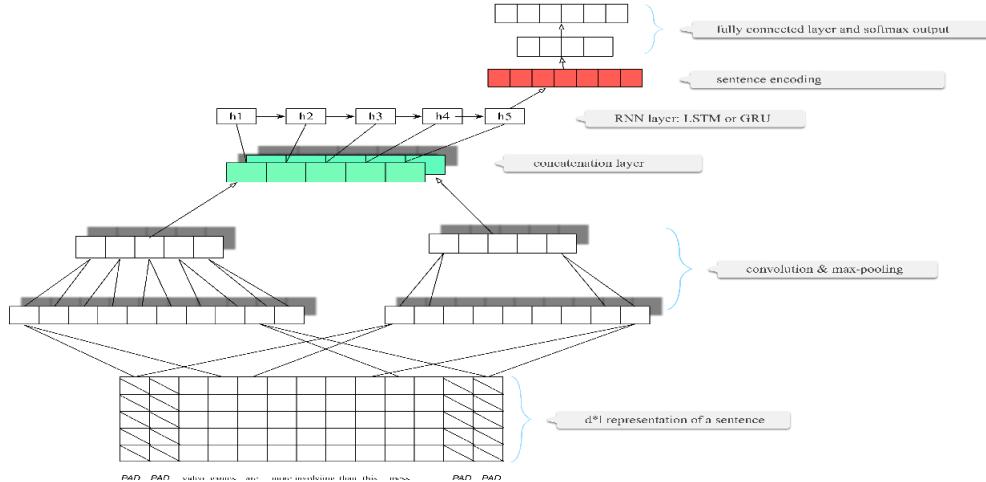


Figure 136: Architecture de C-LSTM avec Pooling

Une convolution asymétrique réalise la convolution en deux étapes convolutives:

- La première étape est représentée par un kernel de dimension $1*d$. Ainsi, l'action de cette opération correspond à une projection vectorielle définie par $R^d \rightarrow R^1$. En d'autres mots, le *Word Embeddings* est représenté par une valeur uniquement à la fin de cette étape.
- La seconde étape est représentée par un kernel de dimension $k*1$ où k correspond à la taille du filtre choisi initialement. Le fonctionnement de cette étape est comparable à une convolution standard sur des données à 1 dimension.

Expérimentalement, la convolution asymétrique a présenté des résultats qualitatifs. De même, on peut observer une ressemblance avec l'approche *Depthwise-Pointwise* très utilisée pour l'analyse d'image et les réseaux faibles consommation.

La concaténation des *feature map* et l'exploitation du réseau récurrent de AC-BLSTM est comparable à celle de C-LSTM. Deux différences importantes sont à noter pour cette dernière.

- Le réseau AC-BLSTM exploite un réseau *Bi-directionnel* afin d'améliorer les capacités prédictives du réseau LSTM.
- La prédiction du label est réalisée à partir d'une couche softmax qui prend en entrée l'intégralité des états internes du BLSTM concaténés. Ainsi, supposons un réseau BLSTM composé de m cellules de dimension n . L'entrée de la couche softmax sera alors de dimension $m * (2 * n)$.

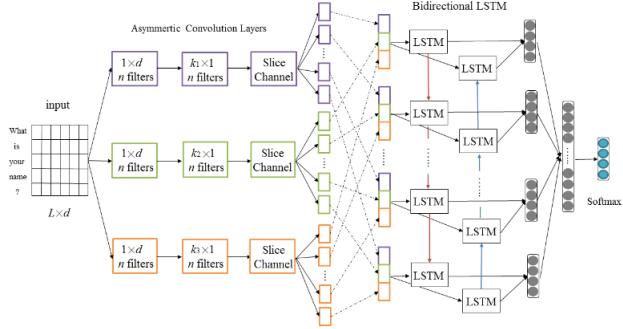


Figure 137: Architecture de AC-BLSTM

Une illustration de l'architecture du réseau est visible sur la Figure 137. Néanmoins, une particularité importante est à remarquer. Contrairement au réseau C-LSTM, AC-BLSTM accepte les kernels de tailles variables. Sans l'exploitation de padding, les *feature map* sont de dimensions variables et ne peuvent être concaténées. Pour résoudre cette problématique, une méthode a été proposée.

Supposons un filtre de dimension k_i appliqué sur une séquence de taille L , une convolution sans padding produira une *feature map* de dimension $D_i = L - k_i + 1$. Supposons maintenant un filtre de dimension k_j tel que $k_j > k_i$ alors, la dimension de la *feature map* sera de $D_j = L - k_j + 1$ avec $D_i > D_j$. L'approche standard consiste à **supprimer** les dimensions supérieures à $\min(\text{Card}(D_{k, \text{filters}})) = \hat{D} = D_j$. Cependant, elle est très destructrice.

Supposons c_i , *feature map* du filtre i. La méthode proposée consiste à extraire pour chaque *feature map* c, les valeurs c_i^t avec $t \geq L - \hat{D} + 1$ et d'appliquer une couche Full-Connected afin de produire un nouvel attribut qui remplacera les valeurs placées en entrée de la couche FC. Les dimensions de différentes *feature map* sont ainsi harmonisées. Un exemple illustratif de l'action réalisée est visible sur la Figure 138.

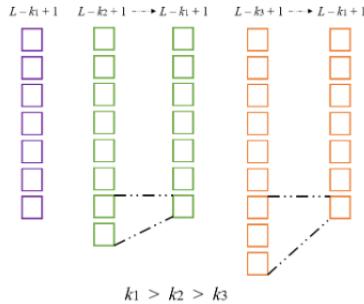


Figure 138: Harmonisation des dimensions des feature map selon l'approche de AC-BLSTM

14 Apprentissage par Renforcement et Deep Learning

Important: Ceci ne constitue pas un cours expliquant l'apprentissage par Renforcement. Bien que des notions soient introduites, de nombreux points importants seront ignorés car s'éloignant de la thématique du Deep Learning, de même que la rigueur mathématique²³² qui sera limitée pour faciliter la compréhension générale des idées explicitées.

Les notations peuvent sembler lourdes et les détails mathématiques difficiles à assimiler. Néanmoins, les mathématiques abordées relèvent essentiellement du jeu d'écriture et d'*astuces* à connaître.

14.1 Approche théorique

14.1.1 Généralités

L'*Apprentissage par Renforcement* est l'une des grandes familles d'apprentissage automatique au même titre que l'apprentissage supervisé et non supervisé par exemple. Cette approche se veut très proche de la méthodologie de l'apprentissage de l'homme basée sur l'expérience et l'analyse des conséquences suite à une action réalisée²³³. Bien que peu répandu en dehors de la Recherche, l'*apprentissage par renforcement* commence à se révéler grâce à sa capacité d'adaptation, sa faible dépendance à des données d'apprentissage pré-fournies et à ses performances. Il est très présent dans le cadre des systèmes autonomes (notamment les véhicules) et de la résolution de jeux (jeux de plateau tels que le Go ou les échecs, jeux vidéo).

²³²Nous ne traiterons pas les preuves de convergence des modèles par exemple alors que ça représente un aspect très important des algorithmes d'apprentissage !

²³³Un enfant apprend de ses chutes jusqu'à réussir à marcher

Les modèles d'*apprentissage par renforcement* reposent sur les interactions avec l'environnement et la réponse de cet environnement vis-à-vis de l'action choisie. Ainsi, la problématique peut être formalisée par un agent A, qui à l'instant t , au sein d'un environnement, doit choisir une action à réaliser, notée a_t selon l'état s_t dans lequel il se trouve. Après son action, l'environnement fournit une récompense r_t correspondant à la conséquence de cette action sur l'environnement. La récompense peut être positive ou négative selon si la réponse de l'environnement est bénéfique ou non. L'agent se trouve alors dans l'état s_{t+1} et choisit une nouvelle action à réaliser (ou s'arrête selon le cas). La Figure 139 résume cette problématique. L'objectif de cet apprentissage est donc de définir un modèle capable de maximiser les gains de récompenses.

