

Cours 6 - Les réseaux de neurones profonds (RNP)

Neila Mezghani

1 & 12 février 2022

Plan du cours

1 Introduction

- Apprentissage profond
- Problèmes de l'apprentissage profond

2 Disparition et explosion des gradients

3 Besoin de grandes quantité de données

- L'apprentissage par transfert
- Techniques d'apprentissage par transfert
- Modèles pré-entraînés

4 Temps de calcul élevé

5 Annexes

Introduction

Disparition et explosion des gradients
Besoin de grandes quantité de données
Temps de calcul élevé
Annexes

Apprentissage profond
Problèmes de l'apprentissage profond

Introduction

Plan du cours

1 Introduction

- Apprentissage profond
- Problèmes de l'apprentissage profond

2 Disparition et explosion des gradients

3 Besoin de grandes quantité de données

- L'apprentissage par transfert
- Techniques d'apprentissage par transfert
- Modèles pré-entraînés

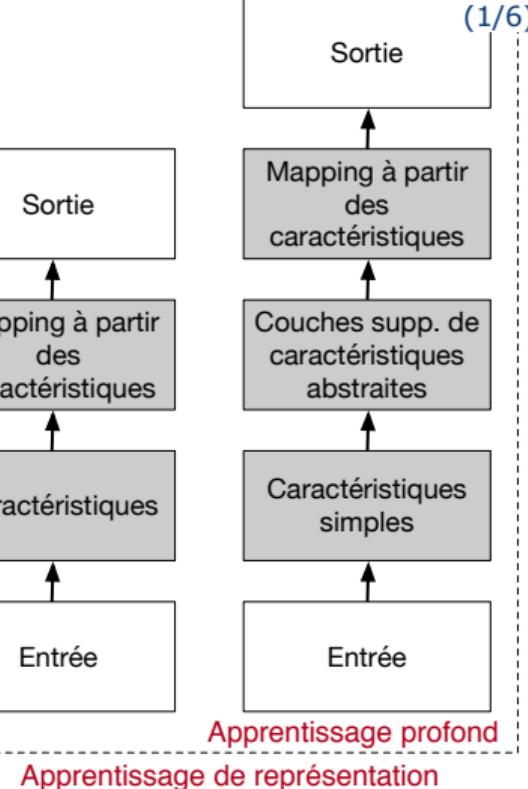
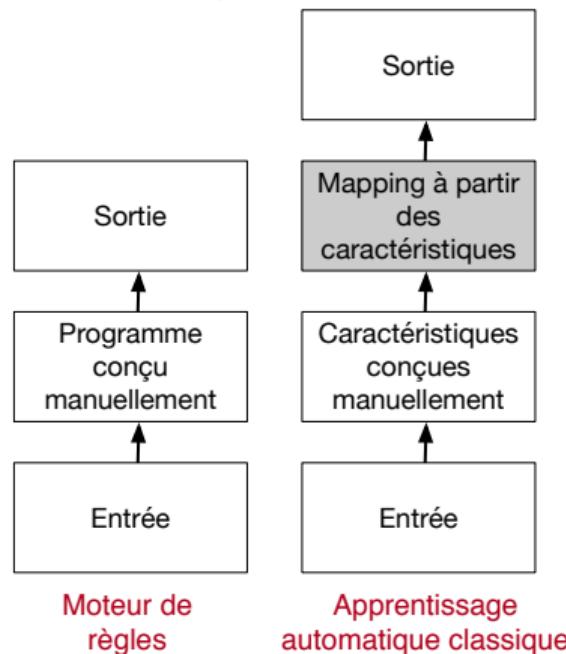
4 Temps de calcul élevé

5 Annexes

Apprentissage profond : Définition

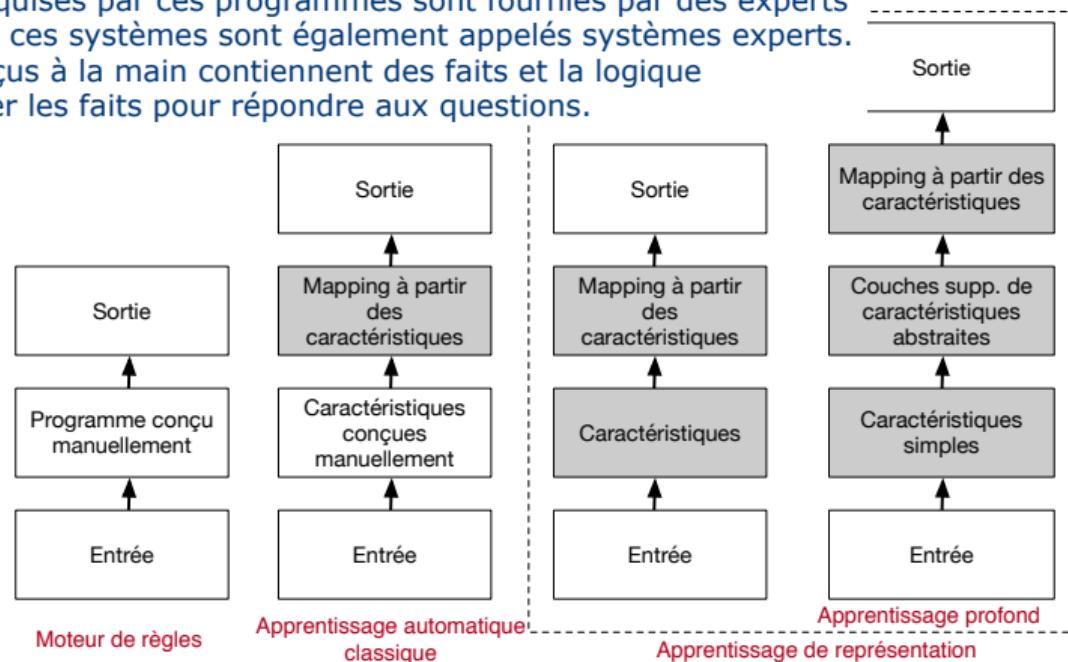
"L'apprentissage profond ou apprentissage en profondeur (deep learning) est un ensemble de méthodes d'apprentissage automatique tentant de modéliser avec un haut niveau d'abstraction des données grâce à des architectures articulées de différentes transformations non linéaires." (dictionnaire terminologique de la langue)

Organigramme descriptif des différentes parties d'un système d'IA.
 Les cases ombrées indiquent les éléments qui se basent sur l'AM



