

Ingénieur en instrumentation

« Réseaux de neurones artificiels »

Anissa MOKRAOUI

Laboratoire de Traitement et Transport de l'Information (L2TI, UR 3043)

Bâtiment E, bureau 211

E-mail : anissa.mokraoui@univ-paris13.fr

Tel : 01 49 40 40 60

1

Qu'est ce qu'un neurone artificiel ?

« Le perceptron simplifié »

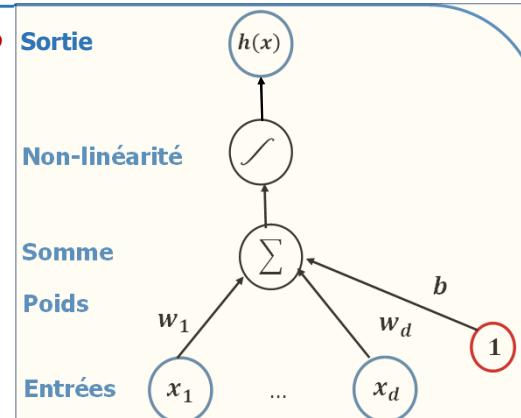
■ Pré-activation du neurone (activation des entrées) :

$$a(x) = b + \sum_i w_i x_i = b + w^T x$$

w= $[w_1, w_2, \dots, w_d]^T$ poids des connexions

b : Biais du neurone

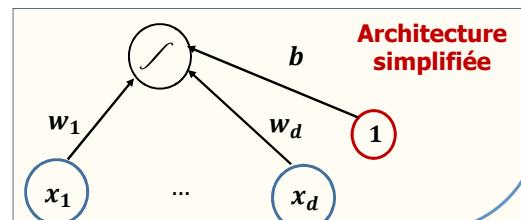
x = $[x_1, x_2, \dots, x_d]^T$ entrées



■ Activation du neurone (sortie) :

$$h(x) = g(a(x)) = g(b + \sum_i w_i x_i) = g(b + w^T x)$$

$g(\cdot)$: Fonction d'activation (introduit des propriétés de non-linéarité)



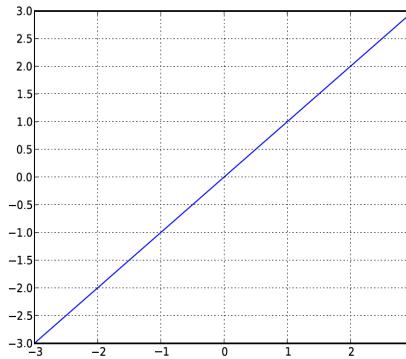
2

Fonctions d'activation

■ Fonction d'activation linéaire :

$$g(a) = a$$

- ✓ Aucun traitement
- ✓ Pas très intéressant



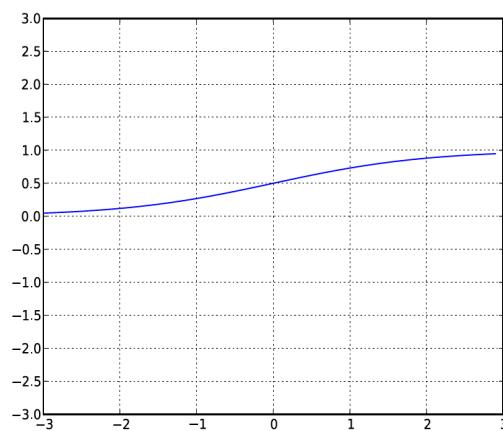
3

■ Fonction d'activation sigmoid :

 `tf.math.sigmoid(z)`

$$g(a) = \text{sigma}(a) = \frac{1}{1 + e^{-a}}$$

- ✓ Toujours positive
- ✓ Strictement croissante
- ✓ Bornée
- ✓ Ecrase la pre-activation du neurone entre 0 et 1

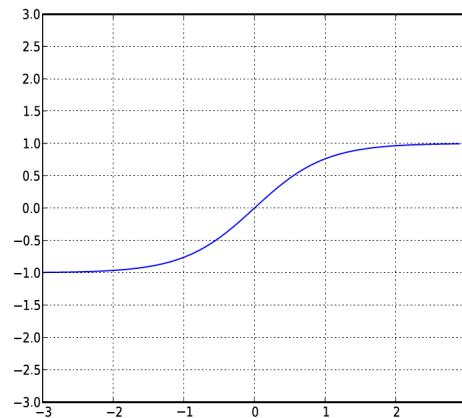


4

■ Fonction d'activation tangent hyperbolique :

$$g(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} = \frac{e^{2a} - 1}{e^{2a} + 1}$$

 `tf.math.tanh(z)`



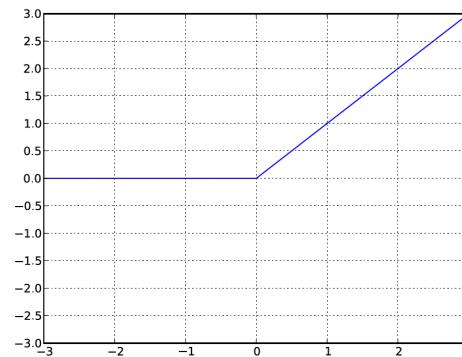
5

- ✓ Peut être négative ou positive
- ✓ Strictement croissante
- ✓ Bornée
- ✓ Ecrase la pre-activation du neurone entre -1 et 1

■ Fonction d'activation linéaire rectifiée (ReLU) :

$$g(a) = \text{relin}(a) = \max(0, a)$$

 `tf.nn.relu(z)`

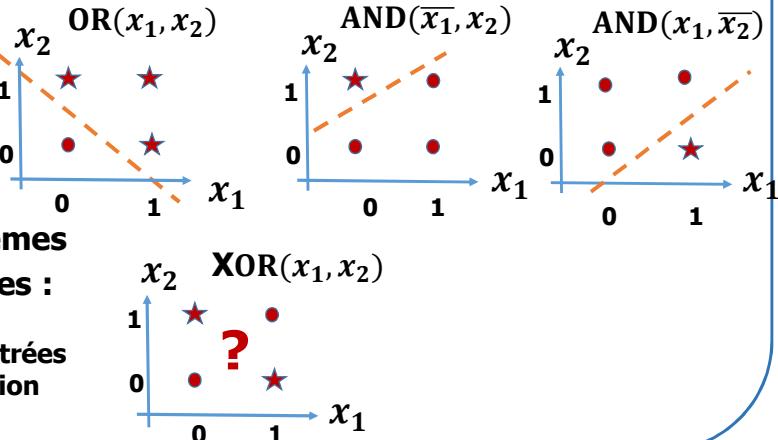


6

- ✓ Toujours positive
- ✓ Strictement croissante
- ✓ Bornée par 0 (limite inférieure)
- ✓ Pas de limite supérieure

Capacité et frontière de décision d'un seul neurone

- Peut-être utilisé pour une classification binaire (logistic regression classifier)
- Avec la fonction d'activation sigmoid, le neurone estime $p(y = 1|x)$
- Résout les problèmes linéairement séparables :
 - Frontière linéaire de décision

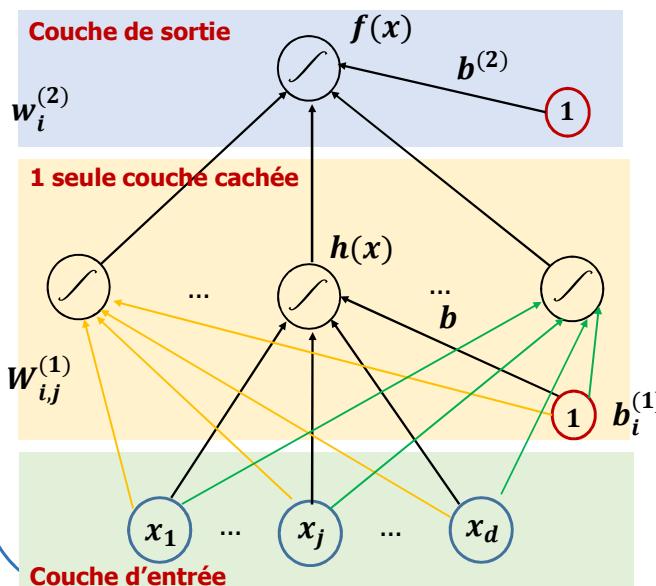


- Ne résout pas les problèmes non-linéairement séparables :

Solution : Transformer les entrées en une meilleure représentation

7

Réseau de neurones à base d'une seule couche cachée



- Pré-activation de la couche cachée :

$$\begin{aligned} a(x)_i &= b_i^{(1)} + \sum_j W_{i,j}^{(1)} x_j \\ a(x) &= \mathbf{b}^{(1)} + \mathbf{W}^{(1)} \end{aligned}$$

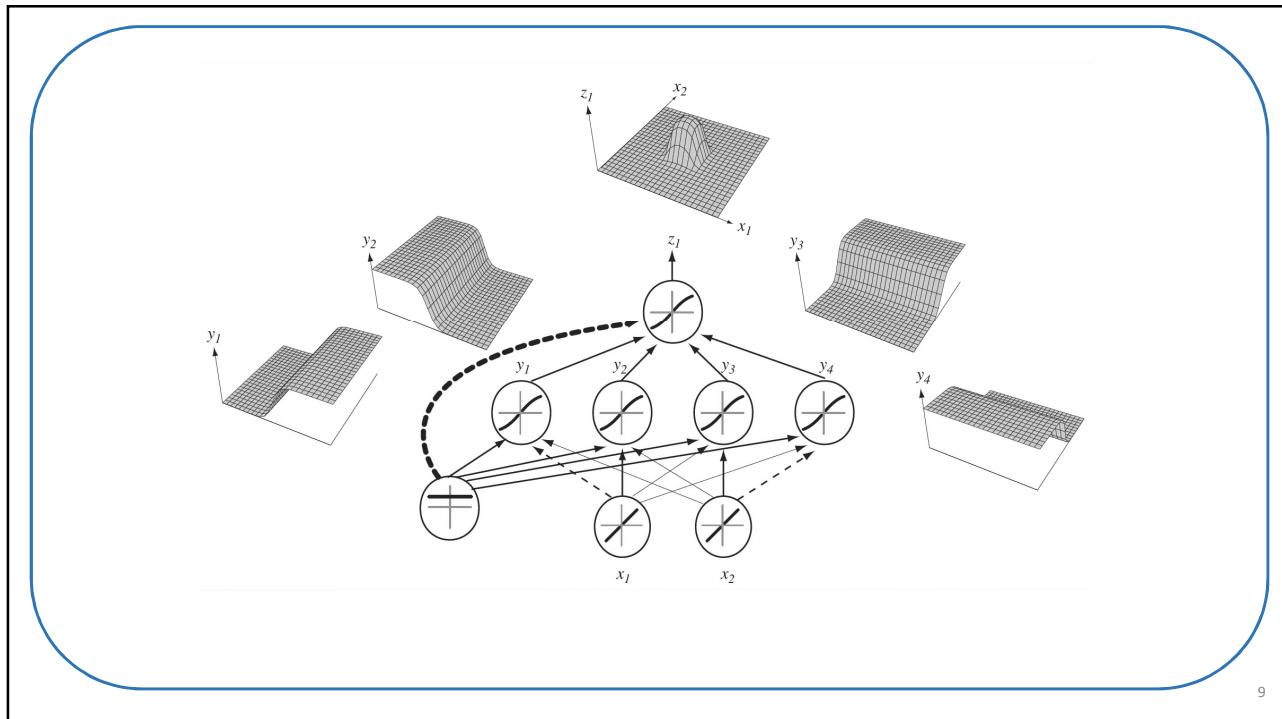
- Activation de la couche cachée :

$$\begin{aligned} h(x) &= g(a(x)) \\ g(\cdot) &: \text{Fonction d'activation} \end{aligned}$$

- Activation de la couche de sortie :

$$\begin{aligned} f(x) &= o(\mathbf{b}^{(2)} + \mathbf{W}^{(2)} \mathbf{h}^{(1)}(\mathbf{x})) \\ o(\cdot) &: \text{Fonction d'activation} \end{aligned}$$

8



9

Fonction d'activation softmax

■ Classification multiclasses :

- ✓ Plusieurs sorties (une par classe)
- ✓ Estimation de la probabilité conditionnelle : $p(y = c|x)$