14.1.1.1 L'environnement

L'environnement représente le milieu où évolue l'agent. Il peut présenter différentes spécificités:

- **Déterministe/Stochastique:** Lors d'une action a_t à partir d'un état s_t , l'agent arrive dans l'état s_{t+1} . L'environnement est défini par un espace d'états S fini ou non et d'un espace d'actions possibles A . Il est *déterministe* si:

$$\forall a_t, s_t \in (A_{s_t}, S), \exists s_{t+1} \in S, P(s_{t+1} | a_t, s_t) = 1$$

Sinon, il est considéré comme *stochastique*. Un environnement *déterministe* est défini par un environnement qui, pour tout stimuli de même nature, réagit de la même manière à partir d'un état donné. Par exemple, si on suppose le jeu Mario, lorsqu'on exécute l'action "sauter", Mario va sauter à chaque fois (si l'état à partir duquel l'action est faite le permet). Au contraire, un environnement est stochastique si:

$$\forall a_t, s_t \in (A_{s_t}, S), \exists s_{t+1} \in S, P(s_{t+1} | a_t, s_t) < 1$$

Cette particularité associe un facteur probabiliste à l'environnement. Ainsi, suite à une action et un état donnés, l'état suivant est déterminé selon une distribution probabiliste. Supposons une voiture, l'action d'accélérer peut réaliser l'accélération ou, dans d'autres cas, ne pas fonctionner en cas d'une panne par exemple.

- **Observable ou Partiellement Observable:** L'environnement est défini comme observable lorsque l'ensemble des états qui le forme sont pleinement définis et connaissables. Au contraire, l'environnement est défini comme partiellement observable lorsque les états ne sont pas ou partiellement connaissables et que l'acteur n'a accès qu'à des observations. Ainsi, un environnement partiellement observable est défini par une fonction de transitions entre les observations (qui définit la probabilité d'observer un phénomène à partir d'une action dans un état donné) et d'une fonction de perception liant observations et état (qui définit la probabilité d'avoir une observation dans un état). Ainsi, supposant, S , ensemble des états,

A , ensemble des actions et Z , ensemble des observations, la fonction de transition entre observations est définie par:

$$(s_t, a_t, z_{t+1}) \in (S, A_{s_t}, Z_{s_t}), T(s_t, a_t, z_{t+1}) = P(z_{t+1} | s_t, a_t)$$

La fonction de perception, noté φ , est définie par:

$$(s_t, z_t) \in (S, Z_{s_t}), \varphi(s, z) = P(z_t | s_t)$$

- **Discret/Continu:** Dans la configuration d'un environnement discret, le temps évolue de manière séquentielle. Ainsi, la fonction temps est définie par un pas discret soit de t à $t+1$. Dans le cas continue, le temps évolue de manière continue soit de t à $t+\Delta t$ avec Δt infiniment petit. De ce fait, un état n'évolue pas de manière discrète mais varie continuellement selon une quantité $\frac{ds}{dt}$ qui peut être comparée à une vitesse de variation d'état.
- **Connu/Inconnu:** L'environnement peut être inconnu de l'acteur. Ainsi, il peut ne pas connaître les récompenses obtenues, les fonctions de transitions voire même les états eux-mêmes. Au contraire, il peut aussi évoluer dans un environnement clairement établi et défini en amont. Par exemple, calculer l'itinéraire d'un trajet peut être associé à un environnement connu. Apprendre à un automate à marcher "par lui-même" est un problème à environnement inconnu car l'automate ne connaît pas l'impact de ses actions ni-même l'étendu des possibilités des différentes postures qu'il peut visiter durant son apprentissage.
- **Stationnaire/Non Stationnaire:** L'environnement peut varier au cours du temps. De ce fait, les récompenses et les fonctions de transitions peuvent ne pas être constantes. Un environnement variable implique un apprentissage permanente et évolutif de l'acteur. Si l'évolution de l'environnement suit un processus aléatoire, l'apprentissage ne peut être réalisé pleinement si ce n'est l'adaptation de l'acteur à son nouveau milieu. Si l'évolution est aléatoire et permanente, l'acteur ne peut apprendre. Comment peut-on jouer à un jeu dont les règles évoluent tout le temps de manière aléatoire ?
- **Fini/Infini:** L'environnement peut posséder un nombre fini d'états comme pour un jeu de plateau par exemple. Au contraire, il peut aussi avoir un nombre infini d'états. Un problème avec un nombre infini d'états est souvent associé à un problème à temps continu qui, du fait du nombre infiniment grand d'états, est jugé à environnement infini.

14.1.1.2 Formalisation du problème

Supposons un environnement stationnaire entièrement observable où le temps est discret. Le problème peut être formalisé par un **problème de décision de Markov** (PDM). Ce problème est défini par le quadruplet (S, A, P, R) défini tel que:

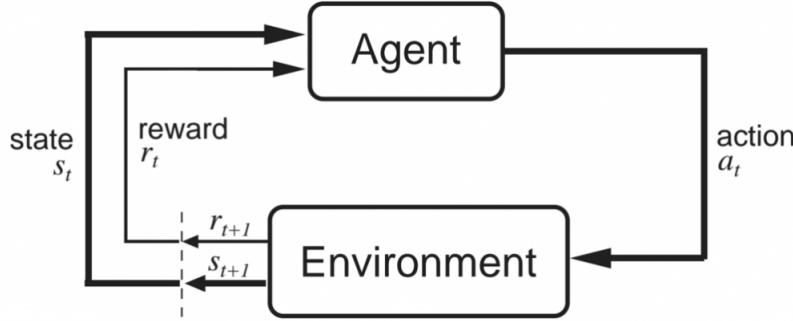


Figure 139: Graphique séquentiel de l'approche par renforcement

- **S:** ensemble d'états **finis**
- **A:** ensemble d'actions fini tel que $A(s)$ est associé à l'ensemble des actions réalisables à l'état s ²³⁴
- **T:** Fonction de transition qui explicite la dynamique de transition entre états pour une action donnée:

$$(s, a, s') \in (S, A, S) \rightarrow P(s, a, s') = Pr(s_{t+1} = s' | a_t = a, s_t = s)$$

- **R:** Fonction de récompenses sur \mathcal{R} qui explicite l'espérance (valeur moyenne) des récompenses renvoyées par l'environnement associées à chaque changement d'état. Elle est définie par:

$$(s, a, s') \in (S, A, S) \rightarrow R(s, a, s') = E(r_t | a_t = a, s_t = s, s_{t+1} = s')$$

Le problème est à *horizon fini* si il existe un (des) état(s) dits *final(aux)*, i.e provoque la fin de l'expérience. Par exemple, la fin d'une partie d'un jeu. Cette particularité permet ainsi la réalisation de plusieurs expériences distincts nommées *itérations* car le problème est à durée finie. Au contraire, le problème peut être à *horizon infini*. Il ne possède donc pas d'état(s) final(aux) et de ce fait, il n'y a qu'une expérience (à durée non finie). Par exemple, la survie d'un robot en mission²³⁵.

Important: On considère que l'état actuel ne dépend **QUE** de l'état précédent et de l'action qui y a été réalisée. Ainsi, la fonction de transition ne considère que l'action et l'état précédents pour choisir l'état actuel. Ceci est une hypothèse très forte qui suppose qu'une situation à l'instant t dépend exclusivement de la situation à l'instant $t-1$. Ceci est généralement faux dans les faits ! Cette

²³⁴Il se peut qu'un sous-ensemble uniquement de A soit réalisable dans un état s de S

²³⁵On peut, selon le point de vue, considérer la panne d'énergie ou l'usure du robot comme un état final si il implique son arrêt total

hypothèse est appelée l'**hypothèse de Markov**.

Lorsqu'on ne considère que la situation précédente pour choisir l'état actuel, on décrit un problème de Markov **d'ordre 1**. Si on considère la situation t-1 et t-2, nous obtenons un problème de Markov **d'ordre 2**. Par convention un PDM est défini si l'hypothèse de Markov d'ordre 1 est définie. Néanmoins, tout problème de Markov d'ordre n peut être ramené à un problème de Markov d'ordre 1 en augmentant le nombre d'états.

