Deep learning

cours M2 Dauphine
(M2 ISF app 2019/20+, M2 Math 2023/24+)
Université Paris Dauphine - PSL

Gabriel Turinici

CEREMADE, Université Paris Dauphine - PSL

Deep learning: actualité, applications et techniques

Exercice:

- par groupes de 4-6 étudiants
- expose de 3-4 minutes présentant le D.L. : 1-2 applications, 1-2 techniques importantes

Introduction: références

Mes travaux et domaines d'intérêt

- Algorithmes SGD d'optimisation stochastique adaptatifs: lien présentation générale, pdf https://arxiv.org/abs/2002.09304
- Convergence des GAN https://arxiv.org/abs/2012.10410
- DL non supervisé (IA génératif): **rapport « Deepfakes & Algorithmes: Menace ou Opportunité ?** (https://zenodo.org/record/4264371)
- IA génératif : VAE et distances de probabilité Radon-Sobolev: https://arxiv.org/abs/1911.13135

- IA génératif : diversité et quantization

- reinforcement learning : portfolio allocation (arXiv:2312.05169), policy gradient algorithms

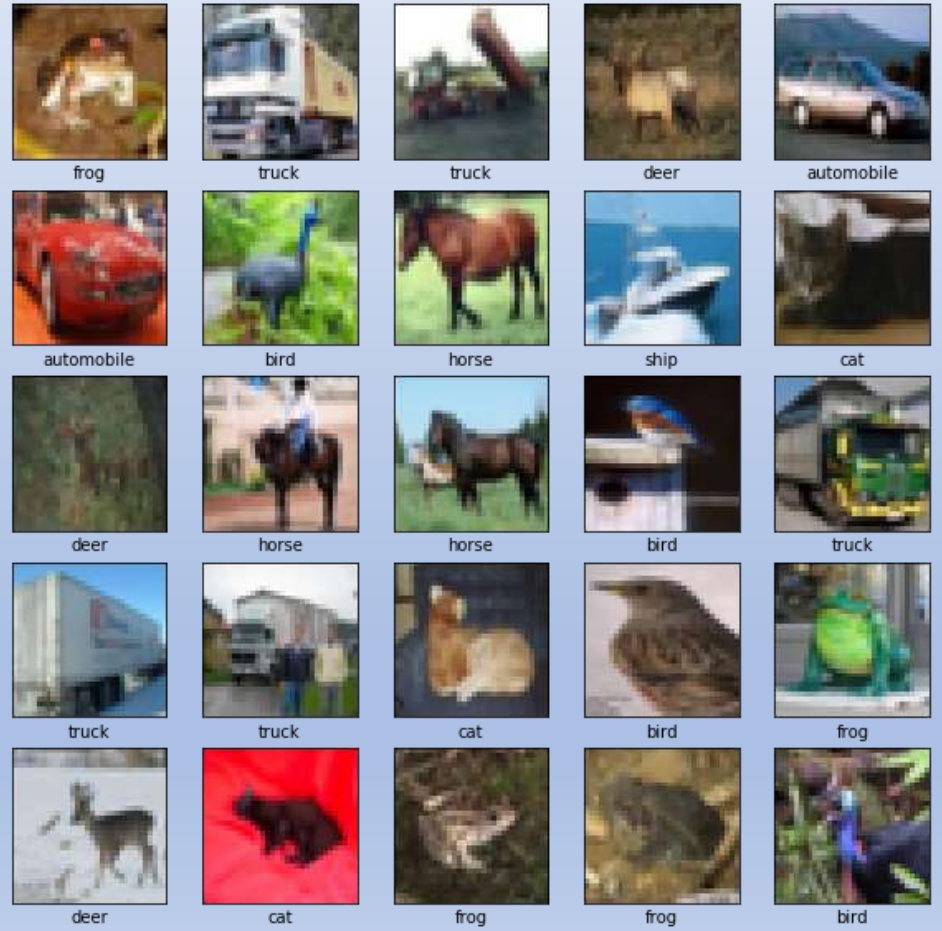
Quelques autres références

- Livres: Deep Learning, Introduction to deep learning, **Neural Networks and Deep Learning: A Textbook**
- Internet: Medium, Kaggle, Github, ...
- Coursera: Andrew Ng

Types de deep learning (I): apprentissage supervisé

classification, régression): ex CIFAR 100: classification par catégories en partant d'une base de données avec étiquettes.