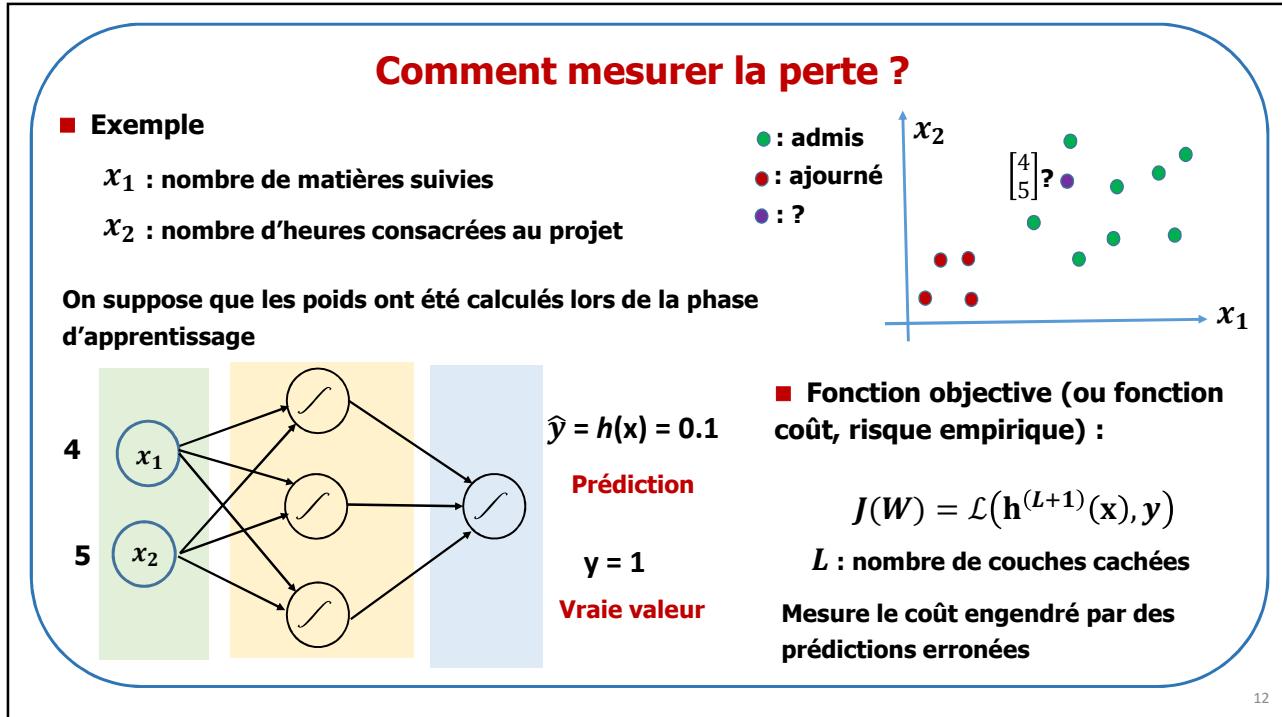
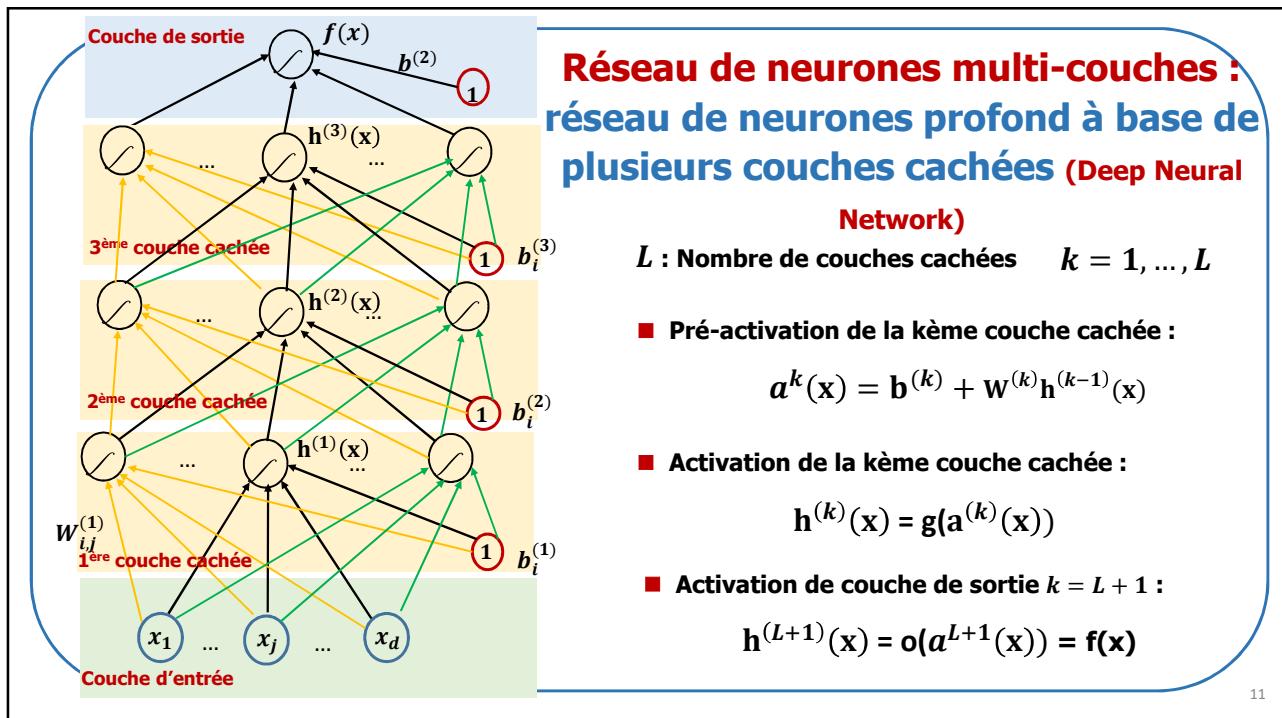
■ Activation de la fonction Softmax à la sortie :

$$o(a) = \text{softmax}(a) = \left[\frac{e^{a_1}}{\sum_c e^{a_c}} \cdots \frac{e^{a_C}}{\sum_c e^{a_c}} \right]^T$$

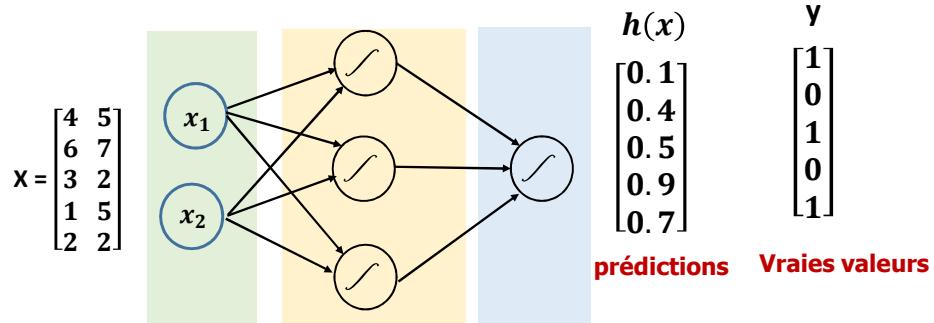
C : Nombre total de classes

- ✓ Softmax est strictement positive
- ✓ La classe prédite est celle dont la probabilité estimée est la plus élevée

10



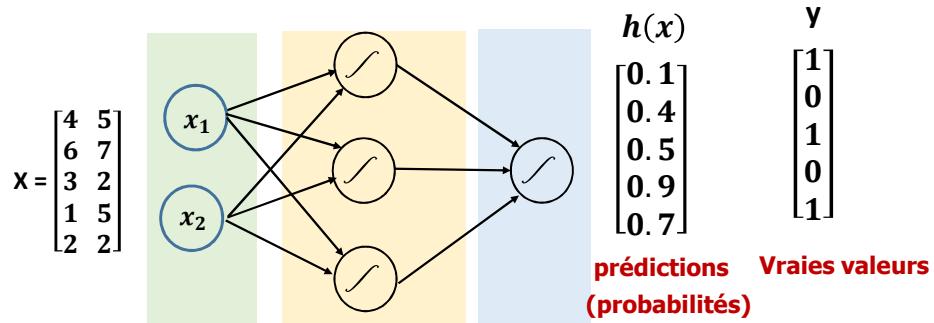
■ Mesure du coût engendré sur le jeu de données :



$$J(W) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(\mathbf{h}^{(L+1)}(x^{(i)}), y^{(i)})$$

13

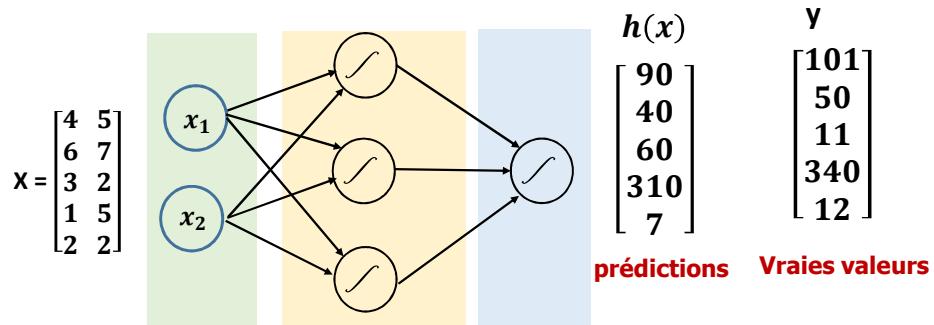
■ Binary Cross Entropy Loss :



$$J(W) = -\frac{1}{N} \sum_{i=1}^N y^{(i)} \log(\mathbf{h}^{(L+1)}(x^{(i)})) + (1 - y^{(i)}) \log(1 - \mathbf{h}^{(L+1)}(x^{(i)}))$$

14

■ Erreur quadratique moyenne :



$$J(W) = -\frac{1}{N} \sum_{i=1}^N \left(y^{(i)} - h^{(L+1)}(x^{(i)}) \right)^2$$

15

Entrainement du réseau de neurones

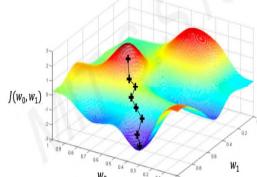
16

- Les poids de connexion du réseau de neurones, notés W^* , sont calculés de façon à minimiser la fonction objective :

$$W^* = \underset{W}{\operatorname{argmin}} J(W) = \underset{W}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^N \mathcal{L}(h^{(L+1)}(x^{(i)}, W), y^{(i)})$$

- Les poids de connexion sont déterminés par l'algorithme de descente de gradient :

Principe :



- ✓ On choisit un point de manière aléatoire
- ✓ On calcule le gradient en ce point $\frac{\partial J(W)}{\partial W}$
- ✓ On avance, d'un pas, dans la direction opposé du gradient
- ✓ On continue jusqu'à convergence

Algorithme :

Initialisation des poids de connexion au hasard ($\mathcal{N}(0, \sigma^2)$)

Répéter

Calcul du gradient $\frac{\partial J(W)}{\partial W}$

Mise à jour des poids de connexion $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$

Jusqu'à convergence

Remarques :

η : Taux d'apprentissage (Learning rate)

$\frac{\partial J(W)}{\partial W}$: peut-être très exigeant en termes de calculs !

17

Choix du taux d'apprentissage (Learning rate)

- Le taux d'apprentissage η joue un rôle important dans le processus d'apprentissage :
 - ✓ η détermine la vitesse de convergence du processus d'apprentissage
 - ✓ η module la correction : η trop faible, lenteur de convergence ; η trop élevé, oscillations
- Pour améliorer le processus d'apprentissage, on fait évoluer η au fil des itérations (fort au début pour accélérer la convergence, faible à la fin pour améliorer la précision)
- Algorithmes proposés et implémentés dans TF (Tensorflow) :

SGD	<code>tf.keras.optimizers.SGD</code>	Kiefer, Wolfowitz, Stochastic Estimation of the Maximum of a regression function. », 1952
Adam	<code>tf.keras.optimizers.Adam</code>	Kingma et al. « Adam: A Method for Stochastic Optimization », 2014.
Adadelta	<code>tf.keras.optimizers.Adadelta</code>	Zeiler et al. « ADADELTA: « An Adaptive Learning Rate Method. » 2012.
Adagrad	<code>tf.keras.optimizers.Adagrad</code>	Duchi et al. « Adaptive Subgradient Methods for Online Learnig and Stochastic Optimization. », 2011
RMSProp	<code>tf.keras.optimizers.RMSProp</code>	

Implémentation avec la librairie TensorFlow (TF)

18

Stratégies – Gradient Stochastique

- L'algorithme de descente de gradient est calculé pour chaque observation :

Initialisation des poids de connexion au hasard ($\mathcal{N}(0, \sigma^2)$)

Répéter

Choix d'une observation i

Calcul du gradient $\frac{\partial j_i(W)}{\partial W}$

Mise à jour des poids de connexion $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$

Jusqu'à convergence

- L'algorithme de descente de gradient est réalisé par lots (batch) :

- ✓ Amélioration de la convergence en réduisant le nombre de passage sur la base entière.
- ✓ Chargement partiel des données en mémoire au fur et à mesure.