14.1.1.3 Définitions fondamentales

L'objectif de l'acteur est de maximiser²³⁶ les récompenses obtenues durant les interactions avec l'environnement. Notons r_t , la récompense obtenue par l'acteur à l'instant t. Le *Retour* définit la somme des récompenses obtenues par l'acteur à partir de l'instant t. Nous avons donc:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

Avec $\gamma \in [0, 1]$. Ce facteur est appelé *facteur de dépréciation*. Il permet de pondérer l'importance des récompenses obtenues selon la proximité temporelle. Si $\gamma = 1$, toutes les récompenses sont considérées avec la même intensité. Le système a donc un raisonnement à très long terme. Au contraire, si $\gamma \rightarrow 0$, le système ne considère que les récompenses proches et n'a pas de vision à long terme. Ce coefficient est très important en cas d'*horizon infini* car, étant donné l'absence d'état final, l'expérience n'a "pas de fin". Il est donc nécessaire de borner la somme par l'ajout de ce critère qui délimite la vision à long terme.

La *politique* représente la capacité de décision de l'agent. Elle définit l'action qui est choisie selon les critères qu'elle considère. Ainsi, nous la représentons telle que:

$$(s, a) \rightarrow (S, A), \pi(s, a) = P(a_t = a | s_t = s)$$

La politique est donc caractérisée par la probabilité de réaliser une action spécifique dans un état s de S. La définition précédente définit une politique *stochastique*. Selon les spécificités voulues pour le modèle, il est possible de désirer une politique *déterministe* où chaque état est associé à une action unique (probabilité de 1²³⁷). De ce fait, la politique suit la définition:

$$s \rightarrow S, \pi(s) = a$$

Un état possède une valeur qui représente sa qualité²³⁸. Cette valeur est liée à la politique et variable dès lors que la politique évolue. Elle est définie par

²³⁶Ou de minimiser selon la formulation de la problématique

²³⁷la politique choisit une action unique mais l'environnement peut être stochastique et faire en sorte que l'agent ne finisse pas nécessairement dans l'état choisi par la politique. Il faut bien différencier l'aspect stochastique de la politique et de l'environnement

²³⁸Par convention, plus la valeur est élevée, plus l'état est qualitatif donc présentant un intérêt à l'atteindre par l'agent

l'espérance des récompenses cumulatives à partir de l'état ciblé peu importe les actions réalisées par la suite. Ainsi, pour un état $s \in S$ et une politique π , nous avons:

$$V^\pi(s) = E(R_t | s_t = s)$$

Alors que V évalue la qualité d'un état peu importe les actions réalisées, Q définit une valeur d'un couple *état-action* pour une politique π donnée. Q évalue donc la qualité d'un état en considérant l'action qui a mené à cet état, au contraire de V qui l'ignore. Nous avons donc:

$$Q^\pi(s, a) = E(R_t | s_t = s, a_t = a)$$

14.1.1.4 L'équation fondatrice: l'équation de Bellman

Nous avons vu que:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

$$V^\pi(s) = E(R_t | s_t = s)$$

Par un jeu d'écriture, nous pouvons donc redéfinir V^π .

$$V^\pi(s) = E(R_t | s_t = s) \tag{11}$$

$$= E\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s\right) \tag{12}$$

$$= E(r_t + \sum_{k=1}^{\infty} \gamma^k r_{t+k} | s_t = s) \tag{13}$$

$$= E(r_t + \gamma \sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} | s_t = s) \tag{14}$$

$$= E(r_t) + \gamma E\left(\sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} | s_t = s\right) \tag{15}$$

Or, nous pouvons constater que:

$$V^\pi(s_{t+1}) = E\left(\sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} | s_t = s\right)$$

$E(r_t)$ présente une difficulté car l'espérance doit considérer toutes les situations possibles. Il est donc nécessaire de tenir de compte de l'aspect *stochastique* de la fonction de transition et de la politique. Dans une configuration *déterministe*, l'équation en sera qu'un cas particulier. Supposons $R(s_t, a_t, s_{t+1})^{239}$, espérance de la récompense obtenue en passant de l'état s_t à s_{t+1} par l'action a_t . Dans un premier temps, nous considérerons l'environnement et la politique comme *déterministes*. Ainsi, chaque action garantit d'atteindre l'état-cible et la politique choisit toujours la même action pour un état donné. Nous obtenons donc:

$$E(r_t) = R(s_t, a_t, s_{t+1})$$

²³⁹Valeur issue de la fonction récompense définie par le PDM

Supposons l'environnement comme stochastique. L'état-cible n'est donc plus garanti d'être atteint mais relève d'une considération probabiliste. De ce fait, la valeur de récompense associée à l'état actuel repose sur la somme des différentes valeur de récompenses obtenues en atteignant les différents états de l'environnement possibles. Nous avons donc:

$$E(r_t) = \sum_{s_{t+1} \in S} P(s_t, a_t, s_{t+1}) R(s_t, a_t, s_{t+1})$$

Avec la même approche, supposons une politique et un environnement stochastiques. Nous obtenons alors:

$$E(r_t) = \sum_{a_t \in A(s)} \pi(s_t, a_t) \sum_{s_{t+1} \in S} P(s_t, a_t, s_{t+1}) R(s_t, a_t, s_{t+1})$$

Par un calcul analogue, nous pouvons donc redéfinir $V^\pi(x)$. Cette équation, nommée **équation de Bellman**, est au coeur de la théorie de l'apprentissage par renforcement et constitue le socle théorique fondamental de cette famille d'algorithme. Elle est définie par:

$$V^\pi(s_t) = \sum_{a_t \in A(s)} \pi(s_t, a_t) \sum_{s_{t+1} \in S} P(s_t, a_t, s_{t+1}) (R(s_t, a_t, s_{t+1}) + \gamma V^\pi(s_{t+1}))$$

L'objectif du modèle est d'obtenir la politique optimale qui maximise les valeurs d'états. Nous noterons la fonction valeur de la politique optimale, $V^*(s)$. Elle constitue l'**équation d'optimalité de Bellman**.

$$V^*(s) = \max_\pi V^\pi(s) = \max_{a \in A(s)} \sum_{s_{t+1} \in S} P(s_t, a_t, s_{t+1}) R(s_t, a_t, s_{t+1})$$

14.1.1.5 Model-Free et Model-Based

En connaissant l'ensemble des éléments du MDP, il est "aisé" de définir une solution avant d'interagir avec l'environnement. On peut donc considérer ce problème comme un problème de *planification* usant d'algorithmes de *programmation dynamique*²⁴⁰. Dans les faits, l'environnement est rarement parfaitement connu, i.e la fonction de récompenses et de transitions sont inconnues. L'agent doit donc être capable d'évoluer sans connaissance préalables de l'environnement où il se trouve. Pour cela, il existe deux approches d'apprentissage: **Model-Free** et **Model-Based**.

Model-Based repose sur l'idée que l'agent doit apprendre un modèle qui approxime les caractéristiques de l'environnement pour définir une politique optimale. A partir de son approximation, il est capable de définir le comportement optimal à suivre pour réaliser son objectif. Ainsi, si l'agent est dans l'état s_t , qu'il réalise l'action a_t et qu'il atteint l'état s_{t+1} avec une récompense r_{t+1} , alors

²⁴⁰Nous ne détaillerons pas ces algorithmes dans ce cours

le modèle cherchera à améliorer son estimation de $P(s_{t+1}|s_t, a_t)$ et de $R(s_t, a_t)$ à travers les valeurs de V. Le modèle apprendra donc les conséquences de la réalisation d'une action dans un état donné. A partir des conséquences connues, le modèle définira la politique la plus performante.

Model-Free s'émancipe de l'approximation du modèle pour définir une politique. Il n'y a donc pas d'approximation de l'environnement, i.e de la fonction de transitions et de récompenses. Le modèle cherche donc à apprendre directement la politique à suivre, i.e quand choisir telle action. L'algorithme *Q-Learning*[118] est l'algorithme de référence pour ce type d'approche.

14.1.1.6 Différence temporelle

Une des principales faiblesses des premières approches par apprentissage par renforcement (sans connaissances explicites du PDM) était la nécessité de devoir finir une itération pour mettre à jour le modèle (par exemple, la méthode de Monte-Carlo). Ce type d'approche demande du temps et surtout, est inefficace dans le cadre d'un environnement à *horizon infini*²⁴¹²⁴². Afin de palier à cette faiblesse, la notion de *différence temporelle* a été introduite par Sutton dans son article de recherche [107].