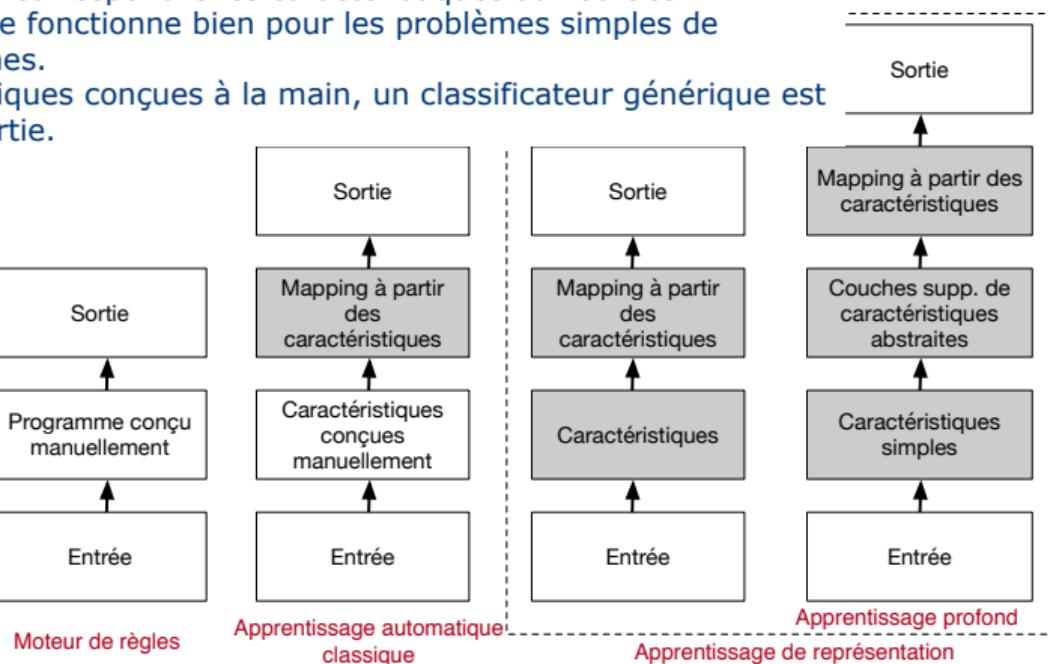
- **Les systèmes à base de règles** sont des programmes d'IA sont conçus à la main.
- Les connaissances requises par ces programmes sont fournies par des experts du domaine concerné : ces systèmes sont également appelés **systèmes experts**.
- Les programmes conçus à la main contiennent des faits et la logique permettant de combiner les faits pour répondre aux questions.

(2/6)



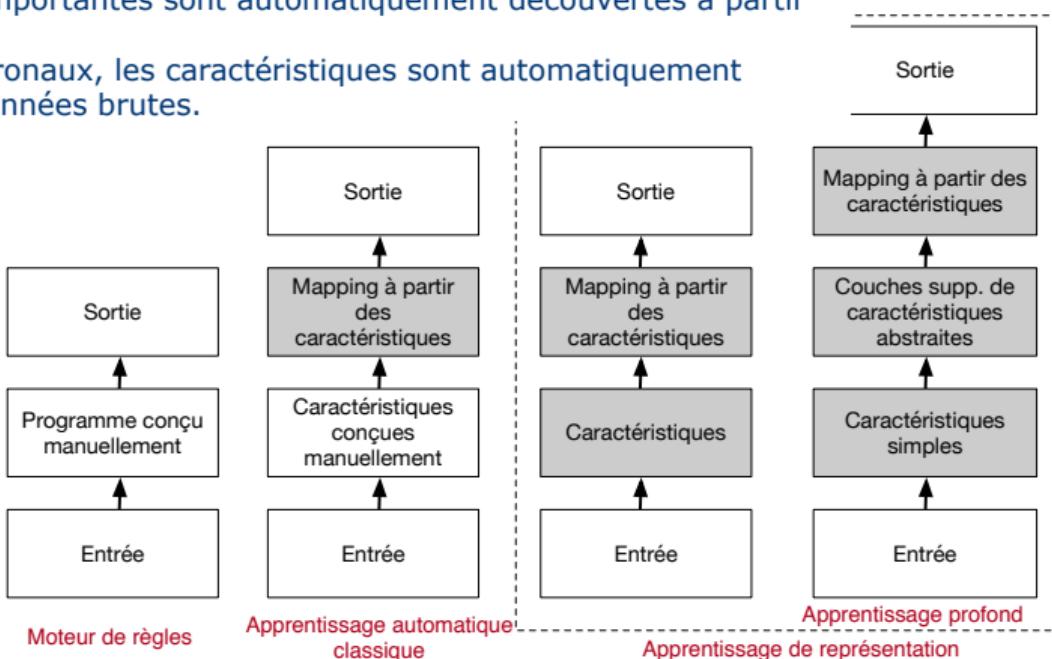
- Dans **l'apprentissage automatique classique**, les caractéristiques importantes de l'entrée sont conçues manuellement et le système apprend automatiquement à faire correspondre les caractéristiques aux sorties.
- Ce type d'apprentissage fonctionne bien pour les problèmes simples de reconnaissance des formes.
- Une fois les caractéristiques conçues à la main, un classificateur générique est utilisé pour obtenir la sortie.

(3/6)



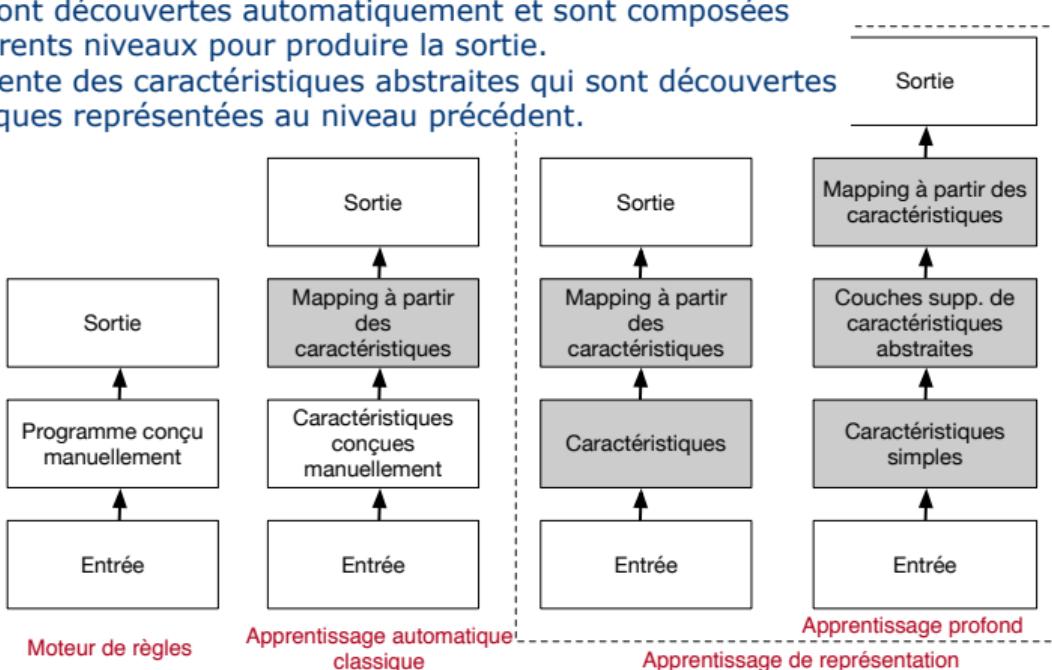
- L'apprentissage par représentation va plus loin et élimine la nécessité de concevoir manuellement les caractéristiques.
- Les caractéristiques importantes sont automatiquement découvertes à partir des données.
- Dans les réseaux neuronaux, les caractéristiques sont automatiquement apprises à partir des données brutes.

(4/6)



- L'apprentissage profond est un type d'apprentissage par représentation dans lequel il existe plusieurs niveaux de caractéristiques.
- Ces caractéristiques sont découvertes automatiquement et sont composées ensemble dans les différents niveaux pour produire la sortie.
- Chaque niveau représente des caractéristiques abstraites qui sont découvertes à partir des caractéristiques représentées au niveau précédent.