Initialisation des poids de connexion au hasard ($\mathcal{N}(0, \sigma^2)$)

Répéter

Choix du lot (Batch) de B observations

Calcul du gradient $\frac{\partial J(W)}{\partial W} = \sum_{i=1}^B \frac{\partial j_i(W)}{\partial W}$

Mise à jour des poids de connexion $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$

Jusqu'à convergence

- L'algorithme de descente de gradient est réalisé par lots (mini-batch) :

- ✓ Amélioration de la convergence en réduisant le nombre de passage sur la base entière.
- ✓ Meilleure précision de l'estimation du gradient
- ✓ Chargement partiel des données en mémoire au fur et à mesure.
- ✓ Calcul en parallèle possible (rapide).

Régularisation

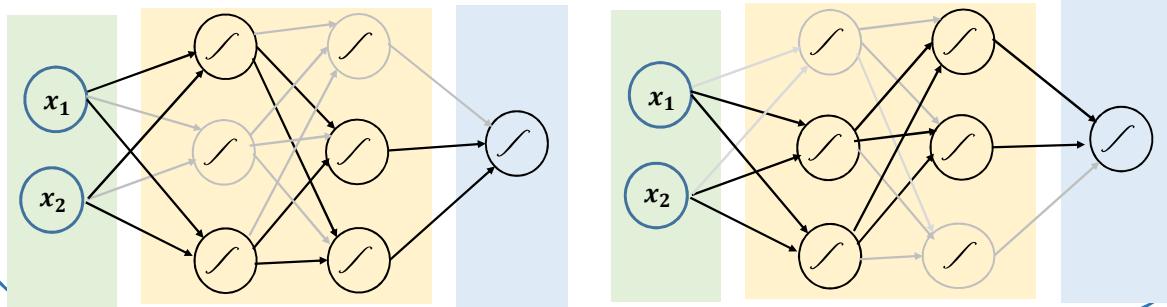
■ La régularisation est une technique qui :

- ✓ Dissuade le problème d'optimisation de choisir des modèles complexes.
- ✓ Améliore la généralisation du modèle sur les données non vues (évite l'overfitting)

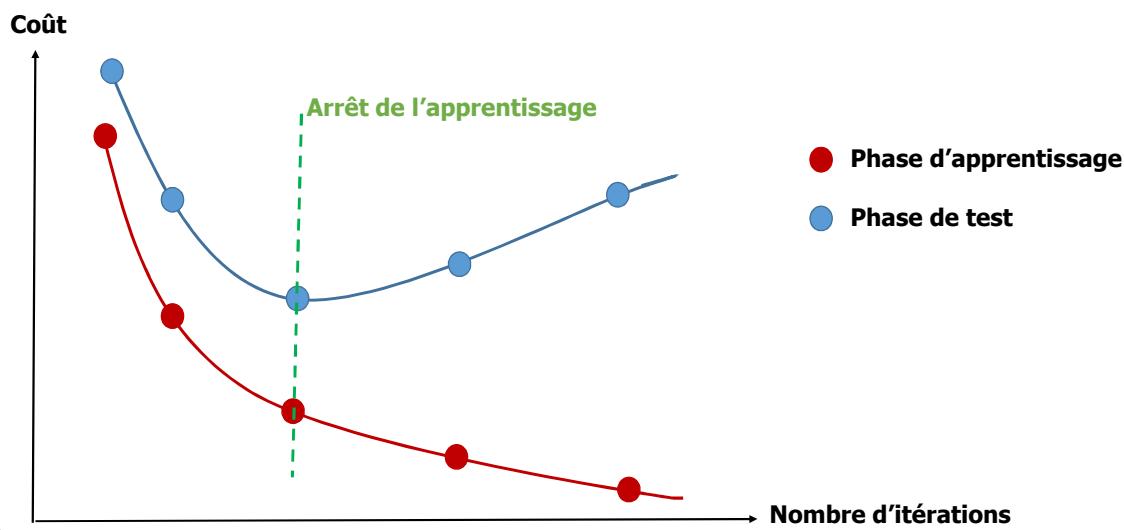
■ Stratégie 1 : Durant la phase d'apprentissage mettre quelques fonctions d'activation, choisies de manière aléatoire, à zéro

■ Examples (dropout avec TF) :

 tf.keras.layers.Dropout(p=0.5)



■ Stratégie 2 : Anticiper l'arrêt de l'apprentissage avant d'avoir un overfitting



Réseaux de neurones convolutifs

« Convolutional Neural Networks (CNN ou ConvNets) »

23

Applications des ConvNets

Détection



Classification

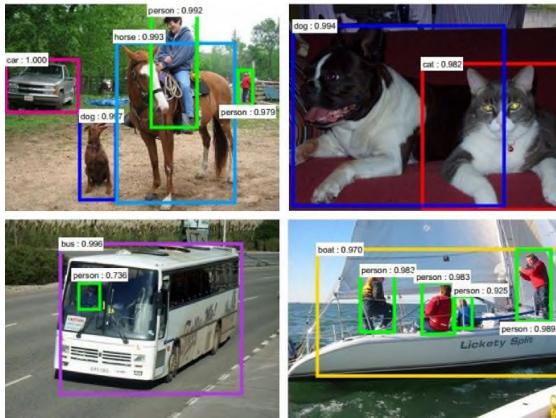


[Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012]

Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012

24

Détection

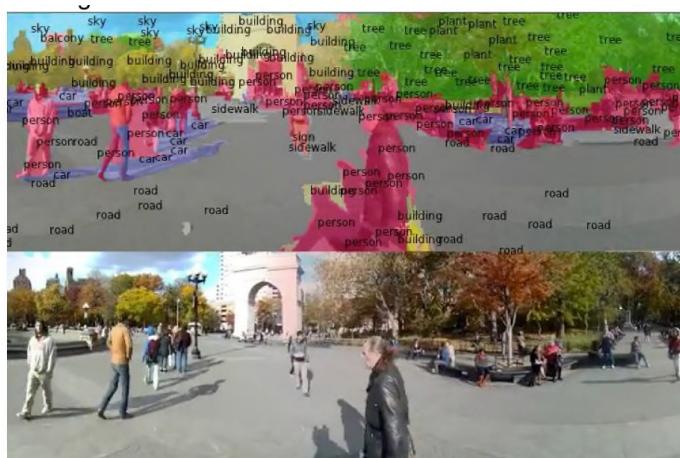


[Faster R-CNN: Shaoping Ren, Kaiming He, Ross Girshick, Jian Sun, 2015]

Figures Shaoping Ren, Kaiming He, Ross Girshick, Jian Sun, 2015

25

Segmentation



[Farabet et al., 2012]

Figures copyright Clement Farabet, 2012

26

Estimation de la pose



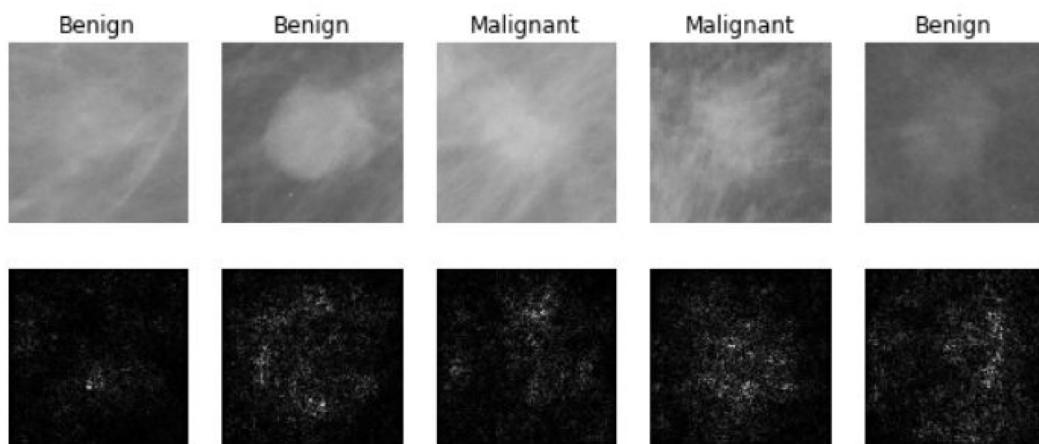
[Toshev, Szegedy 2014]



Conduite autonome

27

Images médicales - diagnostique



Figures copyright Levy et al., 2016

[Levy et al., 2016]

28

Description du contenu d'images

No errors



A white teddy bear sitting in the grass

Minor errors



A man in a baseball uniform throwing a ball

Somewhat related



A woman is holding a cat in her hand



A man riding a wave on top of a surfboard



A cat sitting on a suitcase on the floor



A woman standing on a beach holding a surfboard

Figures copyright Vinyals et al., Karpathy et al., 2015

[Vinyals et al., 2015 ; Karpathy and Fei-Fei, 2015]

29

Reconnaissance

Reconnaissance de baleines



[Kaggle Challenge]

Reconnaissance de routes

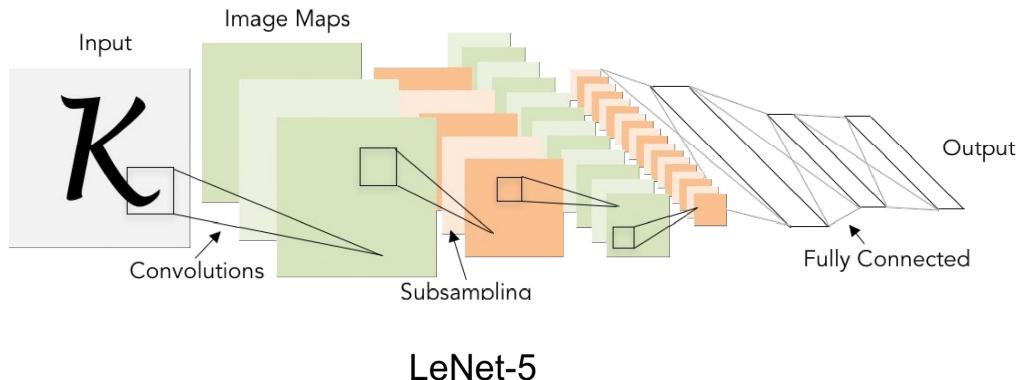


[Mnih and Hinton, 2010]

30

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]



31

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

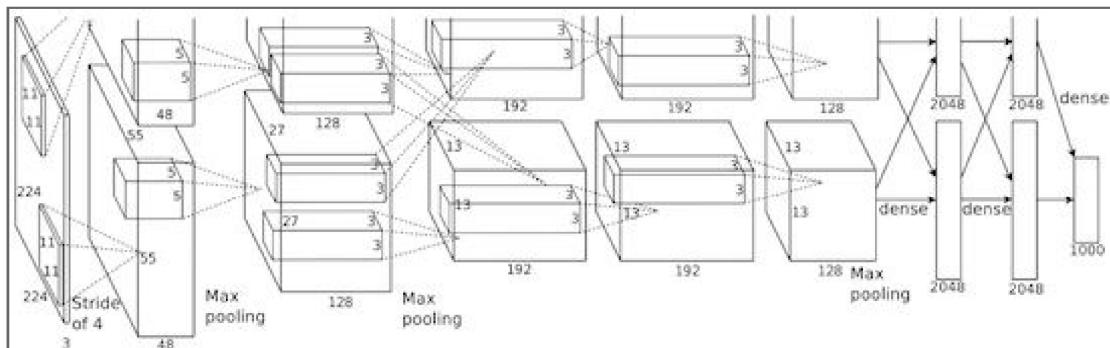


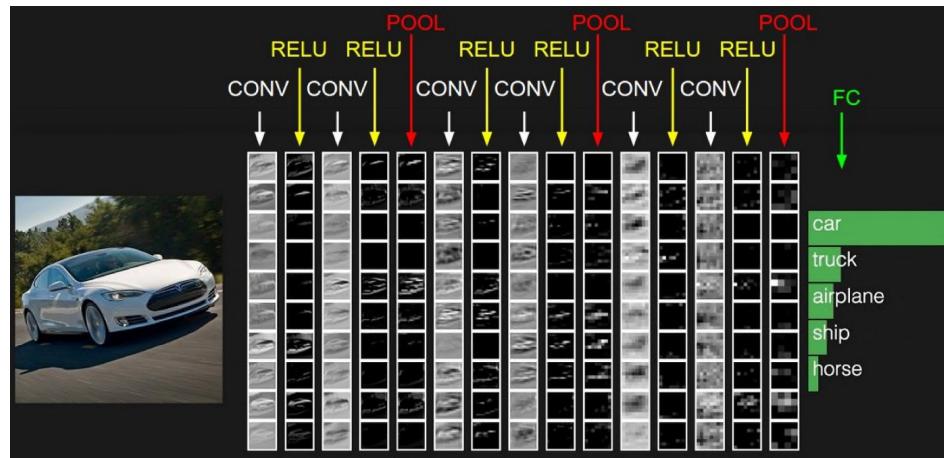
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