Nous avons vu que pour une politique π donnée, nous avons:

$$V^\pi(s_t) = r_t + \gamma V^\pi(s_{t+1})$$

Par un simple jeu d'écriture, nous obtenons:

$$\underbrace{r_t + \gamma V^\pi(s_{t+1})}_{\text{Temporal Difference}} - V^\pi(s_t) = 0$$

Durant l'apprentissage, cette égalité n'est pas vérifiée. Si la valeur actuelle de $V^\pi(s_t)$ est trop élevée, la valeur TD sera négative et positive dans le cas contraire. Cette valeur permet donc d'orienter l'évolution de $V^\pi(s_t)$. La correction se présente sous la forme:

$$V_{t+1}^\pi(s_t) \rightarrow V_t^\pi(s_t) + \alpha(r_t + \gamma V_t^\pi(s_{t+1}) - V_t^\pi(s_t))$$

Avec $\alpha \in [0, 1]$. α est associé à un taux d'apprentissage. Il est pertinent pour se protéger de la condition de stochasticité de l'environnement. En effet, si il est déterministe, nous sommes "sûr" de la modification à réaliser sur V. De ce fait, nous pouvons exploiter un taux égal à 1. Au contraire, si l'environnement est stochastique, les modifications sont incertaines car dépendantes de la distribution des états. Initialement, nous considérerons une valeur proche de 1 car la distribution est inconnue. Cela permet des mises à jour grossières et

²⁴¹Un environnement sans état final donc sans "fin" d'expérience explicite

²⁴²Ceci n'est pas complètement vrai. Il existe des "astuces" pour ignorer ce type de contraintes mais nous ne les considérerons pas

```

Input: the policy  $\pi$  to be evaluated
Initialize  $V(s)$  arbitrarily (e.g.,  $V(s) = 0, \forall s \in \mathcal{S}^+$ )
Repeat (for each episode):
    Initialize  $S$ 
    Repeat (for each step of episode):
         $A \leftarrow$  action given by  $\pi$  for  $S$ 
        Take action  $A$ ; observe reward,  $R$ , and next state,  $S'$ 
         $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$ 
         $S \leftarrow S'$ 
    until  $S$  is terminal

```

Figure 140: Algorithme du TD(0)

importantes (souvent très oscillantes) le temps d'estimer les distributions convenablement. Avec le temps, le taux tendra vers 0 car les distributions seront bien estimées via les valeurs de V et pour permettre une bonne convergence du modèle. Pour une approche par défaut, il est commun de définir α comme une fonction dépendante de l'état considérée tel que:

$$\alpha(s) = \frac{1}{1 + \text{nombre de visite de } s}$$

Temporal difference est utilisée au sein d'un algorithme nommé TD(0) qui exploite sa capacité de correction pour évaluer les valeurs d'états pour une politique donnée (model-based). Il est défini sur la Figure 140.

14.1.1.7 Un standard: l'algorithme Q-Learning

TD(0) évalue une politique mais ne permet pas de la mettre à jour. Il est donc nécessaire de proposer une autre méthode capable de proposer une politique améliorée. Il est possible d'exploiter des méthodes "standards"²⁴³ pour exploiter le résultat de TD(0) ou d'utiliser un autre algorithme qui permet d'apprendre une politique de manière autonome comme l'algorithme *Q-Learning*.

Cette algorithme repose sur la même approche que TD(0) sauf qu'il exploite les valeurs Q et non V . De ce fait, il considère les couples(état,action) et non juste les états. Son comportement est donc spécifique aux actions et non cumulatif comme le fait TD(0). L'algorithme est détaillé sur la Figure 141.

Deux spécificités importantes sont à approfondir. La première est l'utilisation de $\max(Q(s_{t+1}, a_{t+1}))$. Par conséquent, le modèle exploitera toujours la valeur de Q associée à la meilleure action à faire dans l'état $t+1$, indépendamment de l'action qui sera véritablement réalisée (la meilleure action n'est pas forcément celle qui est réalisée !) dans cet état. On caractérise les algorithmes usant de ce type d'approche comme des algorithmes **off-policy**. Un autre algorithme, nommé *SARSA*[106], exploite la même approche que *Q-Learning* à la différence que *SARSA* n'exploite pas la valeur max mais la valeur effectivement choisie

²⁴³non traitées dans ce cours

par le modèle. C'est ainsi un algorithme dit **on-policy**.

La différence effective entre les deux algorithmes est "l'audace" de l'agent. En effet, Q-Learning n'exploite que les meilleures valeurs indépendamment des actions réelles effectuées. De ce fait, Q-Learning ne tient pas compte de situations potentiellement néfastes pour l'agent car la mise à jour des valeurs de Q ne considère pas l'action réellement effectuée. Au contraire, SARSA considère l'ensemble des situations que l'agent visite et se met à jour en adéquation avec les situations "dangereuses" observées. Ce modèle sera donc beaucoup plus prévenant et moins audacieux qu'un agent sous Q-Learning. Pour imaginer, supposons qu'un individu souhaite aller d'un point A à un point B où ces deux points sont au bord d'une falaise. L'agent sous Q-Learning se déplacera le long du bord jusqu'à arriver à destination. En effet, c'est le chemin le "plus court" et efficace. Cependant, il ne tient pas compte du risque (peu probable mais réel) de chute possible. Au contraire, l'agent sous SARSA s'éloignera légèrement du bord, augmentant la distance parcourue mais évitant le piège de la chute. Q-Learning favorise l'efficacité par rapport au risque alors que SARSA réalise un compromis entre les deux. Il n'y a pas de meilleure approche. Tout dépend de la problématique à traiter et du contexte de l'agent.

La seconde spécificité à comprendre est la politique choisie par Q-Learning (ou pour SARSA). Avant de présenter ces politiques, il est nécessaire d'introduire la notion de **compromis exploration/exploitation**.

Pour obtenir une bonne compréhension de l'environnement, il est nécessaire de l'explorer et de le visiter. Cette étape est appelée **exploration**. Lorsque l'environnement est assimilé et connu, il n'est plus pertinent de visiter mais au contraire, il est nécessaire d'exploiter la connaissance obtenue. Cette étape est nommée **exploitation**. Un compromis entre ces deux comportements est au cœur de l'apprentissage par renforcement car c'est l'exploration qui permet de détecter les comportements les plus bénéfiques mais uniquement l'exploitation est capable de les exploiter. La difficulté est donc de comprendre à quel moment l'agent a suffisamment exploré pour pouvoir exploiter ses connaissances.

Différentes politiques peuvent être exploitées dont les plus répandues sont:

- Approche indépendante de la fonction Q:
 - **Approche gloutonne:** Cette approche est la plus élitiste possible et consiste à exploiter les actions associées aux meilleures valeurs Q dès le début. Elle est définie par: $a_{st,glout} = \arg(\max(Q(s_t, a)))$. Cette méthode est peu exploitable car peu efficace. Dès lors qu'un "chemin" aura été découvert, le modèle l'exploitera sans considérer les autres alternatives. Il est donc probable que le modèle n'exploite qu'un minimum local...
 - **Approche ϵ -gloutonne:** Cette approche corrige la faiblesse de la méthode gloutonne en limitant son élitisme. Ainsi, l'agent réalisera

une action gloutonne avec une probabilité ϵ sinon, il réalisera une action aléatoire. la valeur de ϵ varie au cours du temps pour s'adapter à la situation. Ainsi, au début, ϵ sera très faible pour favoriser l'exploration et augmentera progressivement pour finir par exploiter ses connaissances. Cette méthode permet l'étape d'exploration et présente des résultats convenables.

- Approche dépendante de la fonction Q :

- **Approche Softmax:** Cette politique repose sur la fonction Softmax. La probabilité de réaliser une action dans un état donné est proportionnelle à sa valeur par rapport à celles des autres actions de ce même état. Ainsi:

$$P(a_t|s_t) = \frac{Q(s_t, a_t)}{\sum Q(s_t, a_t)}$$

- **Approche Boltzmann:** Cette politique repose sur la distribution de Boltzmann. Elle peut être vue comme un cas particulier de la fonction Softmax. Son intérêt repose sur sa capacité d'adaptation durant l'apprentissage au contraire de Softmax qui est fixé. Elle est définie selon:

$$P(a_t|s_t) = \frac{e^{\frac{Q(s_t, a_t)}{\tau}}}{\sum e^{\frac{Q(s_t, a_t)}{\tau}}}$$

Avec $\tau \in \mathcal{N}_+$. Si $\tau \rightarrow 0$, le comportement de la politique tend à s'approcher de la politique aléatoire. Au contraire, si τ augmente, elle s'approche de la méthode gloutonne. Il est nécessaire de faire varier τ au fil de l'apprentissage afin de favoriser l'exploration en début d'apprentissage et progressivement tendre vers l'exploitation.