(5/6)

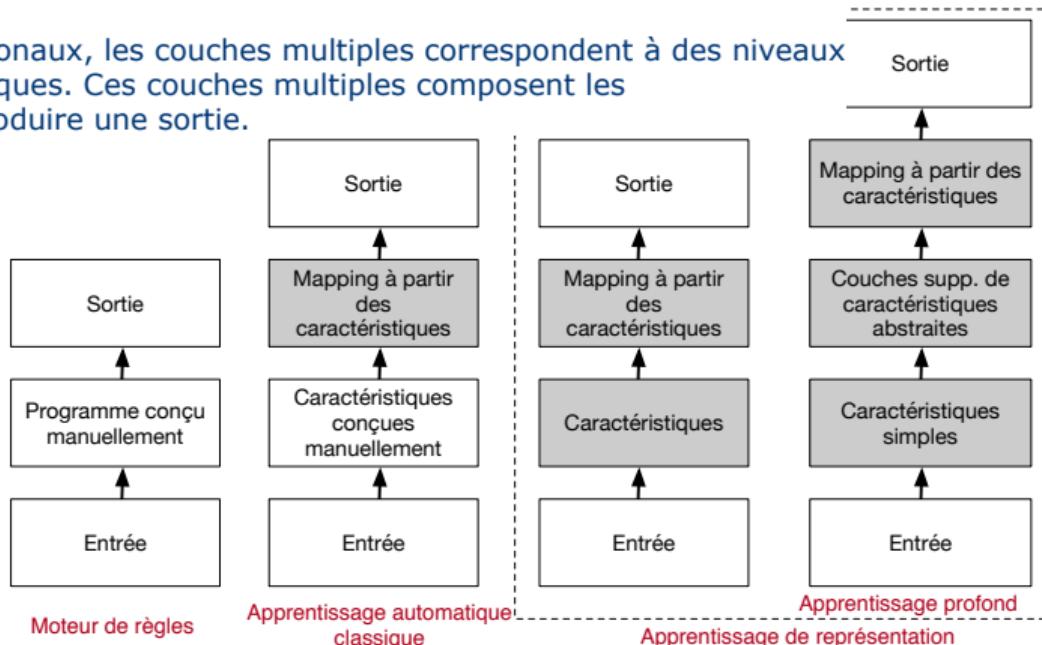


- Dans **l'apprentissage profond** Le niveau d'abstraction augmente avec chaque niveau.

(6/6)

- Ce type d'apprentissage permet de découvrir et de représenter des abstractions de plus haut niveau.

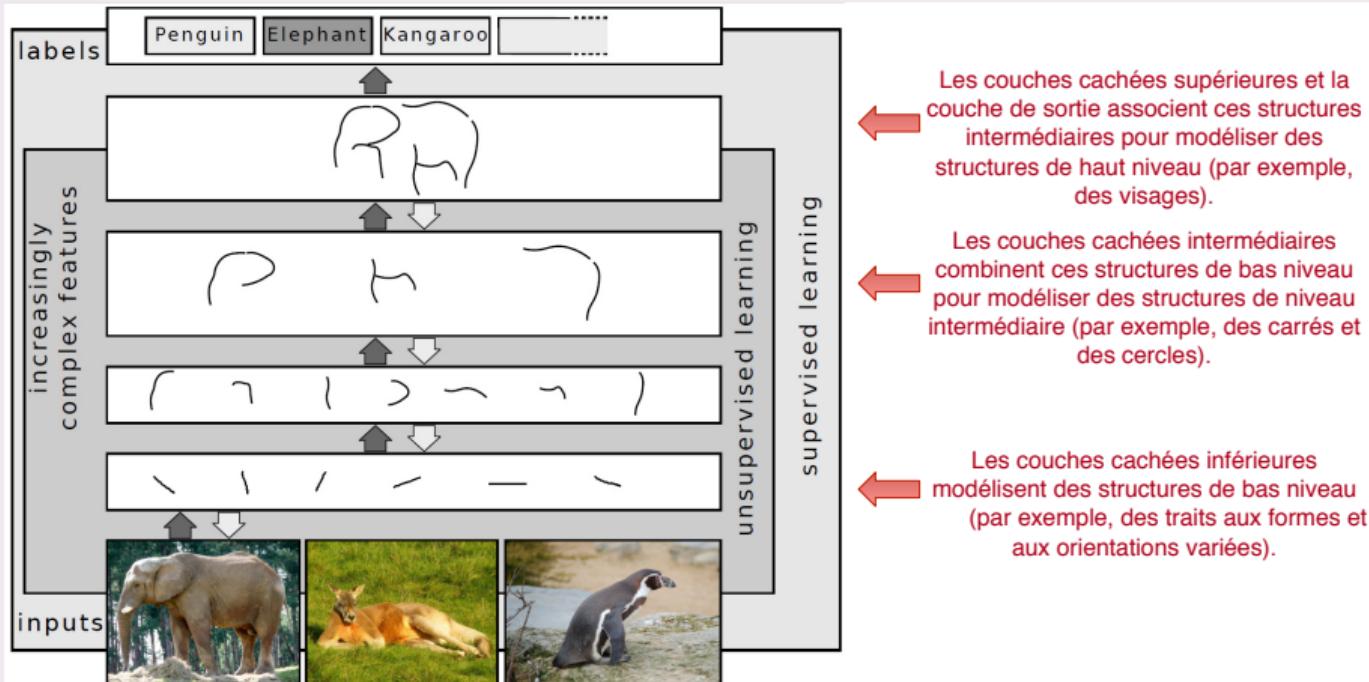
- Dans les réseaux neuronaux, les couches multiples correspondent à des niveaux multiples de caractéristiques. Ces couches multiples composent les caractéristiques pour produire une sortie.



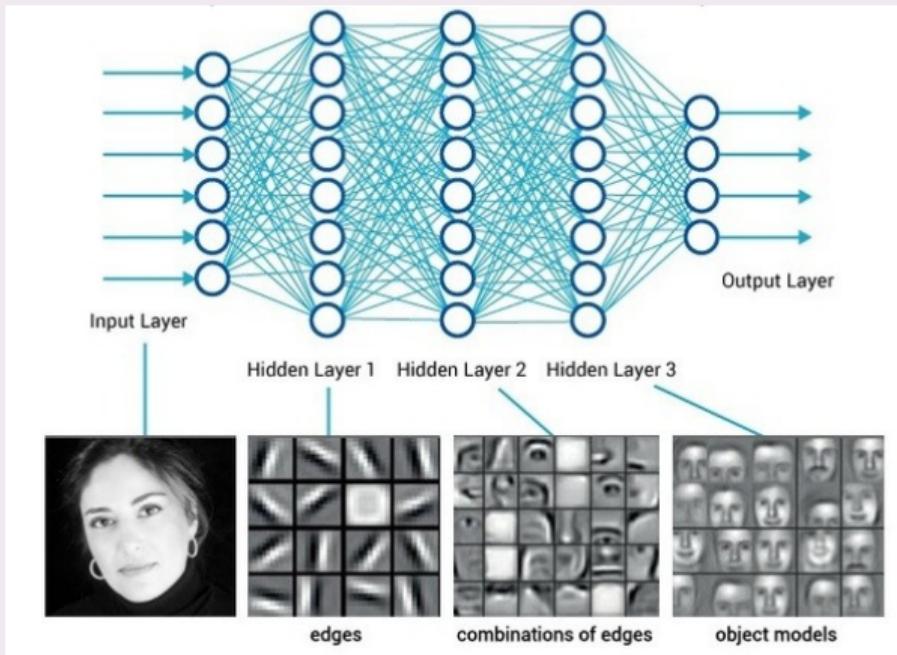
Apprentissage profond : Architecture (1/3)

- Les couches cachées inférieures modélisent des structures de bas niveau (par exemple, des traits aux formes et aux orientations variées).
- Les couches cachées intermédiaires combinent ces structures de bas niveau pour modéliser des structures de niveau intermédiaire (par exemple, des carrés et des cercles).
- Les couches cachées supérieures et la couche de sortie associent ces structures intermédiaires pour modéliser des structures de haut niveau (par exemple, des visages).