“AlexNet”

32

Réseaux de convolution multi-couches

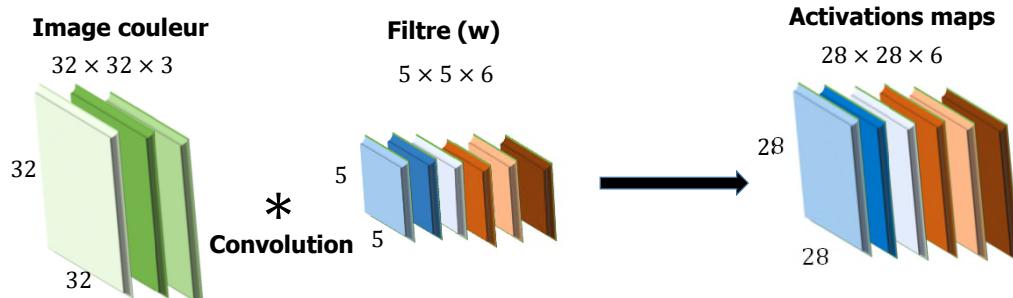
- ConvNets est un réseau de neurones formé par plusieurs couches de convolution entrecoupées par des fonctions d'activation



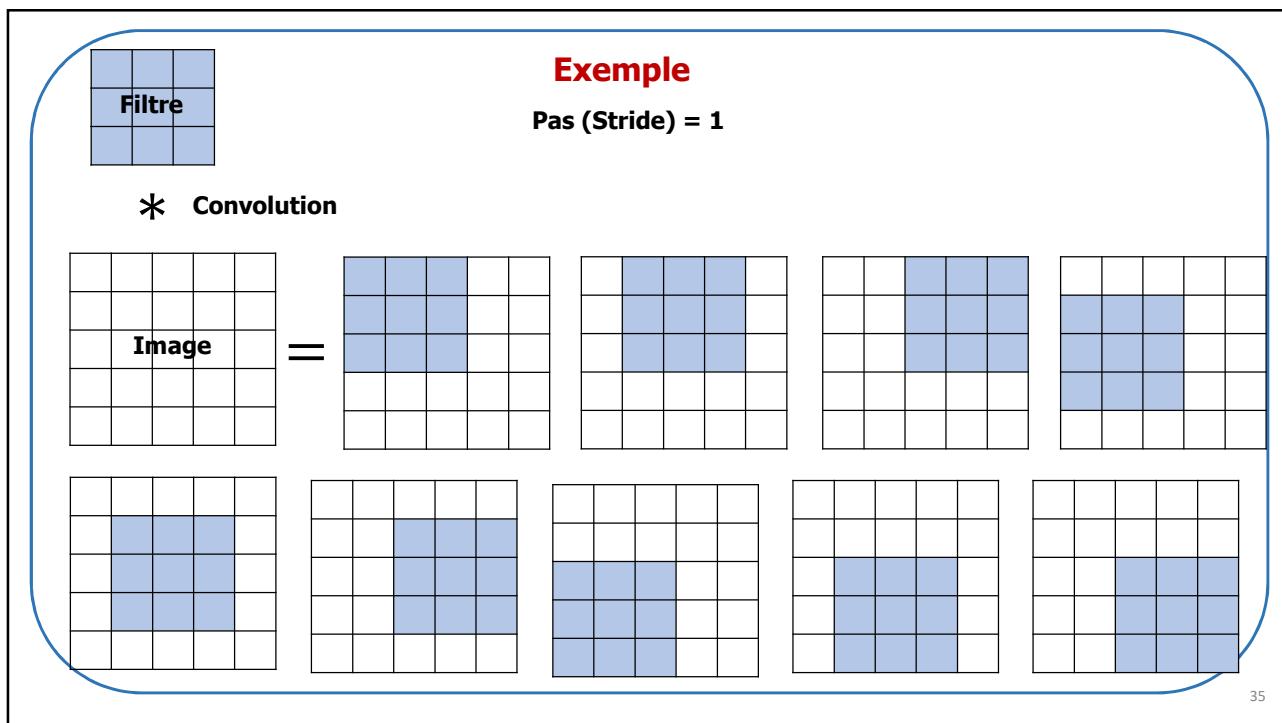
33

Couches de convolution

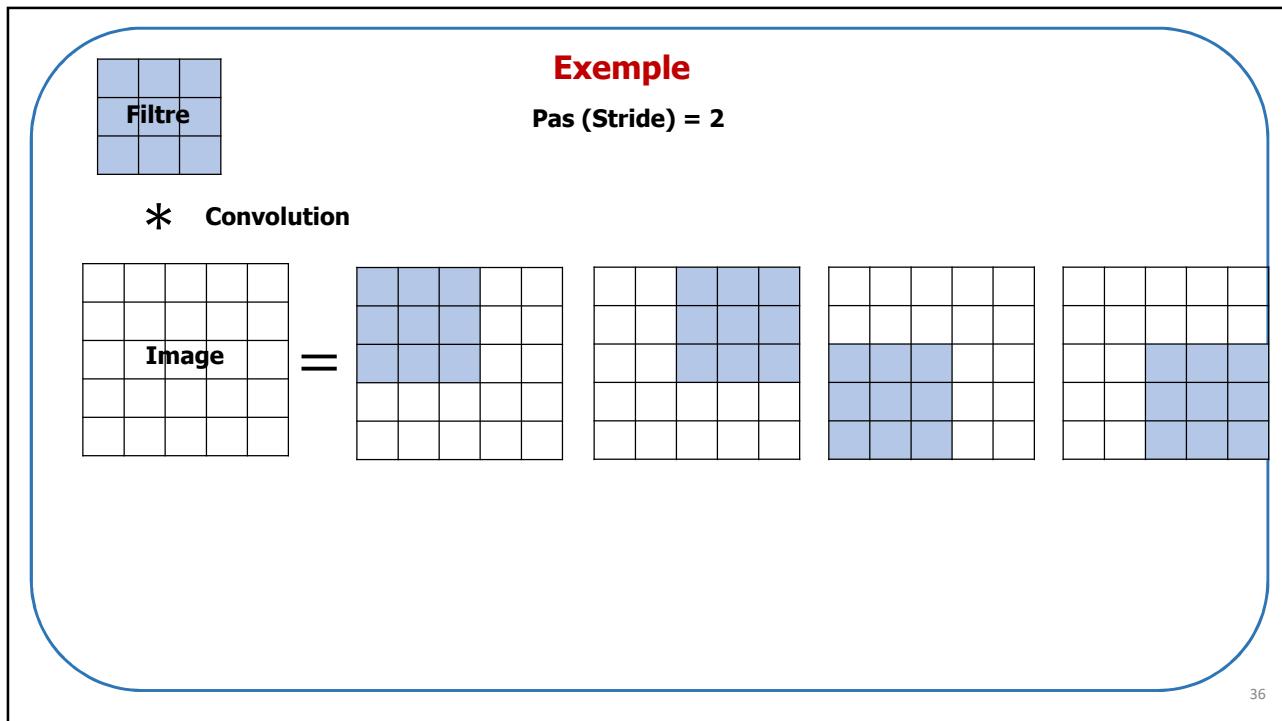
- ConvNets préserve la structure spatiale



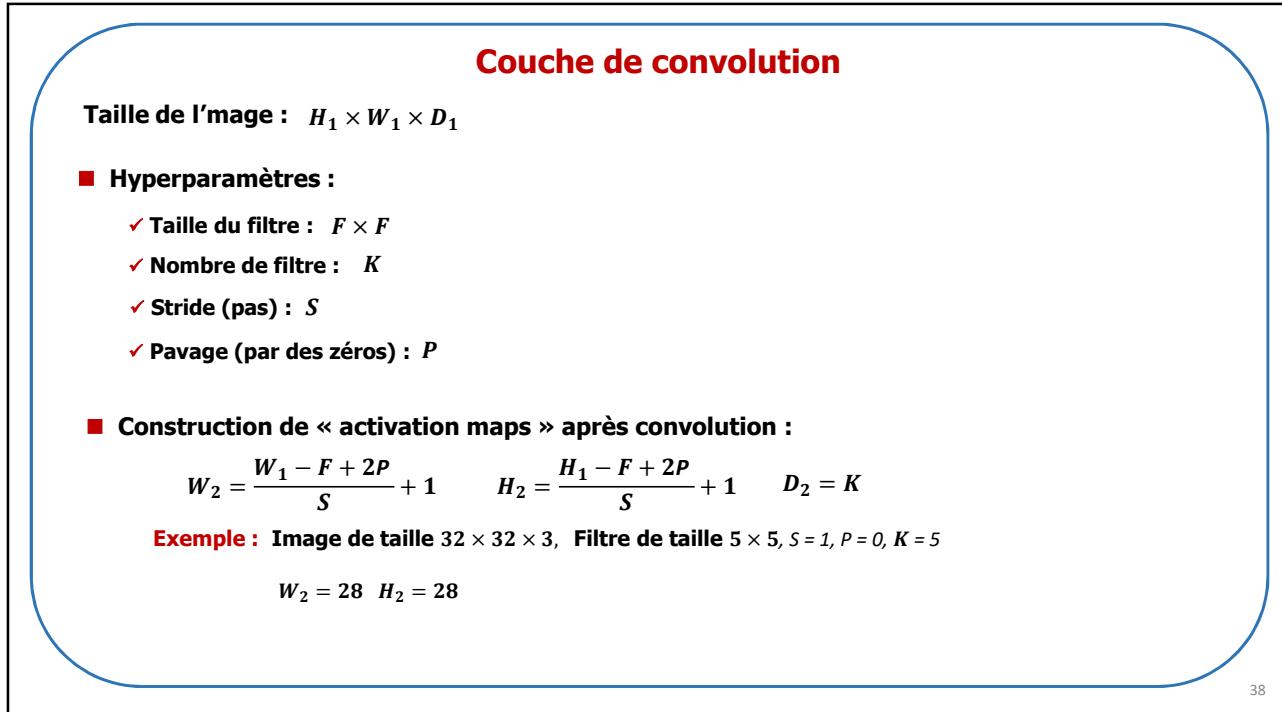
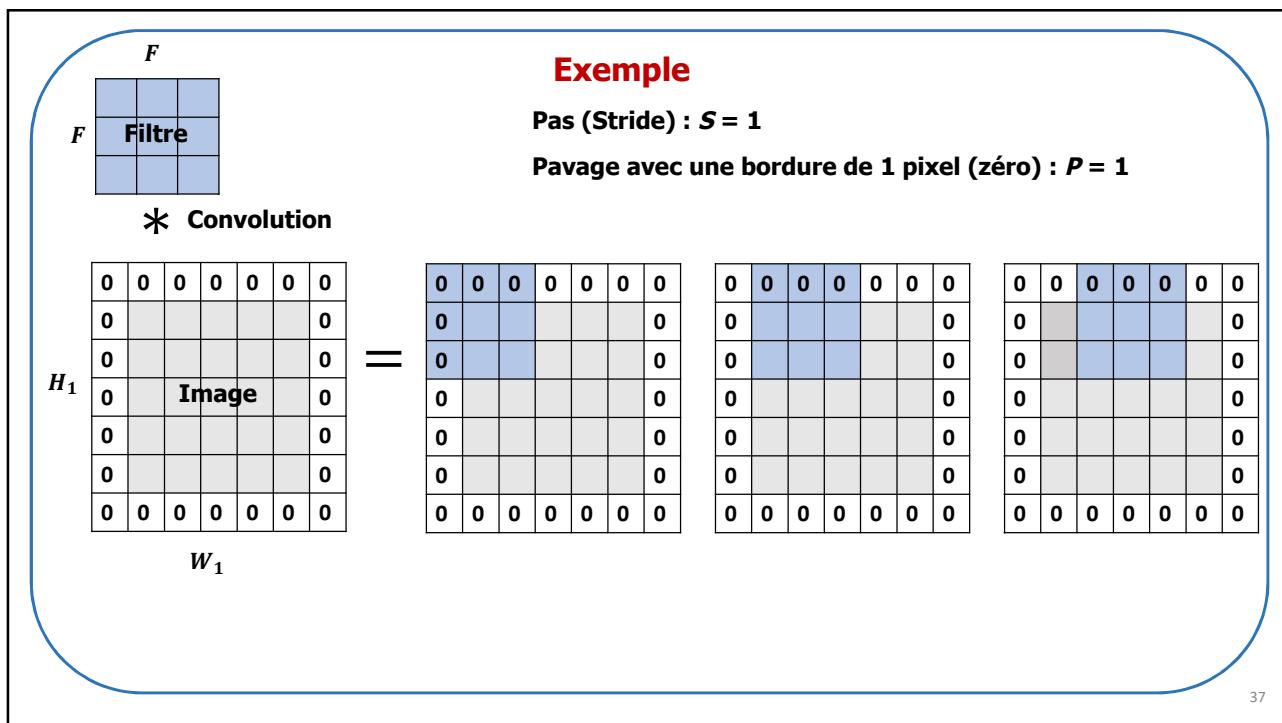
34