Remarque: Les algorithmes étudiés jusqu'alors possèdent "(0)" dans leurs noms. Cela signifie qu'il n'y a pas de considération des **traces d'éligibilité**. En effet, jusqu'alors, nous avons considéré qu'une récompense est due exclusivement à la dernière transition effectuée. Cette hypothèse est fausse car l'état $t-2$, $t-3\dots$ ont aussi une importance sur la situation à l'instant t , ce qui est ignoré par un algorithme de type (0)²⁴⁴. L'idée des *traces d'éligibilité* est de propager la récompense aux transitions précédentes aussi car elles sont actrices de la situation actuelle. Cette éligibilité est quantifiée selon un niveau d'importance représenté par la proximité de la transition avec l'état actuel considéré. Cette approche est caractérisée par l'attribut (λ) ($Q(\lambda)$, $TD(\lambda)\dots$). Nous ne détaillerons pas ces algorithmes mais il est important de connaître leur existence.

²⁴⁴On peut grossièrement faire une analogie avec l'hypothèse de Markov !

```

Initialize  $Q(s, a)$  arbitrarily
Repeat (for each episode):
    Initialize  $s$ 
    Repeat (for each step of episode):
        Choose  $a$  from  $s$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)
        Take action  $a$ , observe  $r, s'$ 
         $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ 
         $s \leftarrow s'$ ;
    until  $s$  is terminal

```

Figure 141: Algorithme du Q-Learning(0)

14.1.1.8 Approximation des valeurs d'états

Jusqu'ici, nous avons étudié des méthodes qui reposent sur un nombre de visites important d'un même état²⁴⁵. Cette condition n'est pas tenable en cas d'espace d'états de très grande dimension (ou infini) car les capacités calculatoires ne le permettent pas. Une approche consiste donc à ne plus évaluer chaque valeur d'état possible mais à approximer une fonction capable de prédire n'importe quel état. Cette approche est très puissante car elle permet de prédire n'importe quel état, y compris des états jamais visités. Pour réaliser cela, il est nécessaire d'exploiter une méthode capable d'approximer des fonctions. Il est reconnu que les réseaux de neurones brillent dans cette tâche d'où leurs utilisations.

L'idée est simple. Un réseau reçoit une entrée associée à un état et la sortie du réseau produit une estimation de la valeur de cet état. En général, un état est caractérisé par différents attributs représentant l'entrée du réseau. Par exemple, dans le cas d'une voiture qui roule, on peut supposer les caractéristiques associées à la vitesse, l'état de la voiture, l'état de la route etc... Le réseau produit ainsi une valeur $\hat{V}(s) = f(s, \theta)$ (ou $\hat{Q}(s)$) avec θ , poids du réseau (paramètres) qui pondèrent les caractéristiques de l'état observé. L'objectif du réseau est donc de minimiser l'erreur $\|f(s, \theta) - V^*(s)\|$ en trouvant la combinaison θ^* optimale. L'utilisation des réseaux de neurones soulèvent de nombreuses problématiques notamment l'apprentissage *online* du réseau, l'exploitation de différences temporelles et non d'une valeur labellisée au pouvoir explicatif juste et nette. Il existe de nombreuses approches qui dépassent le cadre de ce cours mais qu'il est important de considérer si vous voulez pleinement maîtriser ce domaine.

14.2 Deep Q-Learning

Avec l'avènement du Deep Learning, les algorithmes de renforcement reposent de plus en plus sur des structures neuronales. Dans le cadre de ce cours, nous exploiterons l'application des réseaux convolutifs pour la création de *bots* pour des jeux vidéos de type *ATARI*²⁴⁶. Ce n'est qu'un cas particulier de ce qui est

²⁴⁵la convergence "parfaite" vers V^* et Q^* demande une visite infinie de chaque état par exemple...

²⁴⁶Atari est une entreprise française (initialement américaine) de jeu vidéo fondée en 1972

possible de réaliser. Ce cours ne fournit qu'une sensibilisation à ce domaine à travers cet exemple mais un travail de recherche personnel sera nécessaire pour réaliser un état de l'art complet.

Un jeu, en apparence simple, peut présenter une complexité sous-jacente notamment liée aux différentes possibilités d'action réalisable dans le cadre d'une partie. Par exemple, le nombre de Shannon, 10^{120} , est une estimation de la complexité du jeu d'échecs, c'est-à-dire du nombre de parties différentes réalisables dans le cadre d'une partie standard²⁴⁷. Les approches standards de l'apprentissage par renforcement reposent sur un apprentissage itératif basé sur une visite de nombreuses fois d'un même état de l'environnement. Il est évident qu'étant donné la dimension massive de l'espace d'états, ces approches ne sont pas humainement réalisables. Il est donc nécessaire d'étudier une approche plus généraliste capable d'étudier des problèmes à haute dimension.

Le Deep Q-learning[77] exploite les capacités des réseaux neuronaux pour obtenir une approximation de la fonction Q qui définit la valeur de $Q(s,a)$ dans un état s réalisant l'action a. Le réseau neuronal est ainsi capable d'évaluer chaque valeur $Q(s,a)$ même si l'état s est peu ou pas visité (de même pour l'action réalisable dans cet état). Cette approche permet donc de résoudre la problématique de l'espace d'états à haute dimension.

Un robot-joueur possède les mêmes informations qu'un joueur humain. Ainsi, dans le cadre d'un jeu Atari, les informations à disposition sont les images de jeu²⁴⁸(en plus des récompenses associées à l'environnement telles que l'impact sur le score par exemple). Il est donc nécessaire d'adapter le réseau de neurones à cette spécificité d'où l'utilisation d'un réseau convolutif qui présente de bons résultats en analyse d'image.

Les données d'apprentissage du réseau correspondent à des tuples de données de la forme $< s_t, a_t, r_t, s_{t+1} >$ où s_t correspond à un ensemble de N images successives (N=4 dans le cadre des jeux Atari), a_t est l'action réalisée dans l'état s_t , r_t est la récompense obtenue et s_{t+1} , l'état de l'automate après son action représenté par N images. Ces tuples présentent deux risques majeurs. Le premier est la corrélation entre les données. Il est évident que l'image à l'instant t est liée à l'image à l'instant t+1 car c'est dans la continuité du jeu. Le second correspond à la distribution du jeu de données. Supposons le jeu du casse-briques, il est possible que l'automate, du fait de son initialisation aléatoire, favorise un des deux côtés de l'écran. Ceci est problématique car si non traitée, cette particularité peut inhiber l'exploitation d'une zone complète du jeu en favorisant le sur-apprentissage du réseau convolutif. Il est donc nécessaire de lutter contre la corrélation du jeu d'apprentissage et de permettre qu'il soit

²⁴⁷Ce nombre est à dissocier du nombre total de partie réalisable qui est nettement plus élevé

²⁴⁸Dans le cadre d'autres jeux, comme le Cartpole par exemple, l'information à disposition peut être présentée sous forme de données numériques telles que la vitesse, l'angle par rapport à la verticale etc...

le plus représentatif de la distribution réelle du jeu observé.

Ces tuples sont obtenus au cours des expériences finies du jeu et sont conservés au sein du *replay memory*, jeu d'apprentissage de K tuples où K élevé pour limiter les variations de distribution du jeu d'apprentissage associées aux expériences possiblement très variées de l'automate vis-à-vis du jeu. L'idée est que le comportement aléatoire initial de l'automate va permettre une exploration satisfaisante et la dimension élevée du jeu d'apprentissage permettra de limiter le risque de sur-spécialisation des images conservées. L'apprentissage du réseau de neurones sera réalisé avec un minibatch de k tuples tirés au hasard dans le *replay memory* afin de limiter la corrélation entre les tuples qui est dangereuse pour l'apprentissage du réseau. Cette approche constitue l'*Experience Replay*.