Output = diagnostique



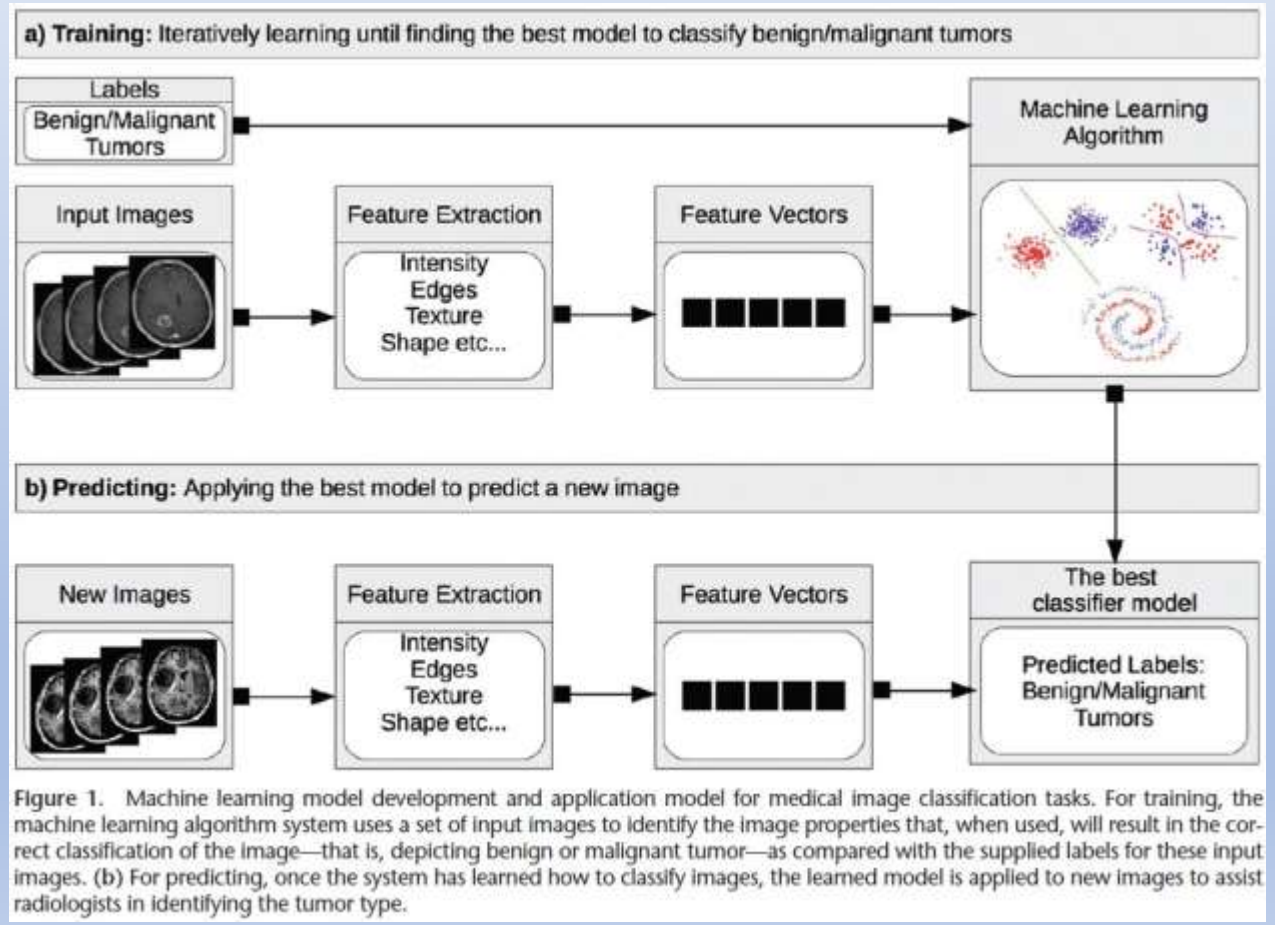
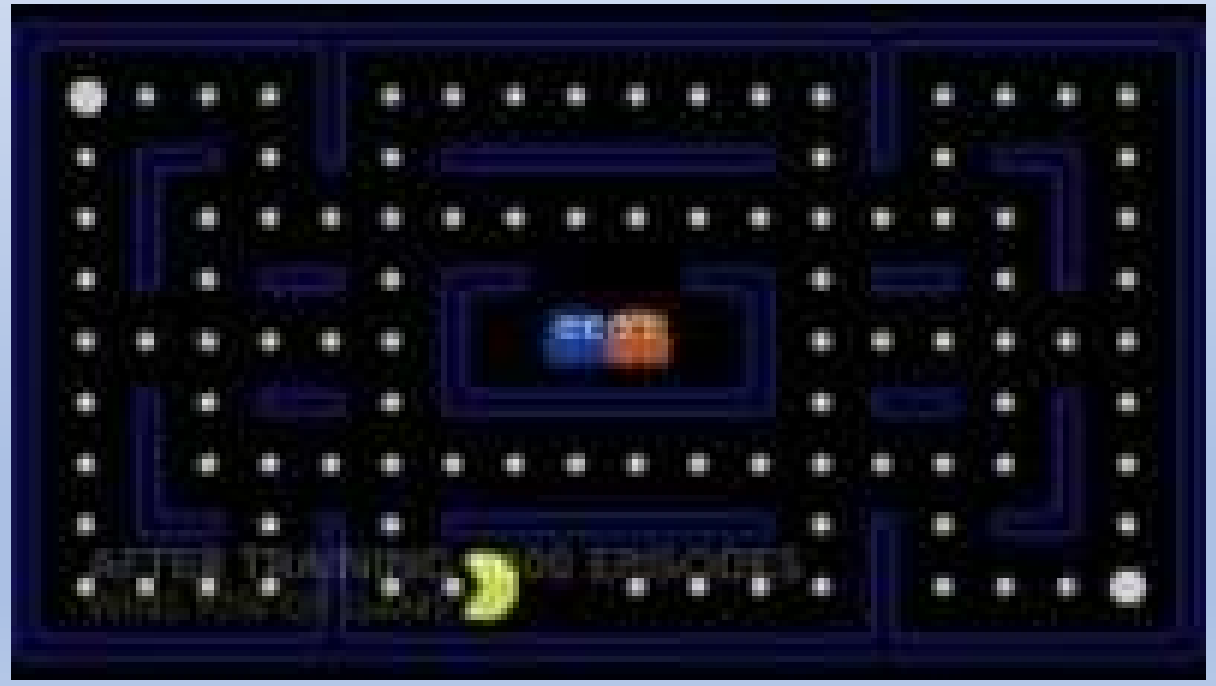