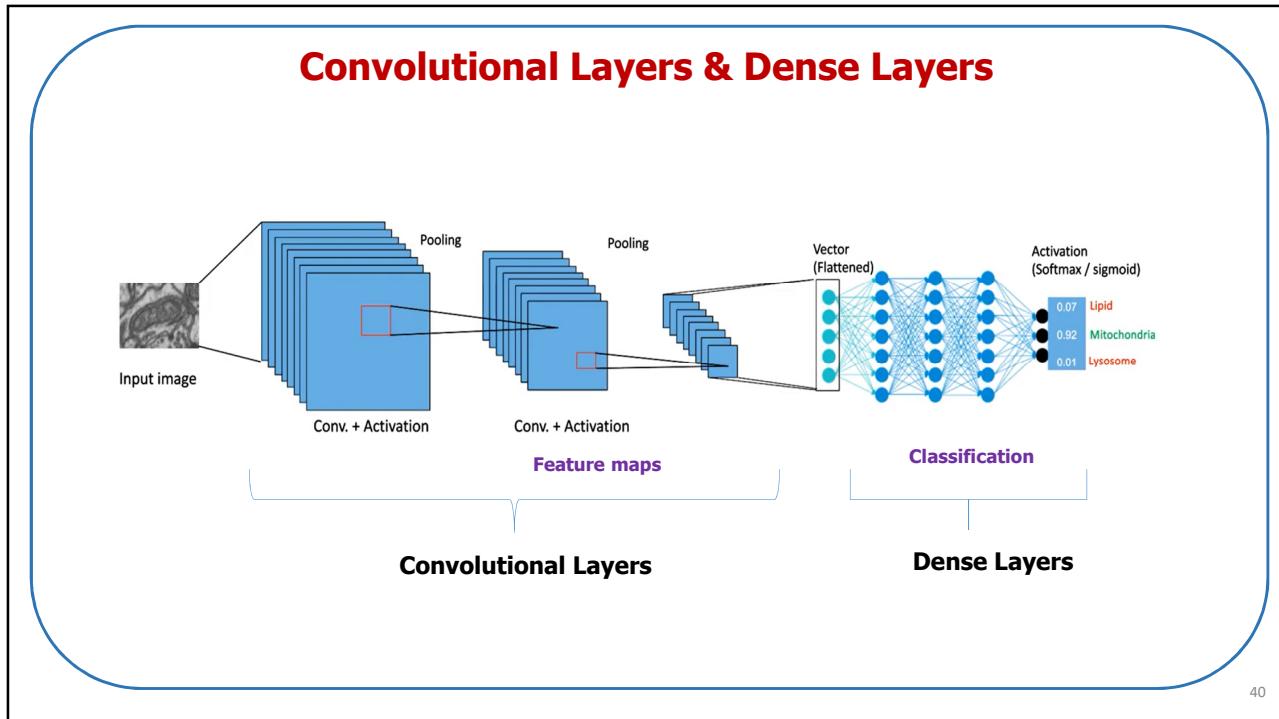
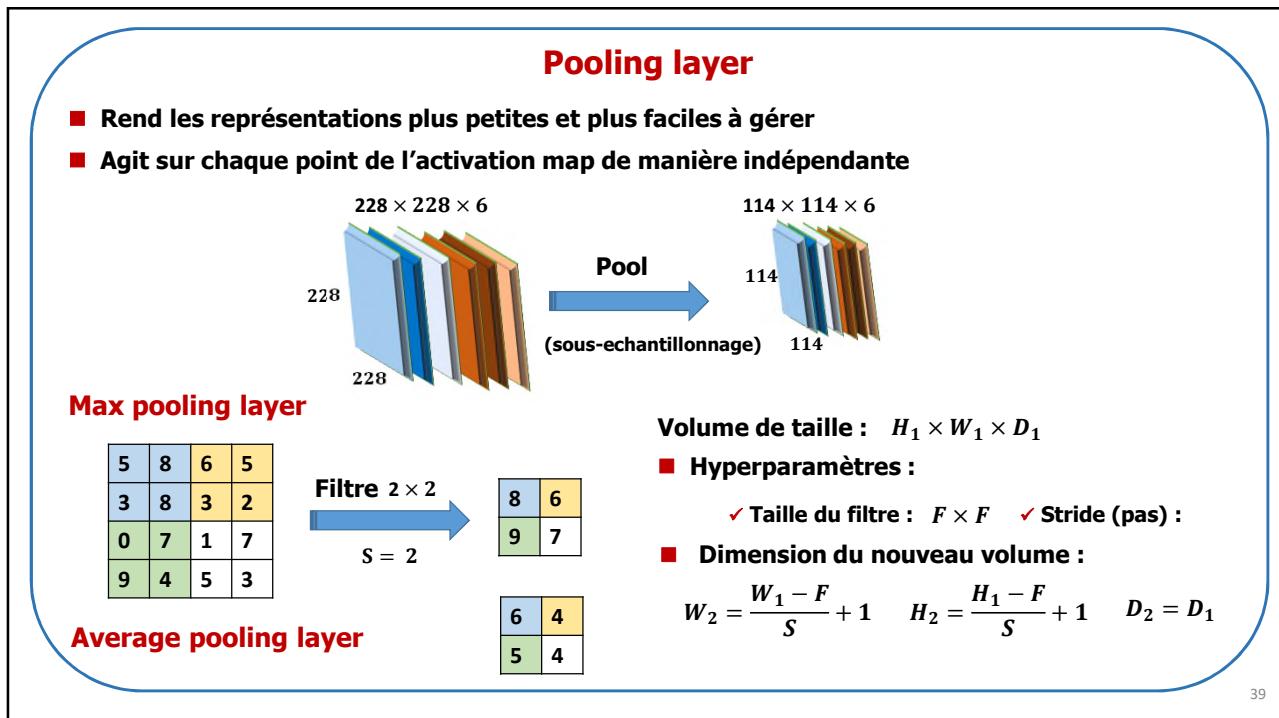


35



36





Libraires de keras :

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D
from keras.layers import MaxPooling2D, Flatten, Dropout
```

Dense Layers :

```
model = Sequential()
model.add(Dense(12, input_dim=8, activation = 'relu'))
model.add(Dense(8, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))
```

41

```
model = Sequential()
model.add(Conv2D(32, (3,3), input_shape=(28,28,1), activation = 'relu'))
model.add(MaxPooling2D(pool_size =(2,2)))
model.add(Dropout(0.2))
```

```
model.add(Conv2D(64, (3,3), activation = 'relu'))
model.add(MaxPooling2D(pool_size =(2,2)))
model.add(Dropout(0.2))
```

```
model.add(Conv2D(128, (3,3), activation = 'relu'))
model.add(MaxPooling2D(pool_size =(2,2)))
model.add(Dropout(0.2))
```

Convolutional Layers

```
model.add(Flatten())
```

```
model.add(Dense(128, (3,3), activation = 'relu'))
model.add(Dense(25, activation = 'softmax'))
```

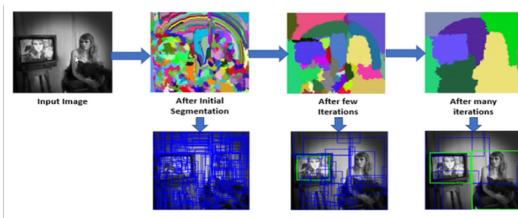
Dense Layers

```
model.compile(loss = 'categorical_crossentropy', optimizer ='adam', metric = ['acc']))
```

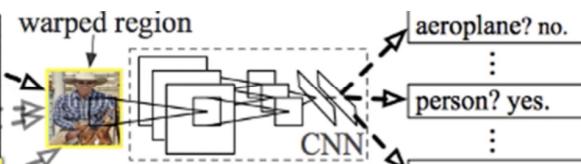
42

Exemples d'applications de détection et de localisation d'objets

Selective search

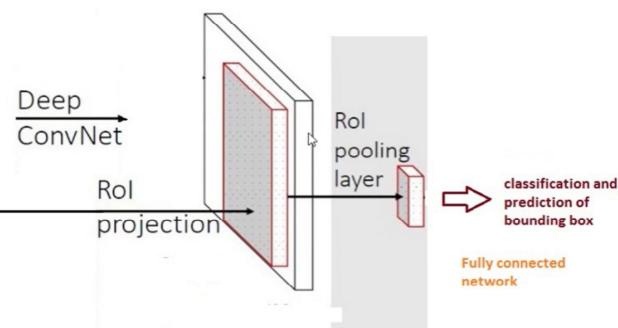


R-CNN (Region CNN)



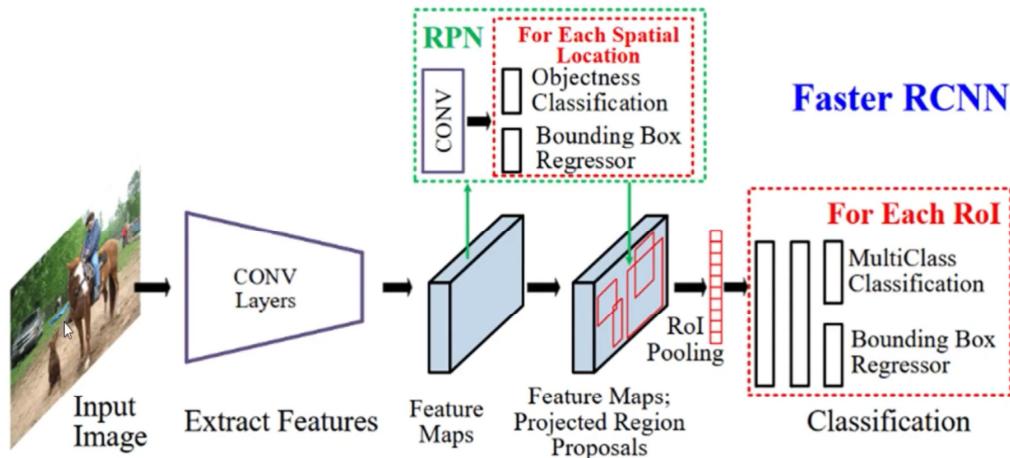
43

Fast R-CNN

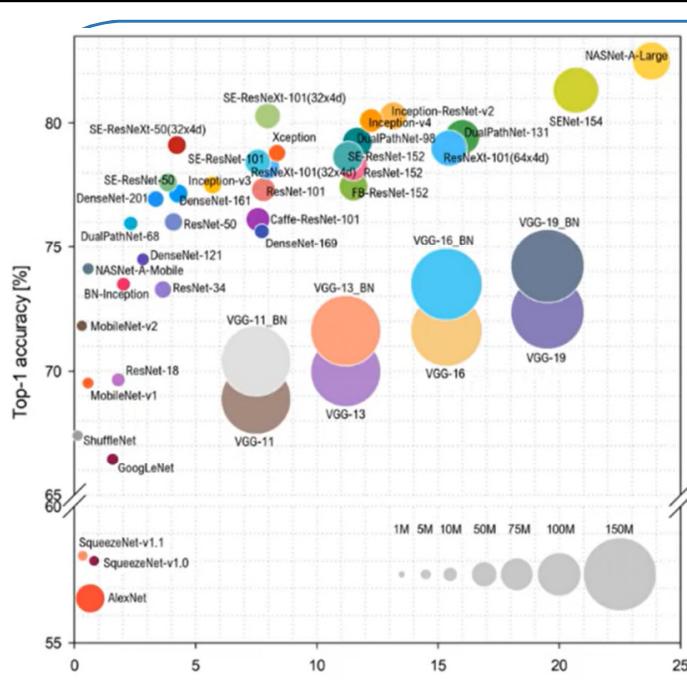


44

Faster R-CNN (avec Region proposal network)



45



- De nombreuses architectures d'apprentissage de réseaux profonds existent (entraînées sur des jeux de données publics (images))

- Les modèles pré-entraînés peuvent servir comme point de départ lorsque le modèle est entraîné sur un nouveau jeu de données : ce qui correspond à un transfert learning

46

Construction de son propre modèle d'apprentissage profond

« TP-TD 5 et 6 »

Modèle séquentiel de keras : une pile linéaire de couches

```
from keras.layers import Dense, Activation, Dropout, Flatten
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling, BatchNormalization
```

■ Première possibilité :

En créant un modèle séquentiel en passant au constructeur une liste d'instances de couches :

```
model = Sequential([
    Dense(32, input_shape=(784,), Activation('relu'),
    Dense(10), Activation('softmax'))]
```

■ Deuxième Possibilité :

En ajoutant à un modèle existant avec la méthode `add()` :

```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

47

Spécification de la forme de l'entrée :

- La première couche d'un modèle séquentiel reçoit les caractéristiques de la forme d'entrée.
- Les autres couches déterminent la forme de leurs entrées par inférence.