Dans le cadre du Q-learning, nous pouvons définir:

$$Q_{i+1}(s_t, a_t) = E[r_t + \gamma * \max_{a_{t+1} \in A_{s_{t+1}}} Q_i(s_{t+1}, a_{t+1}) | s_t, a_t]$$

Ainsi, l'objectif du réseau est d'approximer une fonction qui, pour tout s et a, minimise la fonction de perte suivante (de type *Squared Error Loss*):

$$Loss = \frac{1}{2} \cdot \underbrace{[r_t + \gamma * \max_{a_{t+1}} (Q(s_{t+1}, a_{t+1}; \theta))]}_{\text{target}} - \underbrace{Q(s_t, a_t; \theta)}_{\text{prediction}}]^2$$

L'architecture du réseau est visible sur la Figure 142. Cette architecture souffre des limitations théoriques des réseaux de neurones. Ainsi, il n'y a pas de convergence théorique démontrée bien qu'empiriquement stable. De plus, la distribution des scores de l'automate sont très bruitées et témoignent d'une variance importante. Cette variance témoigne du manque de robustesse du modèle qui présente une trop grande sensibilité aux variations des observations. De plus, on peut noter qu'une variation de faible ampleur des poids du réseau agit de manière significative sur les performances de l'automate (en bien comme en mal). Néanmoins, l'évolution des Q-values tend à converger. Cette dernière observation est contestable car la diminution des taux d'apprentissage simule ce comportement sans véritablement corriger les risques de divergence réels. Ca soigne les symptômes sans s'attaquer aux causes en quelque sorte... Néanmoins, les performances du modèle dépasse ses concurrents (de l'époque !) et dépasse parfois, l'homme lui-même. Cet algorithme présente, cependant, une faiblesse sur des jeux qui demandent une stratégie à long terme. La notion de mémoire est donc à améliorer sur cette algorithme.

14.3 Prioritized Experience Replay

Cette approche propose un postulat où des expériences sont porteuses de plus d'informations que d'autres. La sélection des expériences stockées dans l'Experience Replay étant aléatoire, il est probable que les expériences "riches" pour l'apprentissage

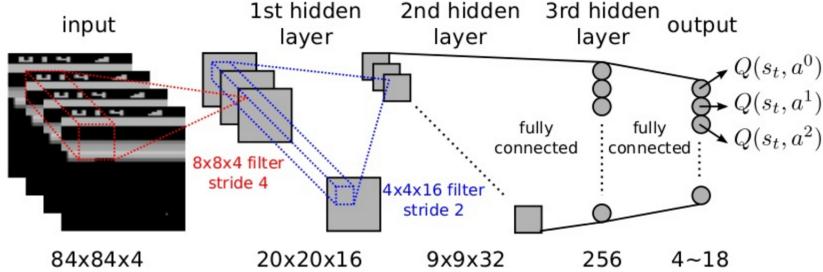


Figure 142: Algorithme du Deep Q-Learning

se voient noyées dans l'ensemble des expériences pour la majorité peu riches. Afin de contrer ce problème, Prioritized Experience Replay[95] propose une approche pour pondérer les expériences afin d'exploiter au mieux les expériences porteuses d'informations.

Pour juger de la pertinence d'une expérience, un indice de priorité est proposé. Il est défini par: $p_i = |\delta_i| + \epsilon$ avec i , index de l'i-ième expériences, δ , erreur entre la prédiction réalisée par le modèle et la valeur effective à obtenir et ϵ , constante non nulle pour éviter qu'une expérience ait un indice de priorité nul.

Utiliser une approche *greedy* sur les indices de priorité (sélectionner que les expériences avec un fort indice) va imposer une sur-représentation de certaines expériences au détriment d'autres et de ce fait, favoriser l'overfitting du modèle. Pour limiter ce phénomène, un compromis entre une approche *greedy* et une sélection aléatoire est nécessaire. L'approche *Softmax* a donc été choisie pour réaliser ce compromis. Ainsi, à chaque expérience, nous associons une probabilité d'être sélectionnée P_i définie telle que $P_i = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$ avec p_i , indice de priorité de l'expérience i , k , nombre d'expériences et α , constante qui définit l'importance de l'indice dans la détermination de la probabilité de sélection. Si α est égale à 0, nous obtenons une sélection aléatoire et si $\alpha \rightarrow \infty$, la sélection est *greedy*.

Afin de limiter l'impact des *outliers* (une expérience avec une erreur significativement plus grande que les autres)²⁴⁹, il peut être nécessaire d'utiliser une autre métrique pour calculer l'indice de priorité. Ainsi, au lieu d'exploiter l'erreur de prédiction, l'indice reposera sur le *rang* de cette erreur. Ainsi, l'erreur la plus importante aura un rang de 1 et l'erreur la moins importante, un rang de k (en supposant k expériences). L'indice de priorité est donc défini par: $\frac{1}{rank(i)}$. Cette approche tend à être plus robuste grâce à son insensibilité aux outliers mais les deux favorisent une vitesse de convergence significativement plus forte

²⁴⁹Supposons un cas extrême où une expérience est infiniment plus grande que les autres. Cette expérience aura une probabilité quasi absolue d'être sélectionnée à chaque fois.

qu'avec une approche aléatoire.

Prioritized Experience Replay introduit un biais du fait de sa sélection non aléatoire. La sélection prioritaire de certaines expériences va forcer le modèle à apprendre de nombreuses fois sur un sous-ensemble d'expériences alors que d'autres seront peu ou pas examinées. Ce phénomène risque de provoquer un overfitting important à terme. Afin de corriger cette faiblesse, il est nécessaire de limiter l'impact d'une expérience souvent exploitée.

Pour cela, la valeur de l'erreur du modèle associée à l'expérience i est redéfinie par $w_i\delta_i$ avec $w_i = (\frac{1}{N} * \frac{1}{P_i})^\beta$ où N , taille du batch d'apprentissage du modèle et β , facteur d'importance permettant de définir l'impact de la régulation sur l'erreur produite par l'expérience. Une expérience souvent observée aura donc une influence régulière dans l'apprentissage du modèle mais son impact sera plus faible que celle des expériences plus rarement exploitées. Au début de l'apprentissage, on exploite une valeur de β proche de 0 car, à cette étape, la présence d'un biais n'est pas trop impactant. Lorsque le modèle commence à converger, réguler le biais est nécessaire et de ce fait, la valeur de β doit tendre vers 1. Ainsi, il sera pertinent, durant l'apprentissage, de faire varier continuellement β de 0 vers 1 afin d'exploiter le pouvoir de converge d'une approche *greedy* et la protection contre l'overfitting d'une approche stochastique.

14.4 Double Deep Q-Learning

Dans l'algorithme du Q-Learning, l'opérateur max utilise la même valeur pour choisir une action et l'évaluer. Cette particularité va favoriser la sélection de valeurs surestimées suite à un excès d'optimisme dû aux erreurs d'apprentissage. En effet, supposons un état où chaque Q doit avoir une valeur identique lors d'une convergence idéale du modèle. Durant l'apprentissage, la présence de l'opérateur max dans l'algorithme du Q-Learning va introduire un biais associé aux valeurs de Q encore bruitées selon les expériences de l'apprentissage. En effet, la valeur de Q avec la plus grande erreur positive sera sélectionnée. Ce phénomène est appelé *Overoptimism*. L'idée du Double Deep Q learning (DDQN)[32] est d'exploiter deux réseaux distincts pour évaluer une action et choisir l'action à réaliser. Cette approche permet ainsi de limiter l'optimisme de l'algorithme et d'être plus "modéré" sur les changements de comportements en limitant le biais de prédiction de $\max_a(Q(s,a,w))$. Cette approche est très utile en début d'apprentissage car le manque d'informations fausse la pertinence des valeurs de Q .

Supposons deux réseaux dont l'ensemble des poids est représenté par w_1 et w_2 . L'estimation de l'erreur par le DDQN devient donc:

$$Loss = \frac{1}{2} \cdot (r_t + \gamma Q(s_{t+1}, \textcolor{red}{argmax}_{a_{t+1}} Q(s_{t+1}, a_{t+1}, w_1), w_2) - Q(s_t, a_t; w_1))^2$$

Les deux modèles sont appris indépendamment et le réseau mis à jour choisi

aléatoirement. Il est possible d'alterner le réseau qui prédit l'action à réaliser et sa valeur pour un apprentissage symétrique.