Apprentissage profond : Architecture (2/3)



Apprentissage profond : Architecture (3/3)



<https://fr.quora.com/Quelle-est-la-différence-entre-l'intelligence-artificielle-l'apprentissage-machine-l'apprentissage-profound-et-le-traitement-du-langage-naturel>

Les principales architectures des réseaux d'apprentissage profonds

Les trois architectures principales des réseaux d'apprentissage profonds sont :

- Les réseaux de neurones convolutifs (CNN) (Cours 6)
- Les réseaux de neurones récurrents (RNN) (Cours 7)
- Les Auto encodeurs (Cours 8)

Plan du cours

1 Introduction

- Apprentissage profond
- Problèmes de l'apprentissage profond

2 Disparition et explosion des gradients

3 Besoin de grandes quantité de données

- L'apprentissage par transfert
- Techniques d'apprentissage par transfert
- Modèles pré-entraînés

4 Temps de calcul élevé

5 Annexes

Problèmes de l'apprentissage profond

- La multiplication et l'ajout de couches permettent la résolution de problématique de classification de données complexes.
- Cependant, elles génèrent des problèmes liés à la profondeur :
 - ◊ Disparition et explosion des gradients
 - ◊ Besoin de grandes quantité de données = Besoin d'un ensemble de données d'entraînement considérable
 - ◊ Temps de calcul élevé
 - ◊ Risque de sur-apprentissage (résolu par l'utilisation de la régularisation)
- Chacun de ses problèmes a fait et fait actuellement l'objet de travaux de recherche \implies plusieurs solutions ont été proposées.

Introduction

Disparition et explosion des gradients

Besoin de grandes quantité de données

Temps de calcul élevé

Annexes

Disparition et explosion des gradients

Disparition et explosion des gradients

- L'algorithme de rétropropagation opère de la couche de sortie vers la couche d'entrée, en propageant au fur et à mesure le gradient d'erreur.
- Lorsque l'algorithme a déterminé le gradient de la fonction de coût par rapport à chaque paramètre du réseau, il utilise les gradients obtenus pour actualiser chaque paramètre au cours d'une étape de descente du gradient.
- Deux problèmes peuvent survenir :
 - Disparition des gradients (vanishing gradients).
 - Explosion des gradients (exploding gradients).

Disparition des gradients (vanishing gradients)

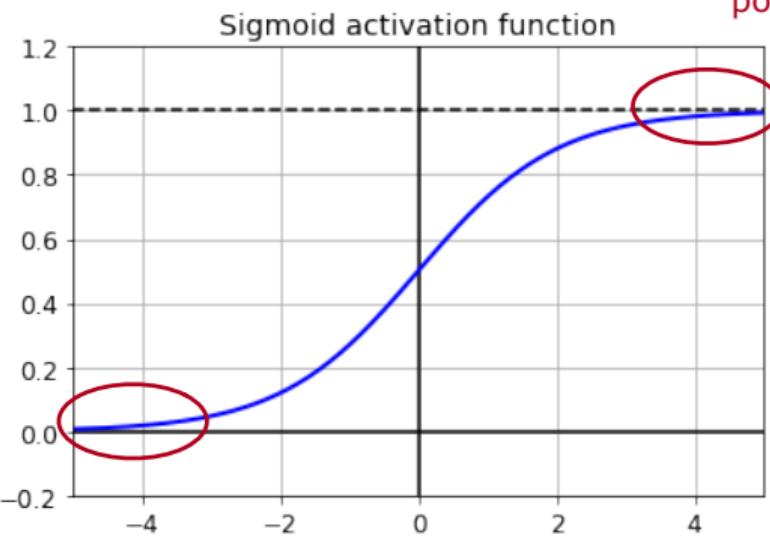
- Pendant que l'algorithme progresse vers les couches inférieures, les gradients deviennent souvent de plus en plus petits.
- Par conséquent, la mise à jour par descente de gradient modifie très peu les poids des connexions des couches inférieures et l'entraînement ne converge jamais vers une bonne solution \implies le problème de disparition des gradients (vanishing gradients).

Explosion des gradients (exploding gradients)

- Dans certains cas, l'opposé peut se produire : Les gradients deviennent de plus en plus gros et de nombreuses couches reçoivent des poids extrêmement importants, ce qui fait diverger l'algorithme \implies Il s'agit du problème d'explosion des gradients (exploding gradients)
- Ce problème est rencontré principalement dans les réseaux de neurones récurrents (RNN)

Disparition et explosion des gradients

- En résumé, les réseaux de neurones profonds souffrent de l'instabilité des gradients
- Cette instabilité était une des causes d'abandon des réseaux de neurones profonds au début des années 2000.
- Ceci est dû notamment à l'association de :
 - ◊ **La fonction d'activation logistique sigmoïde** répandue à l'époque et
 - ◊ **La technique d'initialisation des poids** également la plus répandue, à savoir une loi de distribution normale $\mathcal{N}(0, 1)$.
- Lors de la progression dans le réseau, la variance ne cesse d'augmenter après chaque couche, jusqu'à la saturation de la fonction d'activation dans les couches supérieures.



Lorsque les entrées sont grandes (en négatif ou en positif), la fonction sature en 0 ou en 1, avec une dérivé extrêmement proche de 0.



Lorsque la rétropropagation intervient, elle n'a pratiquement aucun gradient. transmettre en arrière dans le réseau.



Il ne reste donc quasi plus rien aux couches inférieures

Solutions à la disparition et l'explosion des gradients

- Plusieurs solutions/compromis, ont été proposées dans la littérature.
Parmi lesquelles on retrouve :

Solutions à la disparition et l'explosion des gradients

Initialisations de
Glorot et de He

Utilisation de
fonctions d'activation
non saturantes

Normalisation par lots
*« Batch
Normalization, BN»*

L'initialisations de Glorot et de He (1/3)

- Plusieurs solutions (compromis), dont la validité est pratique, ont été proposées les chercheurs Glorot, de He et de Bengio.
- Ces solutions reposent sur le nombre de nœuds d'entrées fan_{in} et ou de sortie fan_{out} pour l'initialisation aléatoires des connexions.
- Cette stratégie d'initialisation est appelée initialisation de Xavier ou initialisation de Glorot

L'initialisations de Glorot et de He (2/3)

- Par exemple, si la fonction d'activation est logistique, Glorot propose une distribution normale avec une moyenne nulle et un écart-type

$$\sigma^2 = \frac{1}{fan_{moyen}}$$

ou bien une distribution uniforme entre $-r$ et r avec

$$r = \sqrt{\frac{3}{fan_{moyen}}}$$

avec

$$fan_{moyen} = \frac{fan_{in} + fan_{out}}{2}$$

L'initialisations de Glorot et de He (3/3)

Initialisation	Fonction d'activations	σ^2 de la distribution normale
He	tanh, logistique, softmax	$\frac{1}{fan_{moyen}}$
Glorot	ReLU et ses variantes	$\frac{2}{fan_{in}}$
Bengio	SEL	$\frac{1}{fan_{in}}$

Exemple : Disparition et explosion des gradients

Keras choisit par défaut l'initialisation de Glorot avec une distribution uniforme. Si on désire changer ce type d'initialisation lors de la création d'une couche, on spécifie le `kernel_initializer`

```
# Initialisation de He
keras.layers.Dense(10, activation="relu",
                  kernel_initializer="he_normal")
```