Comment ?

- Passez l'argument `input_shape` à la première couche (sous forme de tuple (comportant des entiers ou la valeur `None`, où `None` accepte tout entier positif)).
 - La dimension du lot n'est pas incluse dans `input_shape`
 - Si une taille de lot fixe (batch) est spécifiée pour les entrées, vous passez l'argument `batch_size`.
Exemple : Si `batch_size=32` et `input_shape=(6,8)` en argument d'une couche, le système s'attendra alors à ce que chaque lot d'entrée ait la forme de lot `(32, 6, 8)`.
- Certaines couches 2D, telles que les `Dense`, supportent la spécification de la forme de l'entrée par l'argument `input_dim`
- Certaines couches 3D temporelles supportent les arguments `input_dim` et `input_length`

48

Compilation

- Avant d'entrainer le modèle, le processus d'apprentissage doit être configuré en appelant la méthode `compile` qui accepte 3 arguments :

➤ `optimizer` : Il peut s'agir d'un optimiseur défini par son appellation, par exemple `rmsprop` ou `adagrad`, ou d'une instance de la classe Optimizer.
 ➤ `loss` : Fonction de coût que le modèle utilise pour minimiser les erreurs. Elle peut être définie par son appellation (exemple : `categorical_crossentropy`, `mse`)...
 ➤ `metrics` : une liste de métriques.

- Pour un problème de classification multiclasse :

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

- Pour un problème de régression d'erreur quadratique moyenne :

```
model.compile(optimizer='rmsprop', loss='mse')
```

- Pour un problème de classification binaire :

```
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
```

- Pour des métriques sur-mesure :

```
import keras.backend as K
def mean_pred(y_true, y_pred):
    return K.mean(y_pred)
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy', mean_pred])
```

49

Aperçu du modèle : `model.summary()`

Entrainement du modèle :

- Les modèles keras sont entraînés sur des tableaux Numpy d'entrées et des labels.
- Pour entraîner un modèle, on utilise généralement la fonction `fit`.
- Différence entre lot (batch), itération, épisode (epoch) :

Un lot (ou `batch`) d'échantillons représente un ensemble d'échantillons fixé (par exemple un ensemble d'images) qui seront propagés dans le réseau (propagation forward et backward) avant la mise à jour des paramètres du modèle pendant sa phase d'entraînement.

A titre d'exemple :

Soit un jeu de données d'entraînement composé de 3000 images. Si on fixe la taille du lot (`batch_size`) à 32 pour entraîner le réseau. Pour passer en revue les 3000 images, il faut $3000/32 = 94$ itérations, correspondant à un épisode (ou `Epoch`).

Evaluation du modèle : `score = model.evaluate(x_test, y_test, batch_size=128)`

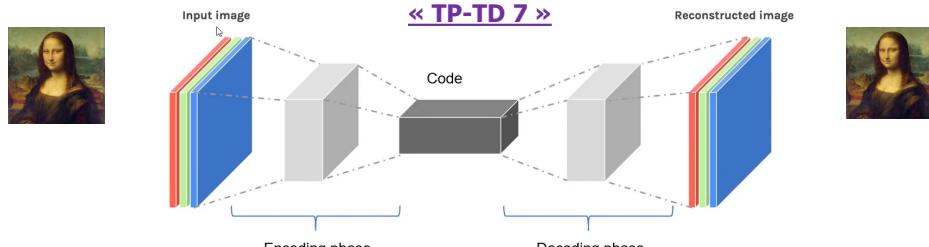
50

Comment choisir la taille du lot pour l'entraînement du modèle ?

- Un mini-lot (petite taille) est choisi lorsque les ressources de calculs sont faibles (RAM+GPU), entraînement rapide.
- Un mini-lot signifie que l'étape de descente de gradient stochastique peut être moins précise et que l'algorithme peut prendre plus de temps pour converger.
- Une dégradation significative de la qualité du modèle est possible pour des lots de taille importante (capacité à généraliser moins bonne).
- Un lot de taille 32 ou 64 est un bon point de départ.

51

Autoencodeurs



■ Réseaux de neurones :

Encodeur : model.add(Conv2D(8, (3,3), activation ='relu', padding='same')
model.add(MaxPooling2D, (2,2), padding = 'same'))

Décodeur : model.add(Conv2D(8, (3,3), activation ='relu', padding='same')
model.add(UpSampling2D, (2,2)))

Entrainement : model.fit(X,X, epochs =100, shuffle ='True')

■ Plusieurs applications :

Compression, débruitage, détection d'anomalie, coloriage d'images ...

52

Réseaux de neurones récurrents

« Recurrent Neural Networks (RNN) »

53

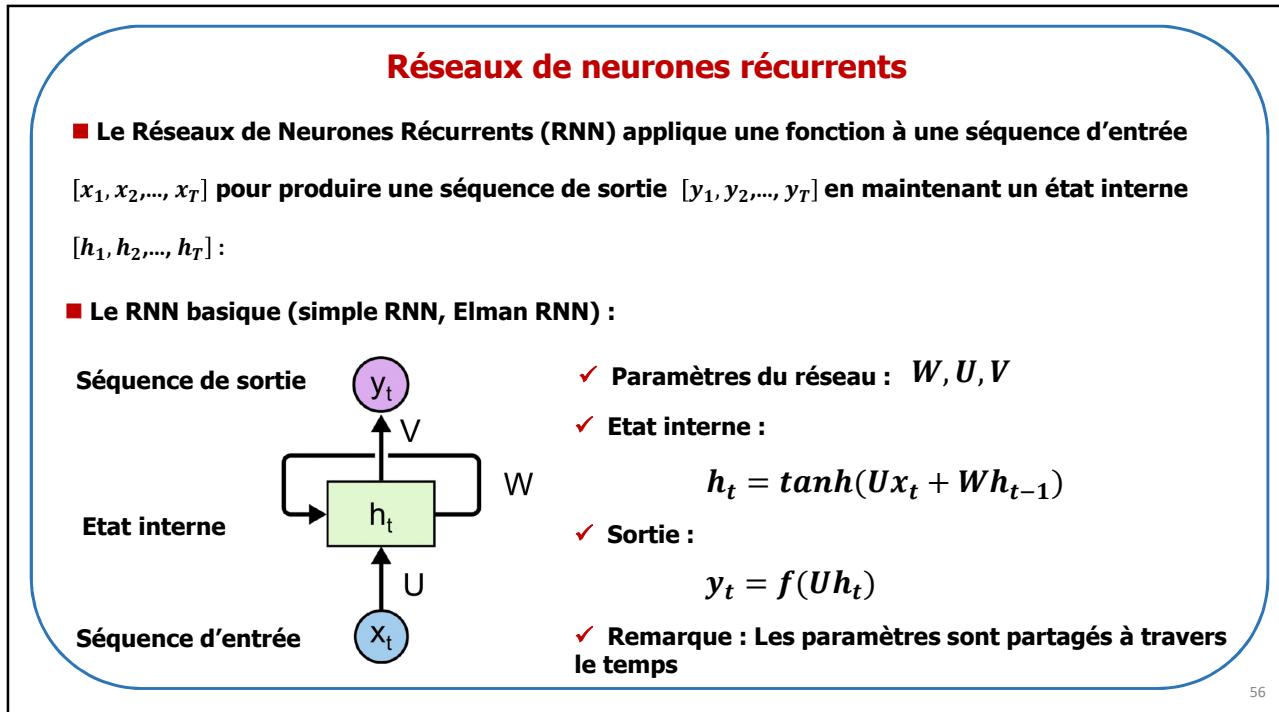
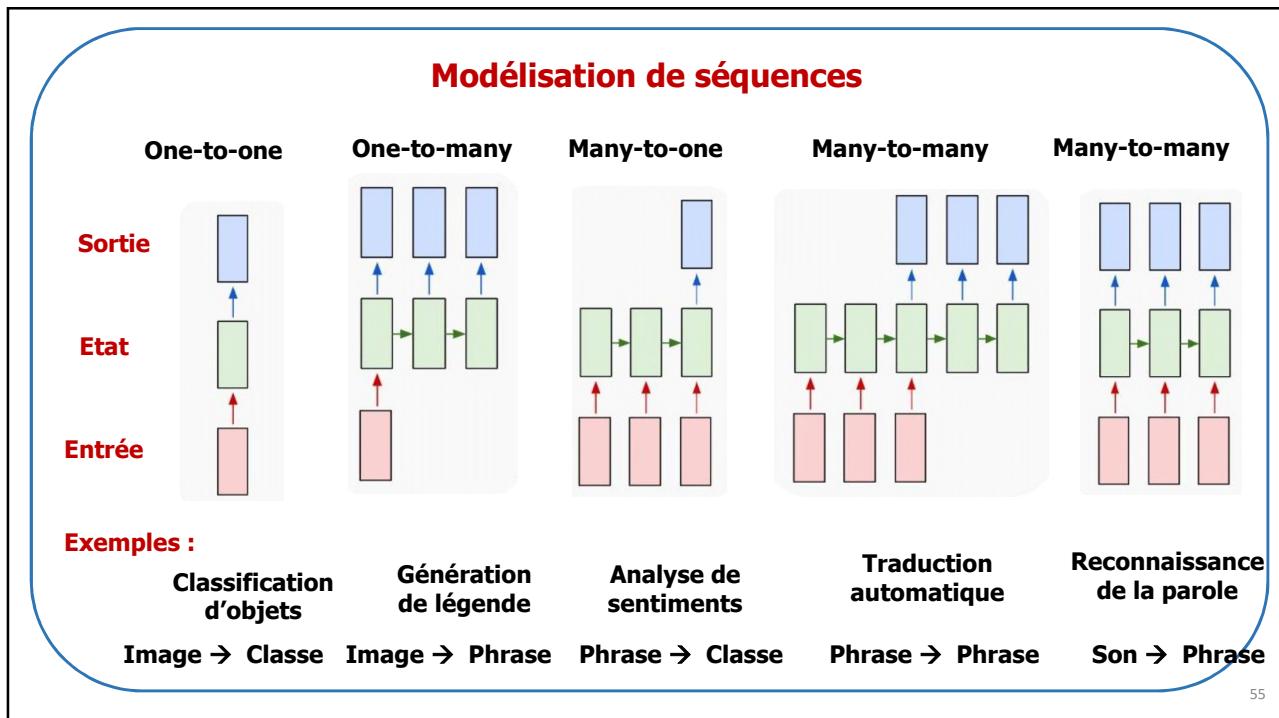
Motivation

Plusieurs exemples de données séquentielles :

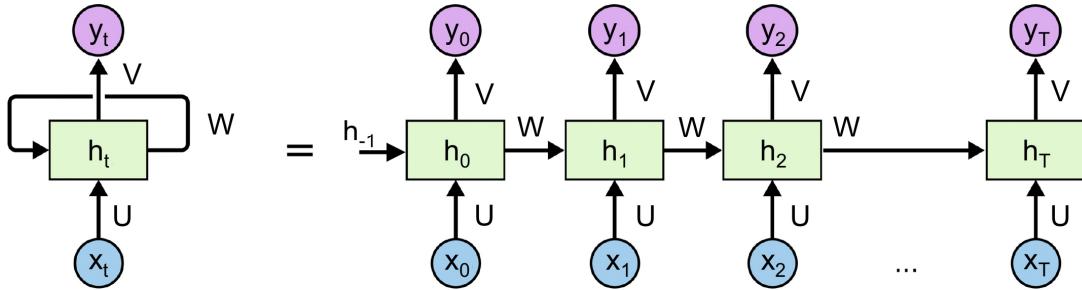
- Audio : ✓ Traduction automatique (taille différente de phrase)
✓ Génération de la parole
✓ Reconnaissance de la parole (séquence audio produit séquence de mots)
- Vidéo : ✓ Génération de légendes
✓ Reconnaissance d'actions
- Texte : ✓ Classification d'emails
✓ Traduction automatique (séquence de mot produit séquence de mots)
- Données biologiques : ✓ Etude de l'ADN
✓ IRM
✓ Electrocardiogramme
- Séries financières :
- etc.