Figure 1. Machine learning model development and application model for medical image classification tasks. For training, the machine learning algorithm system uses a set of input images to identify the image properties that, when used, will result in the correct classification of the image—that is, depicting benign or malignant tumor—as compared with the supplied labels for these input images. (b) For predicting, once the system has learned how to classify images, the learned model is applied to new images to assist radiologists in identifying the tumor type.

Types de deep learning (II): apprentissage par renforcement (reinforcement learning)

apprendre à agir de manière répétée en choisissant dans un ensemble d'actions et en recevant des recompenses: ex. jeu. Pas d'étiquettes, un critère général

Output = stratégie



https://www.youtube.com/watch?v=QilHGSYbjDQ

https://www.youtube.com/watch?v=VMp6pq6_QjI

Paris parking ☺ : 2:42

Types de deep learning (III): apprentisage non-supervisé, génératif (unsupervised learning, generative AI)

But = génération d'objets similaires à un dataset, sans étiquettes,

Output = « création d'objets»

Image générée: résultats très bons, petites imperfections

Source: wikipedia 2021

Générer des images à partir d'une description:
https://www.craiyon.com/

« State of the art » images : Midjourney
https://www.midjourney.com/showcase/recent/

https://archive.org/search?query=creator%3A%22Midjourney%22&sort=-date

https://www.boundless-creativity.com/midjourney-what-artist-style-is-your-favorite/