Solutions à la disparition et l'explosion des gradients

Solutions à la disparition et l'explosion des gradients

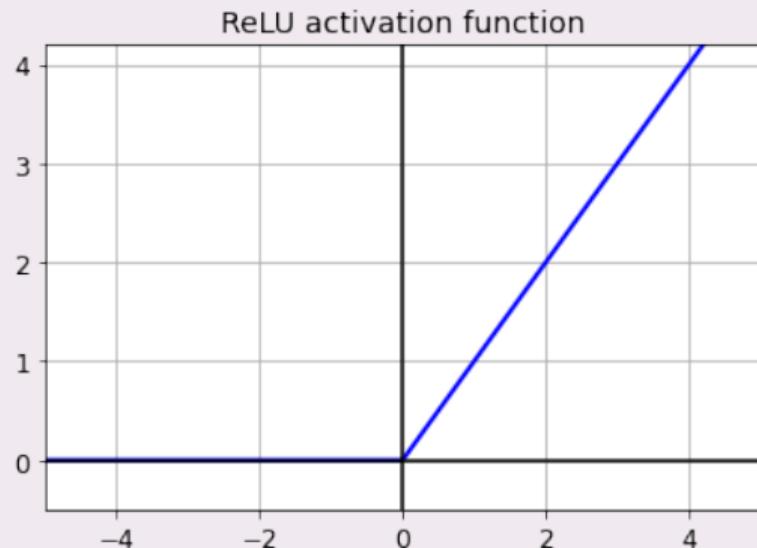
Initialisations de
Glorot et de He

Utilisation de
fonctions d'activation
non saturantes

Normalisation par lots
*« Batch
Normalization, BN»*

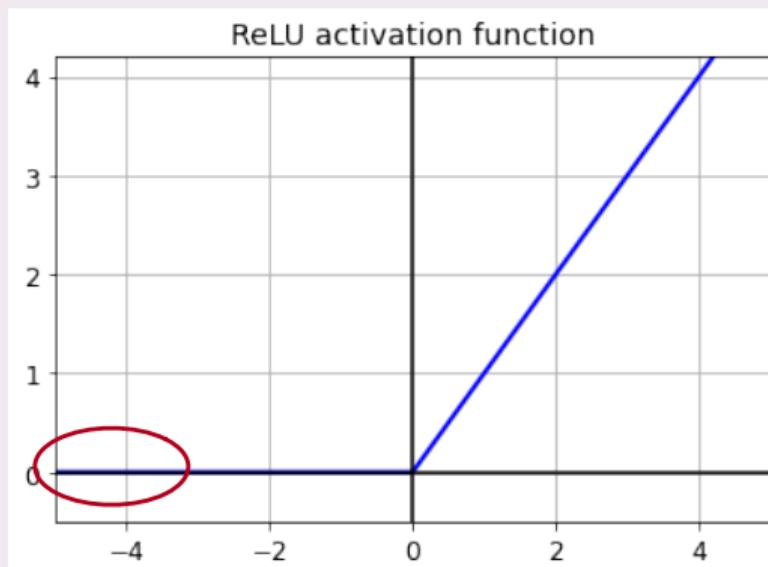
Fonctions d'activation non saturantes (1/3)

- Certains travaux de recherche ont montré que l'instabilité des gradients était en partie due à un mauvais choix de la fonction d'activation.
- Pour un MLP, la fonction d'activation la plus courante est la fonction ReLU principalement parce qu'elle ne sature pas pour les valeurs positives



Fonctions d'activation non saturantes (2/3)

- Malheureusement, la fonction d'activation ReLU n'est pas parfaite.
- Elle souffre d'un problème de mort des ReLU (dying ReLU) : au cours de l'entraînement, certains neurones meurent (saturent)
⇒ Besoin de fonctions d'activations non saturantes



Fonctions d'activation non saturantes (3/3)

La machine d'activation non saturantes :

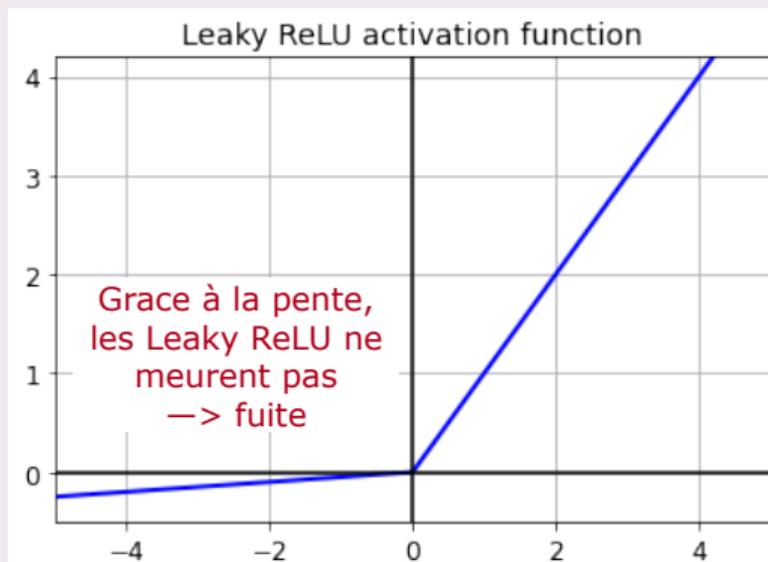
- Leaky ReLU (Leaky Rectified Linear Unit)
- Randomized leaky ReLU (RReLU)
- ELU (Exponential Linear Unit),
- SELU (Scaled ELU)

(1) Leaky ReLU

- Pour résoudre ce problème, on emploie une variante de la fonction ReLU, par exemple Leaky ReLU

$$\text{LeakyRelu}_\alpha(z) = \max(\alpha z, z)$$

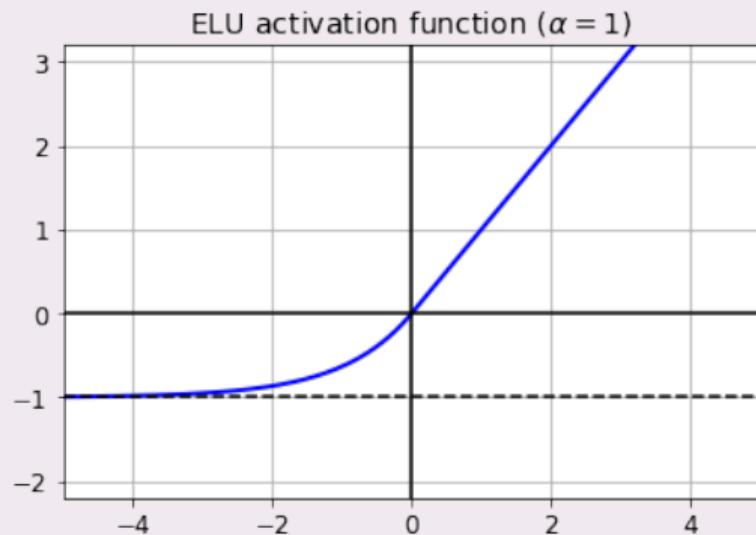
- L'hyperparamètre α définit le niveau de « fuite » de la fonction.



(2) ELU

- Une nouvelle fonction d'activation appelée Elu (Exponential Linear Unit), s'est montrée bien plus performante que toutes les variantes de ReLU

$$\begin{cases} f(x) = \alpha(e^x - 1) & \text{si } x < 0 \\ f(x) = x & \text{si } x \geq 0 \end{cases}$$



Solutions à la disparition et l'explosion des gradients

Solutions à la disparition et l'explosion des gradients

Initialisations de
Glorot et de He

Utilisation de
fonctions d'activation
non saturantes

Normalisation par lots
*« Batch
Normalization, BN»*

Normalisation par lots

- La normalisation par lots (BN, Batch Normalization) consiste à normaliser la distribution des entrées de couche pour lutter contre le décalage de covariance interne \implies permet de contrôler la quantité de décalage des unités cachées.