Il existe énormément de données de tailles variables

54

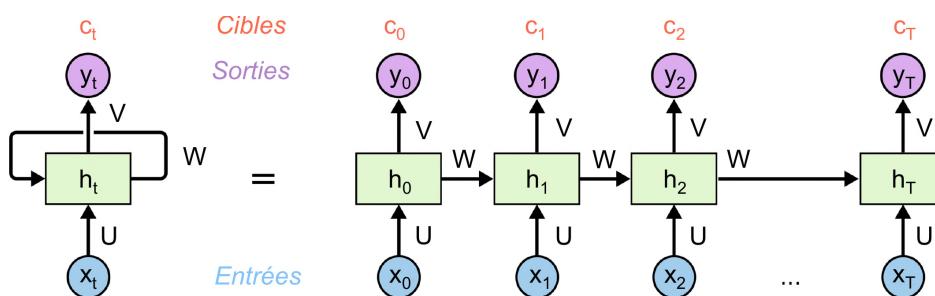


■ Le RNN basique (simple RNN, Elman RNN) :



57

Entrainement des RNN



■ Erreur globale :

✓ Somme de l'erreur à chaque temps : $E = \sum_{t=0}^T E_t(y_t, c_t)$

Où E_t peut être l'erreur quadratique moyenne, entropie croisée, ...

58

Rétropropagation à travers le temps

■ Rétropropagation classique pour adapter les paramètres :

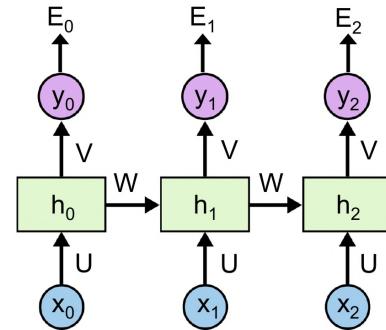
■ Calcul de l'erreur globale :

$$E = \sum_{t=0}^T E_t(y_t, c_t)$$

■ Calculs des gradients sur U, V, W :

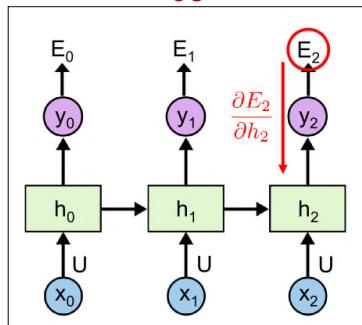
$$\frac{\partial E}{\partial U} = \sum_{t=0}^T \frac{\partial E_t}{\partial U_t} \quad \frac{\partial E}{\partial V} = \sum_{t=0}^T \frac{\partial E_t}{\partial V_t}$$

$$\frac{\partial E}{\partial W} = \sum_{t=0}^T \frac{\partial E_t}{\partial W_t}$$



59

■ Calcul de $\frac{\partial E_2}{\partial U}$

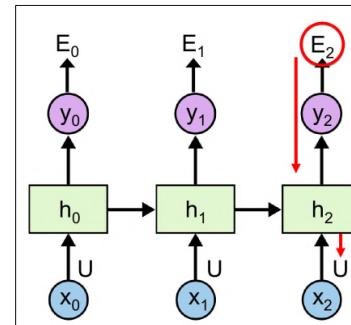


✓ Calculer d'abord : $\frac{\partial E_2}{\partial U}$

✓ Propager l'erreur jusqu'à

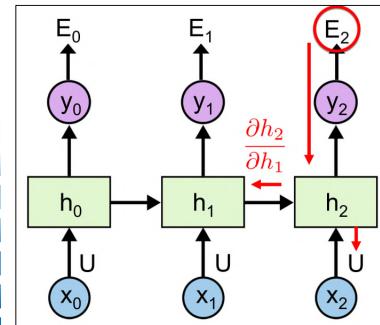
l'état interne : h_2

$$\frac{\partial E_2}{\partial U} = \frac{\partial E_2}{\partial h_2} \dots$$



✓ Propager l'erreur sur U au 2ème pas de temps :

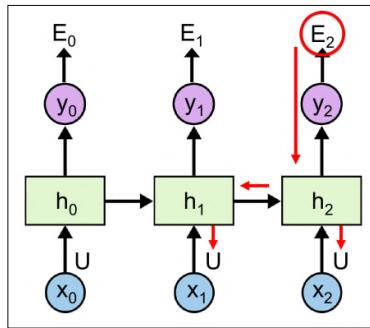
$$\frac{\partial E_2}{\partial U} = \frac{\partial E_2}{\partial h_2}(x_2^T \dots)$$



✓ Propager l'erreur sur U au 1er pas de temps :

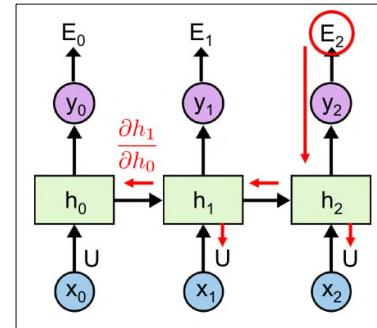
$$\frac{\partial E_2}{\partial U} = \frac{\partial E_2}{\partial h_2}(x_2^T + \frac{\partial h_2}{\partial h_1}(\dots))$$

60



- ✓ Propager l'erreur sur U au 1^{er} pas de temps :

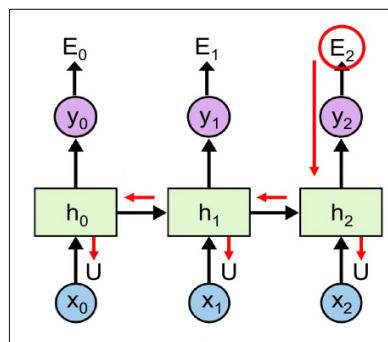
$$\frac{\partial E_2}{\partial U} = \frac{\partial E_2}{\partial h_2} (x_2^T + \frac{\partial h_2}{\partial h_1} (x_1^T + \dots)$$



- ✓ U intervient également dans le calcul de h_0
- ✓ Rétropropager l'erreur à travers le temps :

$$\frac{\partial E_2}{\partial U} = \frac{\partial E_2}{\partial h_2} (x_2^T + \frac{\partial h_2}{\partial h_1} (x_1^T + \frac{\partial h_1}{\partial h_0} \dots$$

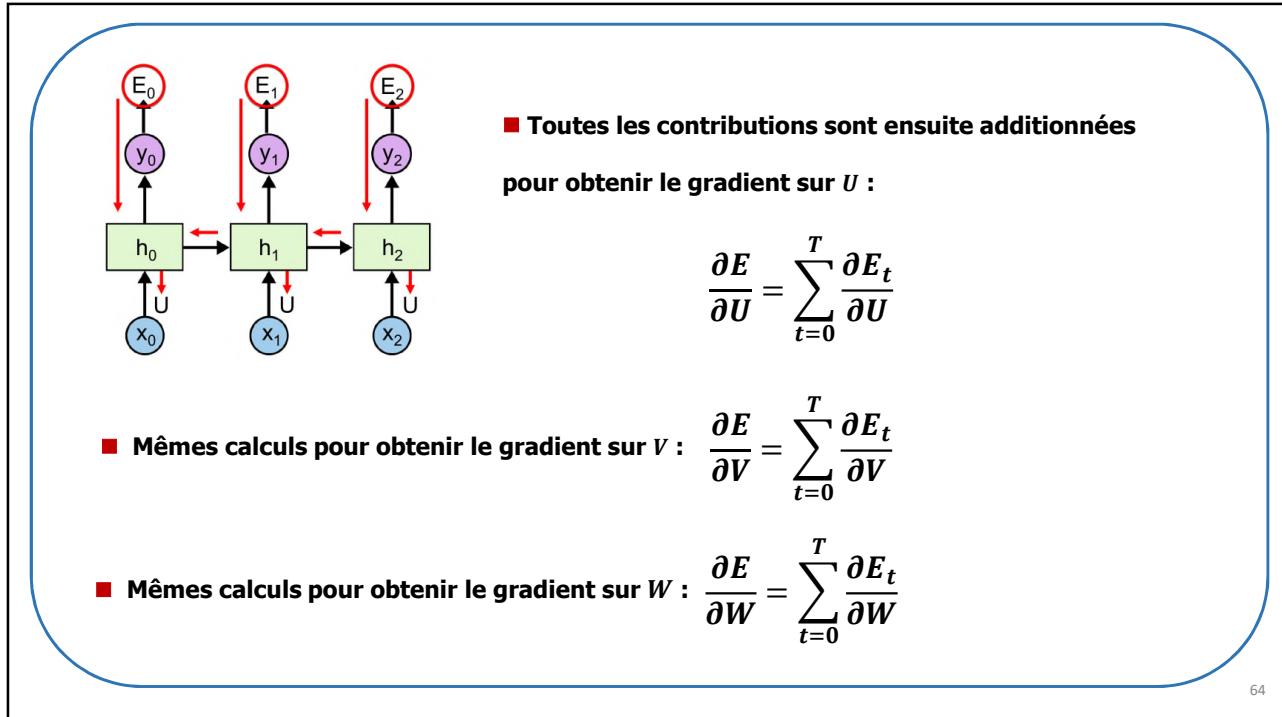
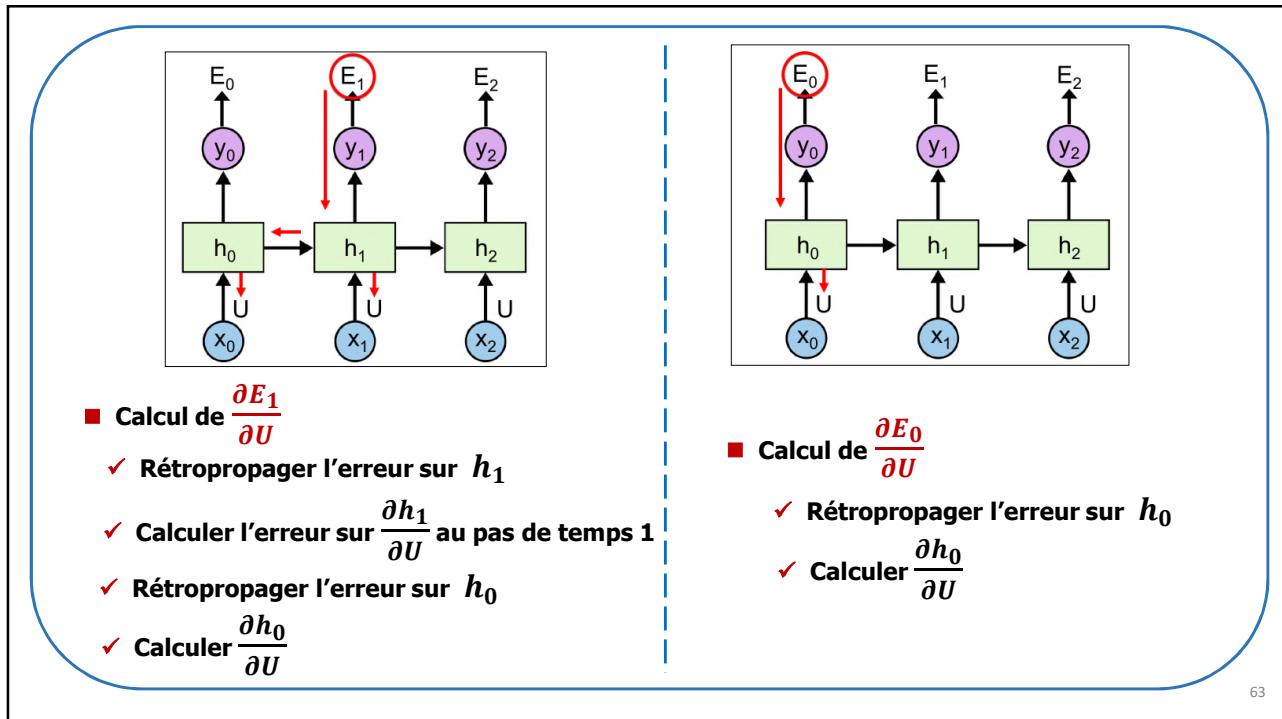
61