Pour des raisons de temps d'apprentissage et de coût matériel, cette approche n'est pas exploitée comme présentée précédemment. Une méthode subsidiaire permet de profiter de la majorité des bénéfices de l'approche *Double* tout en limitant les exigences de calculs. Pour cela, le second modèle n'est pas complètement dissocié du premier mais est équivalent à une copie du modèle initial dont les poids correspondent aux poids du modèle initial à un instant t-n. La prédiction de la valeur de Q sera réalisée avec le modèle ancien et la prédiction de l'action à réaliser, par le réseau à jour. Cette particularité permet de s'émanciper d'un double apprentissage qui est gourmand en ressources matérielles et temporelles en se limitant à l'apprentissage d'un unique réseau.

Supposons 2 Q-network: un "à jour" (w), un "ancien" (w^-). (w) sélectionne l'action à réaliser et (w^-) évalue la valeur de l'action. L'apprentissage du réseau suit la relation suivante:

$$Loss = \frac{1}{2} \cdot (r_t + \gamma Q(s_{t+1}, \text{argmax}_{a_{t+1}} Q(s_{t+1}, a_{t+1}, w), w^-) - Q(s_t, a_t; w))^2$$

14.5 Dueling Deep Q-Learning

Le modèle Dueling (DDQN)[117] repose sur l'idée que la valeur de Q est une combinaison de deux fonctions: A(a,s) qui représente la pertinence de faire une action a dans un état s et V(s), la pertinence d'être dans un état S²⁵⁰. Cette séparation permet de considérer la qualité d'être dans un état indépendamment d'une action. Elle permet ainsi de mieux évaluer un état en l'émancipant d'une action associée. En effet, la plupart du temps, la valeur associée à la réalisation d'une action dans un état s n'a pas d'influence significative sur la valeur de l'état s. Pour considérer cette particularité, V et A sont calculées à partir de deux réseaux feed-forward issus d'une même source (couche de convolution ou un autre réseau feed-forward) et parfaitement isolés l'un de l'autre. Leurs sorties sont par la suite additionnées pour produire la valeur de Q.

Nous avons donc:

$$Q(s, a) = V(s) + A(s, a)$$

$$V(s) = E(Q(s, a))$$

$$A(s, a) = Q(s, a) - E(Q(s, a))$$

$$E(A(s, a)) = 0$$

$$Q(s, a) = V(s) \text{ and } A(s, a) = 0 \text{ if deterministic policy}$$

²⁵⁰Cette combinaison peut laisser penser à une approche model-based qui vise à étudier les spécificités de l'environnement

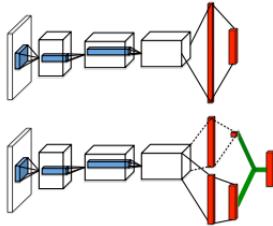


Figure 143: Algorithme du Dueling Deep Q-Learning

Dans les faits, nous exploitons une valeur de A normalisée pour l'apprentissage soit:

$$Q(s, a, \theta, \alpha, \beta) = V(s, \theta, \beta) + (A(s, a, \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} A(s, a, \theta, \alpha))$$

Cette régularisation permet d'assurer que l'un des flux du réseau Dueling déterminera bien la valeur d'un état. Il est aussi possible de régulariser par la valeur max de $A(s, a, \theta, \alpha)$ mais l'utilisation de la moyenne présente de meilleurs résultats.

Le réseau est illustré sur la Figure 143. Expérimentalement, il présente de meilleurs résultats que le DQN et le DDQN.

14.6 Double Dueling Q-Learning

Le modèle Double Dueling Q-Learning (D-DDQN) est une approche qui unit le Double Q-Learning et le Dueling Q-Learning. Ainsi, DDQN exploitera deux réseaux distincts pour réaliser la prédiction de la valeur de Q et de l'action à réaliser (approche Double) et, au sein de ces réseaux, la prédiction de la valeur d'état et de la pertinence d'une action sera dissociée (approche Dueling).

Son efficacité tend à dépasser l'approche Double ou Dueling seule en profitant des bénéfices des deux approches.

15 Ajouts à venir

- 15.1 Application à la reconnaissance vocale**
- 15.2 Application au traitement du langage écrit**
 - 15.2.1 Traduction - Neural Machine Translation**
 - 15.2.2 Recherche d'informations**
 - 15.2.3 Désambiguïsation d'entités**
- 15.3 L'analyse d'image et ses formes**
 - 15.3.1 Segmentation: méthodes State-of-the-art**
 - 15.3.2 Object Tracking: méthodes State-of-the-art**
- 15.4 Machine Learning et Éthique**
 - 15.4.1 adversarial-examples**
- 15.5 Deep Learning Bayésien**
- 15.6 Neuroevolution**
- 15.7 Machine de Turing neuronale**
- 15.8 Geometric Deep Learning**
- 15.9 Neural Architecture Search (NAS)**
- 15.10 Transfer Learning**
 - 15.10.1 Domain Adaptation**
- 15.11 Few Shot Learning**

References

- [1] Hierarchical variational autoencoders for music. 2017.
- [2] Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. *CoRR*, abs/1606.04474, 2016.
- [3] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. *ArXiv e-prints*, January 2017.
- [4] Quoc V. Le Ilya Sutskever Lukasz Kaiser Karol Kurach James Martens Arvind Neelakantan, Luke Vilnis. Adding gradient noise improves learning for very deep networks. *arXiv*, novembre 2015.
- [5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.
- [6] Dana H. Ballard. Modular learning in neural networks. pages 279–284, 1987.
- [7] Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Neural optimizer search with reinforcement learning. *CoRR*, abs/1709.07417, 2017.
- [8] Mateusz Buda, Atsuto Maki, and Maciej A. Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *CoRR*, abs/1710.05381, 2017.
- [9] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *CoRR*, abs/1605.07678, 2016.
- [10] G. Chen, D. Ye, Z. Xing, J. Chen, and E. Cambria. Ensemble application of convolutional and recurrent neural networks for multi-label text categorization. pages 2377–2383, May 2017.
- [11] KyungHyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014.
- [12] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.
- [13] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015.

- [14] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann LeCun. Very deep convolutional networks for natural language processing. *CoRR*, abs/1606.01781, 2016.
- [15] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.
- [16] Carl Doersch. Tutorial on variational autoencoders. *CoRR*, abs/1606.05908, 2016.
- [17] Timothy Dozat. Incorporating nesterov momentum into. 2015.
- [18] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. (UCB/EECS-2010-24), Mar 2010.
- [19] Jeffrey L. Elman. Finding structure in time. *COGNITIVE SCIENCE*, 14(2):179–211, 1990.
- [20] Mark Everingham, S. M. Eslami, Luc Gool, Christopher K. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *Int. J. Comput. Vision*, 111(1):98–136, January 2015.
- [21] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59:2004, 2004.
- [22] I. Freeman, L. Roesse-Koerner, and A. Kummert. EffNet: An Efficient Structure for Convolutional Neural Networks. *ArXiv e-prints*, January 2018.
- [23] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C. Berg. DSSD : Deconvolutional single shot detector. *CoRR*, abs/1701.06659, 2017.
- [24] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.
- [25] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.
- [26] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. pages 2672–2680, 2014.
- [27] Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vázquez, and Aaron C. Courville. Pixelvae: A latent variable model for natural images. *CoRR*, abs/1611.05013, 2016.
- [28] Raia Hadsell, Sumit Chopra, and Yann Lecun. Dimensionality reduction by learning an invariant mapping. 2006.

- [29] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.
- [30] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Shijian Tang, Erich Elsen, Bryan Catanzaro, John Tran, and William Dally. Dsd: Regularizing deep neural networks with dense-sparse-dense training flow. 07 2016.
- [31] Stephen Jose Hanson and Lorien Y. Pratt. Comparing biases for minimal network construction with back-propagation. pages 177–185, 1989.
- [32] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI’16*, pages 2094–2100. AAAI Press, 2016.
- [33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR*, abs/1406.4729, 2014.
- [34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.
- [35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.
- [36] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Movenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.
- [37] Jeremy Howard and Sebastian Ruder. Fine-tuned language models for text classification. *CoRR*, abs/1801.06146, 2018.
- [38] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017.
- [39] Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P. Xing. Controllable text generation. *CoRR*, abs/1703.00955, 2017.
- [40] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. Snapshot ensembles: Train 1, get M for free. *CoRR*, abs/1704.00109, 2017.
- [41] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.
- [42] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. *CoRR*, abs/1603.09382, 2016.