Exemple : Normalisation par lots

Keras permet la mise en oeuvre de la normalisation par lots très facilement via **BatchNormalization**

```
keras.layers.Dense(300, activation="elu",  
                  kernel_initializer="he_normal"),  
keras.layers.BatchNormalization(),  
keras.layers.Dense(100, activation="elu",  
                  kernel_initializer="he_normal"),  
keras.layers.BatchNormalization(),  
keras.layers.Dense(10, activation="softmax")  
])
```

Besoin de grandes quantité de données

Besoin de grandes quantité de données

- Les modèles de Deep learning requièrent de grande quantité de données
⇒ si le modèle est supervisé, on a besoin de beaucoup de données étiquetées = l'étiquetage des données est très fastidieux et prend beaucoup de temps.
- Le besoin en grande quantité de données entraîne une lenteur de la phase d'entraînement et une complexité accrue des modèles. ⇒ d'où le besoin d'utiliser des couches pré-entraînées = Apprentissage par transfert.

Plan du cours

1 Introduction

- Apprentissage profond
- Problèmes de l'apprentissage profond

2 Disparition et explosion des gradients

3 Besoin de grandes quantité de données

- **L'apprentissage par transfert**
- Techniques d'apprentissage par transfert
- Modèles pré-entraînés

4 Temps de calcul élevé

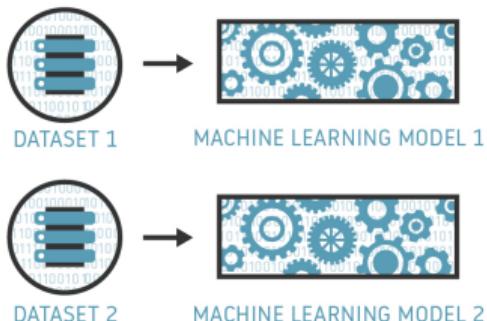
5 Annexes

L'apprentissage par transfert : Définition (1/2)

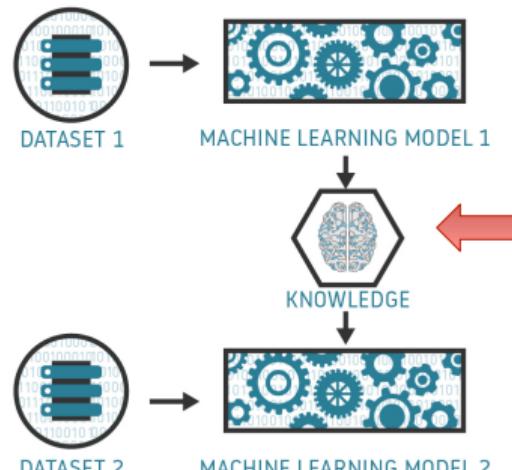
- En général : L'apprentissage par transfert repose sur une idée simple qui consiste à ré-exploiter les connaissances acquises dans d'autres configurations (sources) pour la résolution d'un problème particulier (cible).
- Dans le cas des réseaux de neurones, le transfert d'apprentissage (transfer learning) consiste à utiliser un réseau de neurones existant qui accomplit une tâche comparable à celle visée
- Il permet d'accélérer considérablement l'entraînement et d'obtenir de bonnes performances avec des jeux de données d'entraînement assez petits.

L'apprentissage par transfert : Définition (2/2)

TRADITIONAL MACHINE LEARNING



TRANSFER LEARNING



Avec l'apprentissage par transfert, au lieu d'entraîner le modèle à chaque fois de nouveau, on peut utiliser les connaissances acquises pour une tâche afin de résoudre des tâches connexes.

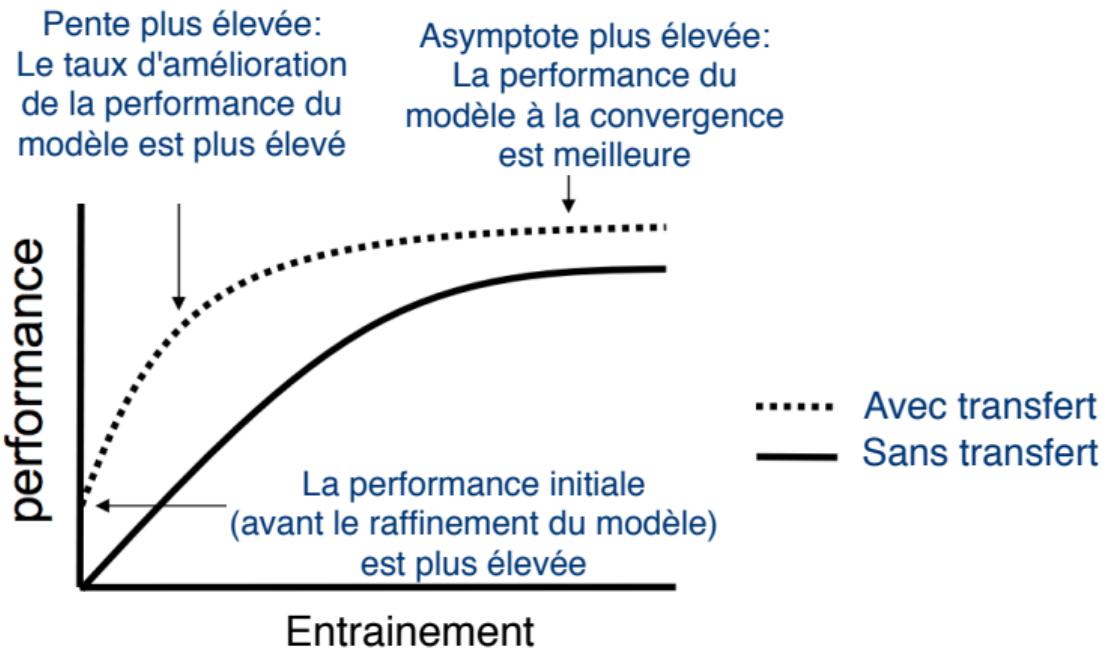
<https://datascience.aero/transfer-learning-aviation/>

Transfert d'apprentissage : Avantages (1/2)

Trois avantages possibles suite à l'utilisation de l'apprentissage par transfert :

- Début plus élevé : La performance initiale du modèle (avant le raffinement du modèle) est plus élevée qu'elle ne le serait autrement.
- Pente plus élevée : Le taux d'amélioration de la performance du modèle est plus élevé qu'il ne le serait autrement.
- Asymptote plus élevée : À la convergence la performance du modèle est meilleure qu'elle ne le serait autrement.