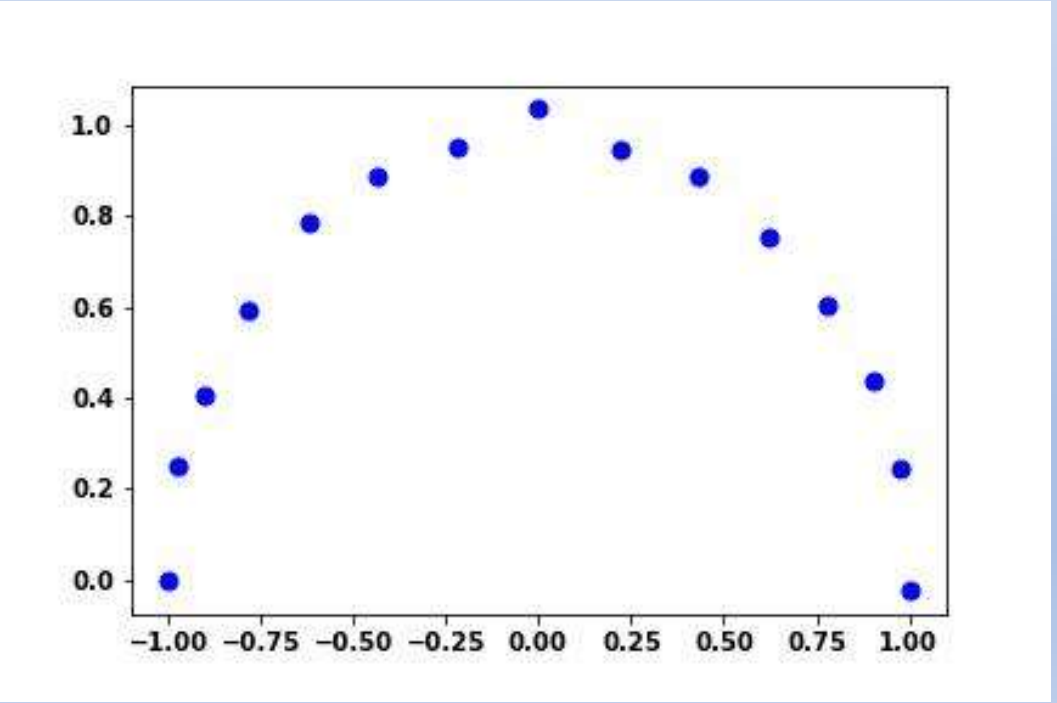
Vidéo : https://www.wombo.ai/
Autres applications : voix à partir de l'image etc.

Principes de fonctionnement des réseaux neuronaux et IA: régularisation, réduction de dimension

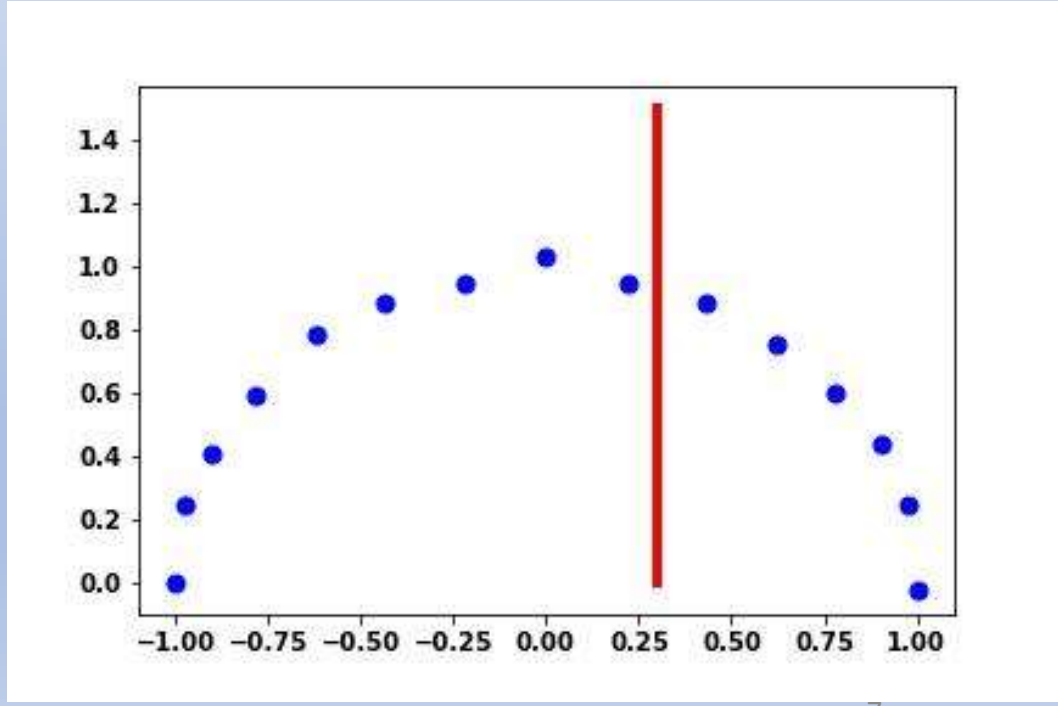
Comment une architecture de type NN peut prédire des choses ?