- [43] Haiwen Huang, Chang Wang, and Bin Dong. Nostalgic adam: Weighing more of the past gradients when designing the adaptive learning rate. *CoRR*, abs/1805.07557, 2018.
- [44] Aapo Hyvärinen and Urs Köster. Complex cell pooling and the statistics of natural images. *Network: Computation in Neural Systems*, 18(2):81–100, 2007. PMID: 17852755.
- [45] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.
- [46] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.
- [47] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015.
- [48] Shaoqing Ren Jian Sun Kaiming He, Xiangyu Zhang. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv*, février 2015.
- [49] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *CoRR*, abs/1404.2188, 2014.
- [50] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *CoRR*, abs/1404.2188, 2014.
- [51] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, Angela Y. Wu, Senior Member, and Senior Member. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:881–892, 2002.
- [52] Wojciech Marian Czarnecki Katarzyna Janocha. On loss functions for deep neural networks in classification. *arXiv*, février 2017.
- [53] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.
- [54] Gary King and Langche Zeng. Logistic regression in rare events data. *Political Analysis*, 9:137–163, 2001.
- [55] D. P Kingma and M. Welling. Auto-Encoding Variational Bayes. *ArXiv e-prints*, December 2013.

- [56] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.
- [57] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *CoRR*, abs/1706.02515, 2017.
- [58] G. R. Koch. Siamese neural networks for one-shot image recognition. 2015.
- [59] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).
- [60] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-100 (canadian institute for advanced research).
- [61] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. pages 1097–1105, 2012.
- [62] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *CoRR*, abs/1605.07648, 2016.
- [63] Hoa T. Le, Christophe Cerisara, and Alexandre Denis. Do convolutional networks need to be deep for text classification ? *CoRR*, abs/1707.04108, 2017.
- [64] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.
- [65] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Detnet: A backbone network for object detection. *CoRR*, abs/1804.06215, 2018.
- [66] Depeng Liang and Yongdong Zhang. AC-BLSTM: asymmetric convolutional bidirectional LSTM networks for text classification. *CoRR*, abs/1611.01884, 2016.
- [67] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2013.
- [68] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016.
- [69] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.
- [70] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

- [71] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.
- [72] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016.
- [73] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017.
- [74] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015.
- [75] Wolfgang Maass. Efficient agnostic pac-learning with simple hypothesis. pages 67–75, 1994.
- [76] Alireza Makhzani and Brendan J. Frey. k-sparse autoencoders. *CoRR*, abs/1312.5663, 2013.
- [77] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013.
- [78] Y. NESTEROV. A method for unconstrained convex minimization problem with the rate of convergence $o(1/k^2)$. *Doklady AN USSR*, 269:543–547, 1983.
- [79] Andrew Ng. Sparse autoencoder, cs294a lecture notes.
- [80] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jian-shu Chen, Xinying Song, and Rabab K. Ward. Deep sentence embedding using the long short term memory network: Analysis and application to information retrieval. *CoRR*, abs/1502.06922, 2015.
- [81] Hengyue Pan and Hui Jiang. Annealed gradient descent for deep learning. pages 652–661, 2015.
- [82] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning.
- [83] Lutz Prechelt. Early stopping — but when? pages 53–67, 2012.
- [84] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.
- [85] Yoshua Bengio Razvan Pascanu, Tomas Mikolov. On the difficulty of training recurrent neural networks. *arXiv*, février 2013.

- [86] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. 2018.
- [87] J. Redmon and A. Farhadi. YOLOv3: An Incremental Improvement. *ArXiv e-prints*, April 2018.
- [88] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.
- [89] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.
- [90] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. *CoRR*, abs/1803.09050, 2018.
- [91] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.
- [92] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contracting auto-encoders: Explicit invariance during feature extraction. 2011.
- [93] Abhijit Guha Roy, Nassir Navab, and Christian Wachinger. Concurrent spatial and channel squeeze & excitation in fully convolutional networks. *CoRR*, abs/1803.02579, 2018.
- [94] Hojjat Salehinejad, Julianne Baarbe, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaei. Recent advances in recurrent neural networks. *CoRR*, abs/1801.01078, 2018.
- [95] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.
- [96] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015.
- [97] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. A hybrid convolutional variational autoencoder for text generation. *CoRR*, abs/1702.02390, 2017.
- [98] Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. *CoRR*, abs/1603.05201, 2016.
- [99] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1605.06211, 2016.

- [100] Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. *JOURNAL OF COMPUTER AND SYSTEM SCIENCES*, 50(1):132–150, 1995.
- [101] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.
- [102] Leslie N. Smith. No more pesky learning rate guessing games. *CoRR*, abs/1506.01186, 2015.
- [103] Leslie N. Smith. A disciplined approach to neural network hyperparameters: Part 1 - learning rate, batch size, momentum, and weight decay. *CoRR*, abs/1803.09820, 2018.
- [104] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. 15:1929–1958, 06 2014.
- [105] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.
- [106] R. Sutton and A. Barto. *Reinforcement learning: An introduction*. Cambridge, MA, MIT Press, 1998.
- [107] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, Aug 1988.
- [108] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.
- [109] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.
- [110] Christian Szegedy, Scott E. Reed, Dumitru Erhan, and Dragomir Anguelov. Scalable, high-quality object detection. *CoRR*, abs/1412.1441, 2014.
- [111] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.
- [112] L. G. Valiant. A theory of the learnable. pages 436–445, 1984.
- [113] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

- [114] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. pages 1096–1103, 2008.
- [115] Li Wan, Matthew D Zeiler, Sixn Zhang, Yann Lecun, and Rob Fergus. Regularization of neural networks using dropconnect. 01 2013.
- [116] Xingyou Wang, Weijie Jiang, and Zhiyong Luo. Combination of convolutional and recurrent neural network for sentiment analysis of short texts. 2016.
- [117] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1995–2003, New York, New York, USA, 20–22 Jun 2016. PMLR.
- [118] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.
- [119] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Trans. Evol. Comp*, 1(1):67–82, April 1997.
- [120] Alexander Wong, Mohammad Javad Shafiee, Francis Li, and Brendan Chwyl. Tiny SSD: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection. *CoRR*, abs/1802.06488, 2018.
- [121] Lingxi Xie, Jingdong Wang, Zhen Wei, Meng Wang, and Qi Tian. Disturblabel: Regularizing CNN on the loss layer. *CoRR*, abs/1605.00055, 2016.
- [122] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016.
- [123] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853, 2015.
- [124] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alexander J. Smola, and Eduard H. Hovy. Hierarchical attention networks for document classification. 2016.
- [125] Wenpeng Yin and Hinrich Schütze. Multichannel variable-size convolution for sentence classification. *CoRR*, abs/1603.04513, 2016.
- [126] D. Yogatama, C. Dyer, W. Ling, and P. Blunsom. Generative and Discriminative Text Classification with Recurrent Neural Networks. *ArXiv e-prints*, March 2017.

- [127] Dingjun Yu, Hanli Wang, Peiqiu Chen, and Zhihua Wei. Mixed pooling for convolutional neural networks. 2014.
- [128] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122, 2015.
- [129] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016.
- [130] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.
- [131] Matthew D. Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *CoRR*, abs/1301.3557, 2013.
- [132] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.
- [133] Ke Zhang, Miao Sun, Tony X. Han, Xingfang Yuan, Liru Guo, and Tao Liu. Residual networks of residual networks: Multilevel residual networks. *CoRR*, abs/1608.02908, 2016.
- [134] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017.
- [135] Ye Zhang, Stephen Roller, and Byron C. Wallace. MGNC-CNN: A simple approach to exploiting multiple word embeddings for sentence classification. *CoRR*, abs/1603.00968, 2016.
- [136] Ye Zhang and Byron C. Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *CoRR*, abs/1510.03820, 2015.
- [137] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis C. M. Lau. A C-LSTM neural network for text classification. *CoRR*, abs/1511.08630, 2015.
- [138] L. Zhu, R. Deng, Z. Deng, G. Mori, and P. Tan. Sparsely Connected Convolutional Networks. *ArXiv e-prints*, January 2018.