Transfert d'apprentissage : Avantages (2/2)



Plan du cours

1 Introduction

- Apprentissage profond
- Problèmes de l'apprentissage profond

2 Disparition et explosion des gradients

3 Besoin de grandes quantité de données

- L'apprentissage par transfert
- **Techniques d'apprentissage par transfert**
- Modèles pré-entraînés

4 Temps de calcul élevé

5 Annexes

Techniques d'apprentissage par transfert(1/4)

On peut distinguer trois techniques d'apprentissage par transfert :

Apprentissage par transfert *« Transfer learning »*

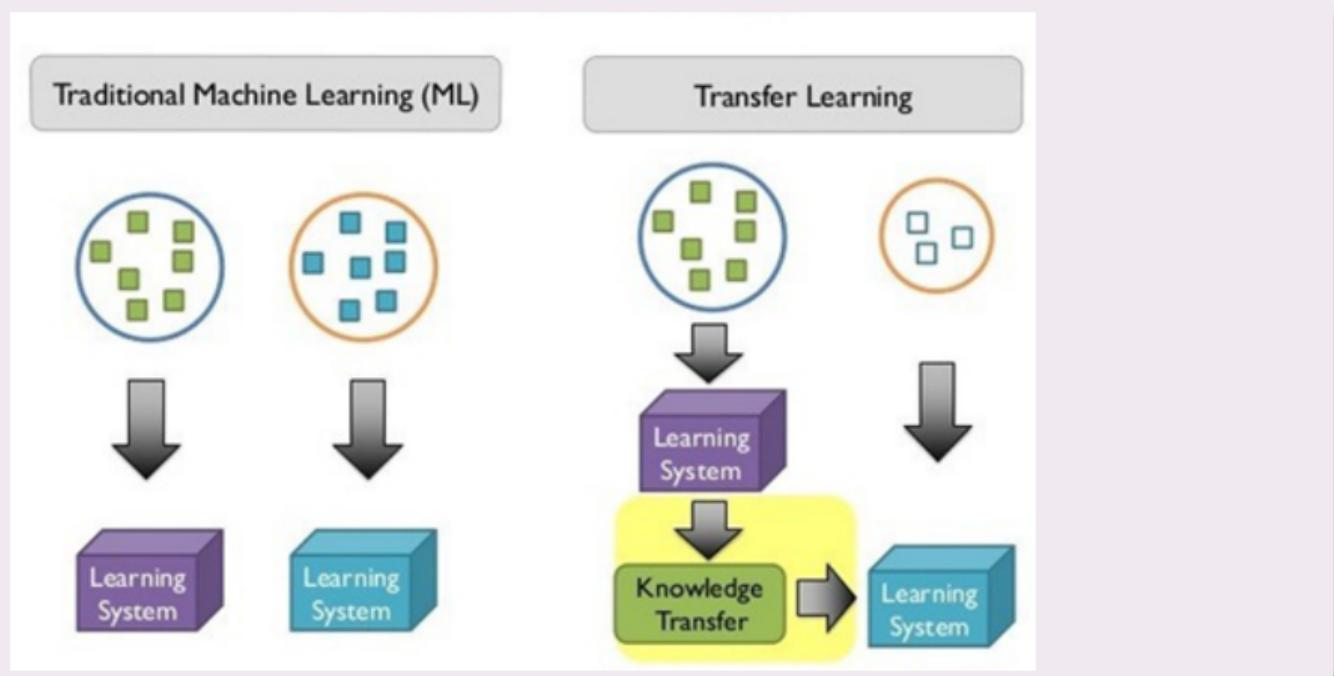
Apprentissage par transfert inductif
« Inductive Transfer Learning »

Apprentissage par transfert non supervisé
« Unsupervised Transfer Learning »

Apprentissage par transfert transductif
« Transductive Transfer Learning »

Apprentissage par transfert inductif (2/4)

- Dans le cas de l'apprentissage par transfert inductif, les domaines source et cible sont similaires, c-à-d, comprennent les mêmes données.
- Par contre, les tâches source et cible sont différentes mais proches.
- L'idée consiste à utiliser les modèles existants pour réduire de manière avantageuse le champ d'application des modèles possibles.
- **Exemple :** Utiliser un modèle entraîné pour la détection d'animaux sur des images pour construire un modèle capable d'identifier des chiens.



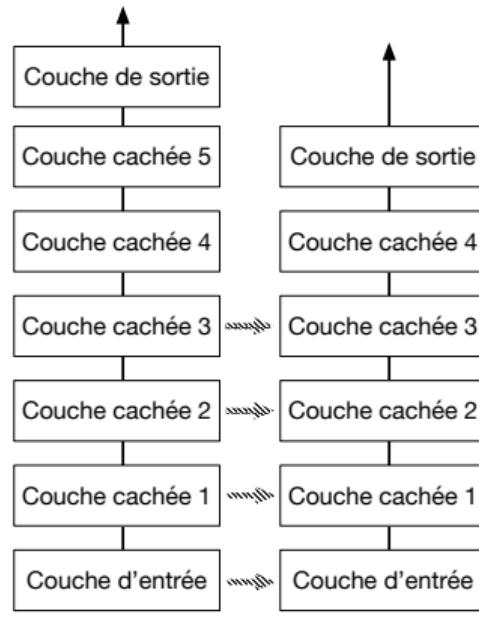
Apprentissage par transfert non supervisé (3/4)

- Dans le cas de l'apprentissage par transfert non supervisé, les domaines source et cible sont similaires mais ces données ne sont pas étiquetées.
- Les tâches source et cible sont différentes mais proches. L'idée consiste alors à utiliser les modèles existants pour réduire de manière avantageuse le champ d'application des modèles possibles.
- L'objectif principal ici est de résoudre des tâches d'apprentissage non supervisées telles que le clustering et la réduction de la dimensionnalité dans le domaine cible.

Apprentissage par transfert transductif (4/4)

- Dans cette configuration, la tâche source et la tâche cible sont identiques, mais le domaine source et le domaine cible sont différents.
- La plupart des données étiquetées existent dans le domaine source et il n'y a absolument aucune donnée étiquetée présente dans le domaine cible.

Exemple : Transfert d'apprentissage (1/2)



- On dispose d'un RNP A
- On fixe toutes les couches à réutiliser afin que la descente de gradient ne les modifie
- On entraîne le RNP B et on examine ses performances
- On libère une (ou deux) des couches cachées supérieures pour appliquer la rétropropagation
- On examine si les performances s'améliorent.

Exemple : Transfert d'apprentissage (2/2)

Keras permet la réutilisation de couche entraînées. Le chargement de modèle existant se fait via `Keras.models.load_model`

```
model_A = keras.models.load_model("my_model_A.h5")
model_B_on_A = keras.models.Sequential(model_A.layers[:-1])
model_B_on_A.add(keras.layers.Dense(1, activation="sigmoid"))
```

 Le Hierarchical Data Format (HDF) est un ensemble de formats de fichiers permettant de sauvegarder et de structurer des fichiers contenant de très grandes quantités de données. (.hdf, .h4, .hdf4, .he4 (pour HDF4), .h5, .hdf5 et .he5 (pour HDF5))

Plan du cours

1 Introduction

- Apprentissage profond
- Problèmes de l'apprentissage profond

2 Disparition et explosion des gradients

3 Besoin de grandes quantité de données

- L'apprentissage par transfert
- Techniques d'apprentissage par transfert
- Modèles pré-entraînés

4 Temps de calcul élevé

5 Annexes

Modèles pré-entraînés (1/6)

- Il existe une douzaine de modèles de reconnaissance d'images très performants qui peuvent être téléchargés et utilisés comme base pour la reconnaissance d'images et les tâches connexes de vision par ordinateur.
- Trois des modèles les plus populaires sont les suivants :

Modèles de réseaux de neurones pré-entraînés

VGG
(par ex. VGG16 ou VGG19)

GoogLeNet
(par ex. InceptionV3)