- ✓ Rétropropagation complète de $\frac{\partial E_2}{\partial U}$:

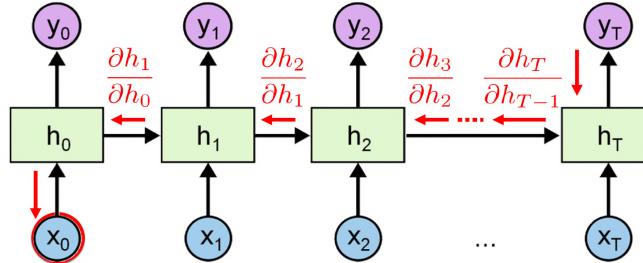
$$\frac{\partial E_2}{\partial U} = \frac{\partial E_2}{\partial h_2} \left(x_2^T + \frac{\partial h_2}{\partial h_1} \left(x_1^T + \frac{\partial h_1}{\partial h_0} x_0^T \right) \right)$$

62



Quelques difficultés d'apprentissage

■ Apprendre des dépendances à long terme : propager le gradient loin dans le temps



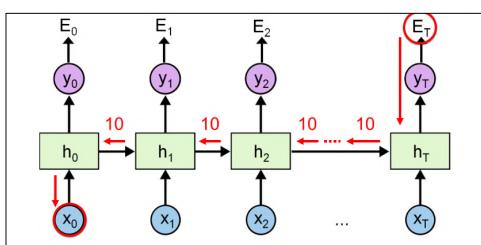
■ La propagation des gradients à travers de nombreux pas de temps peut devenir instable et créer des problèmes d'entraînement :

$$\frac{\partial y_T}{\partial x_0} = \frac{\partial y_T}{\partial h_T} \frac{\partial h_T}{\partial h_{T-1}} \cdots \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial h_0} \frac{\partial h_0}{\partial x_0}$$

65

Problèmes

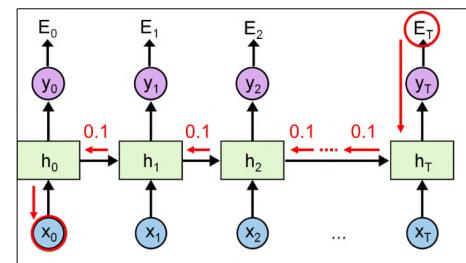
- ✓ Explosion du gradient (amplifié à chaque temps)
- ✓ Risque de divergence des paramètres



■ Solution : Clipping gradient

$$g = \frac{\partial E}{\partial W} \quad \text{If } \|g\| \geq \text{seuil} \text{ then} \\ g \leftarrow \frac{\text{seuil}}{\|g\|} g \\ \text{end if}$$

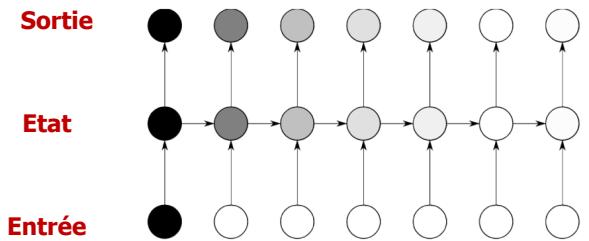
- ✓ Dissipation du gradient (atténuation à chaque temps)
- ✓ Empêche l'apprentissage de dépendance à long terme



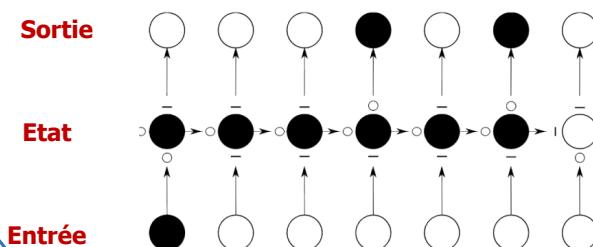
■ Solution : Architecture RNN particulière

66

Architecture RNN



■ **Manque de mémoire :** RNN oublie peu à peu la première entrée

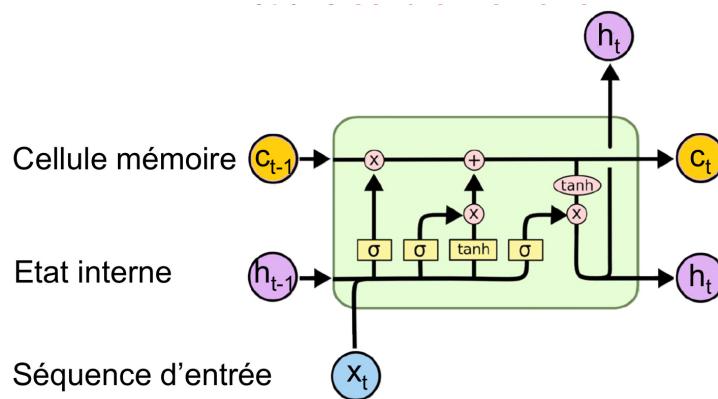


■ **Idée :** Introduction de 3 gates (ouvert, - fermé) pour contrôler l'entrée, la sortie et l'efficacement de l'état pour retenir et propager l'information dans le temps

67

Long Short-Term Memory (LSTM)

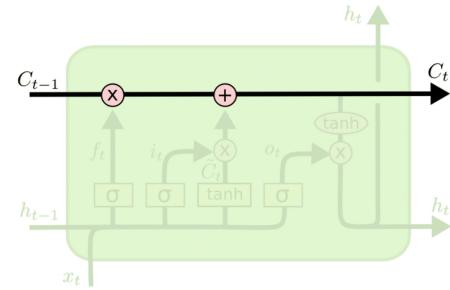
■ Réduction du problème de dissipation avec un mécanisme de gates et une cellule mémoire :



68

■ Cellule mémoire de l'architecture LSTM

- ✓ Point clé de l'architecture LSTM
- ✓ Très peu d'opérations
- ✓ L'information est transmise très simplement



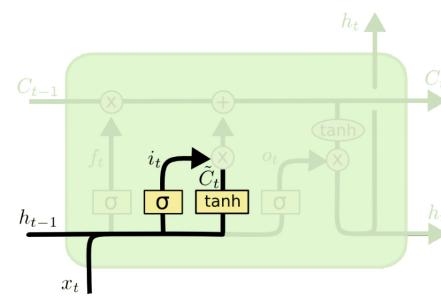
■ Calcul de l'input gate

- ✓ L'input gate i_t contrôle ce qui entre dans la cellule mémoire :

$$i_t = \sigma(U_i x_t + W_i h_{t-1} + b_i)$$

- ✓ Calcul de ce qui va être ajouté à la cellule mémoire :

$$g_t = \tanh(U_g x_t + W_g h_{t-1} + b_g)$$



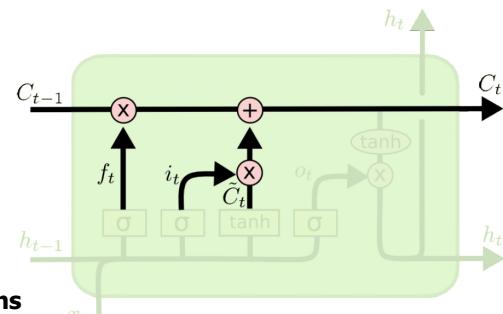
69

■ Calcul de la cellule mémoire :

- ✓ Mise à jour de la cellule mémoire à l'aide de l'input i_t et de la forget gate f_t :

$$c_t = i_t \odot g_t + f_t \odot c_{t-1}$$

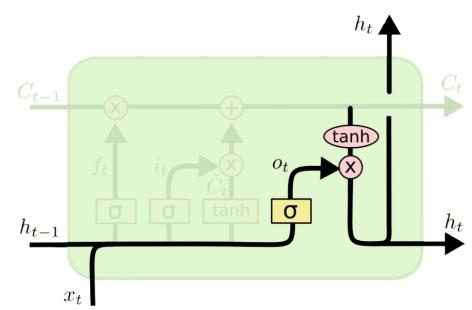
⊗ multiplication terme à terme



- ✓ L'input gate permet d'ajouter de l'information dans la cellule et la forget gate permet d'oublier l'information déjà présente dans la cellule

■ Calcul de l'output gate :

- ✓ L'output gate contrôle ce qui sort de la cellule mémoire : $o_t = \sigma(U_o x_t + W_o h_{t-1} + b_o)$

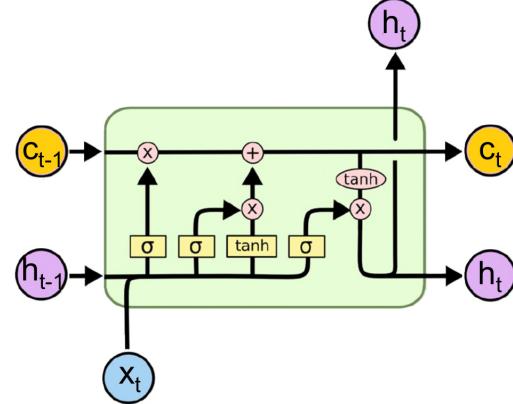


- ✓ Calcul de l'état interne : $h_t = o_t \odot \tanh(c_t)$
⊗ multiplication terme à terme

70

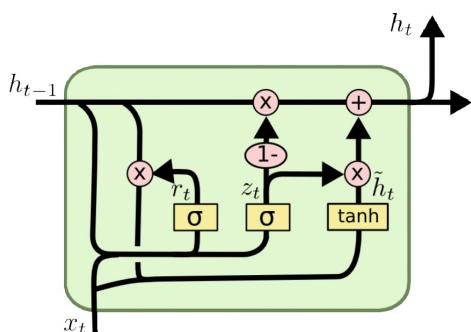
■ Résumé de la stratégie LSTM :

$$\begin{aligned}
 i_t &= \sigma(U_i x_t + W_i h_{t-1} + b_i) \\
 f_t &= \sigma(U_f x_t + W_f h_{t-1} + b_f) \\
 o_t &= \sigma(U_o x_t + W_o h_{t-1} + b_o) \\
 g_t &= \tanh(U_g x_t + W_g h_{t-1} + b_g) \\
 c_t &= i_t \odot g_t + f_t \odot c_{t-1} \\
 h_t &= o_t \odot \tanh(c_t)
 \end{aligned}$$



71

Gated Recurrent Unit (GRU)



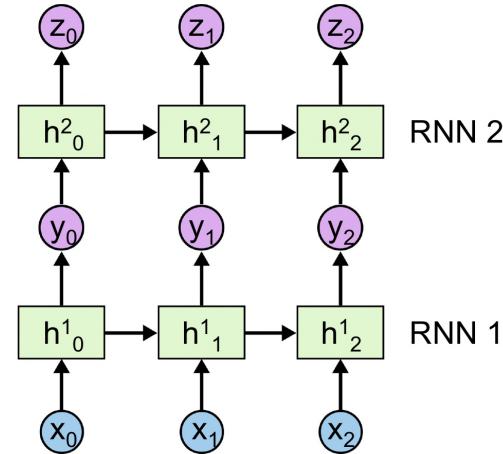
- Variante populaire de l'architecture LSTM
- Pas de cellule mémoire explicite
- Input et forget gates combinées
- Performance similaire au réseau LSTM
- Plus rapide à calculer

$$\begin{aligned}
 z_t &= \sigma(U_z x_t + W_z h_{t-1} + b_z) \\
 r_t &= \sigma(U_r x_t + W_r h_{t-1} + b_r) \\
 g_t &= \tanh(U_g x_t + W_g (r_t \odot h_{t-1}) + b_g) \\
 h_t &= z_t \odot g_t + (1 - z_t) \odot h_{t-1}
 \end{aligned}$$

72

Piles de réseaux récurrents : RNN profonds

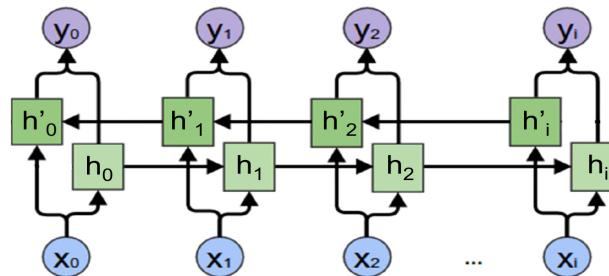
- La séquence de sortie de la première couche du RNN est la séquence d'entrée de la seconde couche d'entrée du RNN



73

Réseaux récurrents bidirectionnels

- Deux réseaux RNN (différents paramètres) :



- Un second réseau RNN pour la lecture de la séquence en sens inverse
- Permet d'avoir de l'information sur ce qui se passe avant et après

74