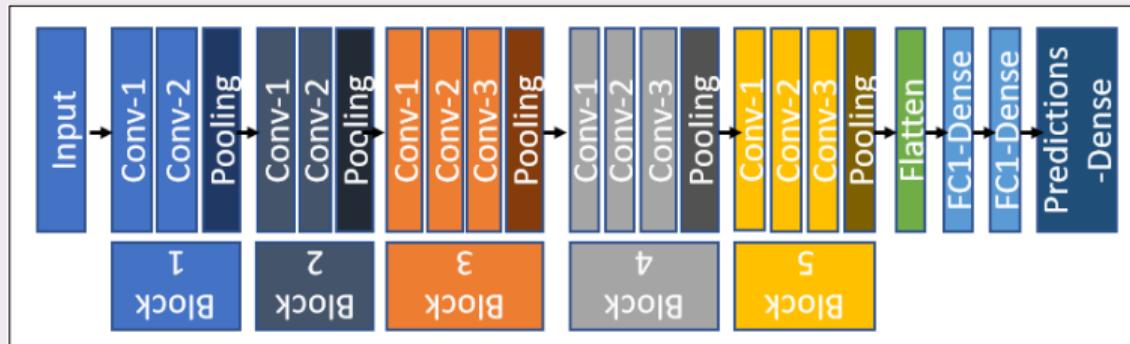
Réseau résiduel
(par ex. ResNet50)

Modèles pré-entraînés (2/6)

- Les trois modèles sont largement utilisés pour l'apprentissage par transfert :
 - ◊ en raison de leurs performances,
 - ◊ parce qu'il s'agit d'exemples qui ont introduit des innovations architecturales spécifiques, à savoir des structures cohérentes et répétitives (VGG), des modules d'inception (GoogLeNet) et des modules résiduels (ResNet).
- Keras donne accès à un certain nombre de modèles pré-entraînés très performants développés pour la reconnaissance d'images.

Modèles pré-entraînés : VGG (3/6)

- VGG16 est un modèle de réseau de neurones à convolution.
- L'architecture du VGG16 est décrite dans le document «*Very Deep Convolutional Networks for Large-Scale Image Recognition*»
- Ce modèle atteint une précision de test de 92,7% dans ImageNet



Modèles pré-entraînés : VGG (4/6)

- Nous pouvons, ensuite, utiliser toutes (ou partiellement) les couches du modèle pré-entraîné pour le développement du nouveau modèle.
- Étapes :
 - ◊ Charger le modèle pré-entraîné
 - ◊ Ajouter de nouvelles couches.

Modèles pré-entraînés : VGG (5/6)

Keras permet de télécharger facilement le modèle pré-entraîné VGG16 et d'afficher son contenu.

```
from keras.applications.vgg16 import VGG16
# load model
model = VGG16()
# summarize the model
model.summary()
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

Total params: 138,357,544

Modèles pré-entraînés : VGG (6/6)

Nous pouvons, ensuite, ajouter les nouvelles couches

```
# load model without classifier layers
model = VGG16(include_top=False, input_shape=(300, 300, 3))
# add new classifier layers
flat1 = Flatten()(model.layers[-1].output)
class1 = Dense(1024, activation='relu')(flat1)
output = Dense(10, activation='softmax')(class1)
# define new model
model = Model(inputs=model.inputs, outputs=output)
```

Temps de calcul élevé

Rappel : La descente de gradient (1/2)

- L'objectif de l'apprentissage est de minimiser la fonction de coût en trouvant la valeur optimisée des poids \implies On utilise l'algorithme de descente de gradient.
- Rappelons que la descente de gradient met à jour les poids selon l'équation :

$$\theta \leftarrow \theta - \eta \nabla_{\theta} J(\theta)$$

$J(\theta)$ est la fonction de coût par rapport aux poids

$\nabla_{\theta} J(\theta)$ est le gradient de la fonction de cout

η est le taux d'apprentissage.

- On a un seul poids w (ou θ) dans une fonction de coût $C(w)$
- On doit déterminer la valeur de w qui minimise $C(w)$
- On fait une mise à jour du paramètre w en selon l'équation:
$$\theta \leftarrow \theta - \eta \nabla_{\theta} J(\theta)$$

1

Rappel : La descente de gradient (2/2)

- Différents types de descentes de gradient sont :
 - ◊ Descente de gradient par lots ou descente de gradient par batch
 - ◊ Descente de gradient stochastique
 - ◊ Descente de gradient en mini lot
- Lorsque le réseau de neurones est très profond, l'entraînement devient très lent
 - ⇒ on peut utiliser des optimiseurs

Optimiseurs

Il existe plusieurs optimiseurs, dont :

Optimiseurs « *Optimisers* »

Optimisation avec
inertie
« *Momentum Optimization* »

Gradient accéléré de
Nesterov « *NAG, Nesterov Accelerated Gradient* »

Optimisation Adam
et Nadam
« *L'algorithme AdaGrad* »

Optimisation avec inertie

- Contrairement à la descente de gradient classique, l'optimisation avec inertie s'intéresse aux gradients antérieurs
- À chaque itération, elle soustrait le gradient local (multiplié par le taux d'apprentissage η) au vecteur d'inertie m et actualise les poids θ en additionnant simplement ce vecteur d'inertie

$$\theta_{t+1} \leftarrow \theta_t - \eta \nabla_\theta J(\theta) + \gamma \nu_t$$

γ est un terme constant (généralement = 0.9) appelé **momentum**.
 ν_t est le dernier changement (dernière mise à jour) pour θ .

- À chaque époque, on fait rouler la balle en bas de la colline.
- Plus la pente est grande, plus la balle roule plus rapidement vers le minimum local dans l'itération suivante.

$$\theta_{t+1} \leftarrow \theta_t - \eta \nabla_{\theta} J(\theta) + \gamma \nu_t$$

—> Exactement ce qu'on veut

2

Gradient accéléré de Nesterov

- Le gradient accéléré de Nesterov est une variante de l'optimisation avec inertie
- Il mesure le gradient de la fonction de coût non pas à la position locale θ mais légèrement en avant à la position.

Optimisation Adam

- Dans cet algorithme d'optimisation, les moyennes courantes des gradients et des seconds moments des gradients sont utilisées.
- L'optimiseur Adam est utilisé pour calculer les taux d'apprentissage adaptatifs pour chaque paramètre.
- Nadam est l'optimiseur Adam avec le momentum Nesterov.

Exemple : Optimiseurs avancés

Keras permet de fixer les optimizers moyennant : `keras.optimizers.SGD`

```
# Optimizer Momentum
optimizer = keras.optimizers.SGD(lr=0.001, momentum=0.9)
# optimizer Nesterov
optimizer = keras.optimizers.SGD(lr=0.001, momentum=0.9,
                                 nesterov=True)
# optimizer AdaGrad
optimizer = keras.optimizers.Adam(lr=0.001, beta_1=0.9,
                                   beta_2=0.999)
```

Annexes

Époque/Lot/Itération (1/3)

Dans la terminologie des réseaux de neurones :

- Un lot (ensemble/batch/partie) = un ensemble d'échantillons de la base de données d'entraînement à traiter.
 - ◊ Il est souvent impossible de transmettre l'ensemble des données au réseau neuronal en une seule fois.
 - ◊ Plus la taille du lot est élevée, plus nous aurons besoin d'espace mémoire.

Époque/Lot/Itération (2/3)

Dans la terminologie des réseaux de neurones :

- Une époque (epoch) = une passe (avant et arrière) de tout les échantillons de la base de données dans le réseau de neurone.
- Le nombre d'itérations = nombre de passes nécessaires pour compléter une époque.
 - ◊ Une passe utilise les échantillons du lot.

Époque/Lot/Itération (3/3)

On a 1000 échantillons, on peut choisir une taille de lot de 20 échantillons



Une époque consiste à passer tout les 1000 échantillons de la base de données



Si on a 1000 échantillons et une taille de lot de 20 échantillons, alors dans une époque on exécutera 50 itérations