Exemple du chatGPT : une sorte de complétion automatique (smartphones …)

base d'entrainement



point à « deviner » x=0.3, y=???



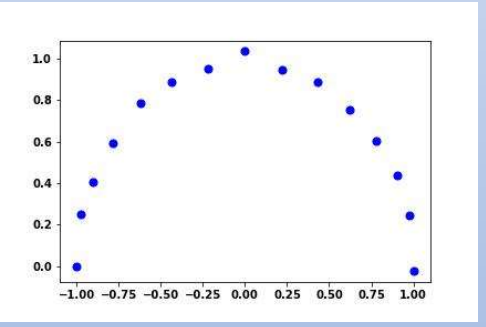
Principes de fonctionnement des réseaux neuronaux et IA: régularisation, réduction de dimension

Comment une architecture de type NN peut prédire des choses ?

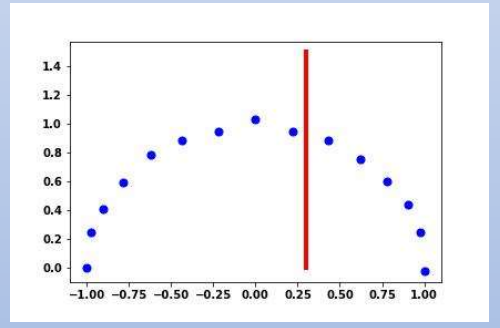
Exemple du chatGPT : une sorte de complétion automatique (smartphones …)

Exemple 1 :

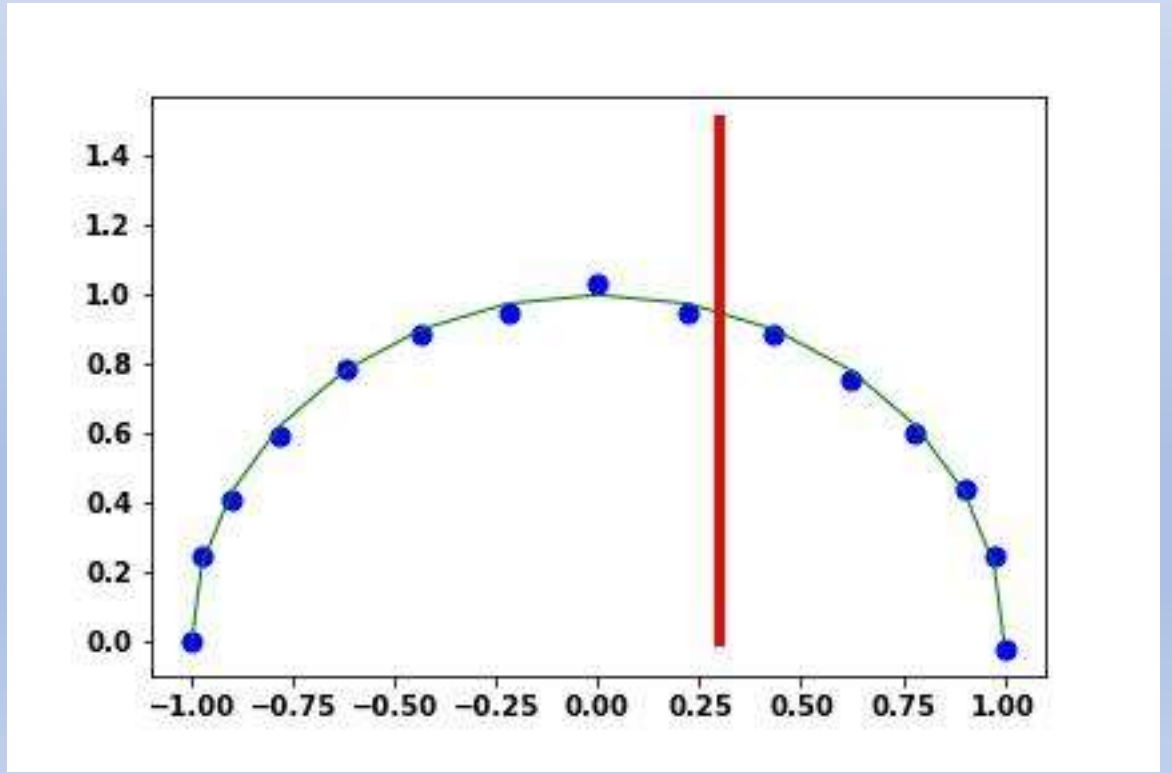
base d'entrainement



point à « deviner » x=0.3, y=???



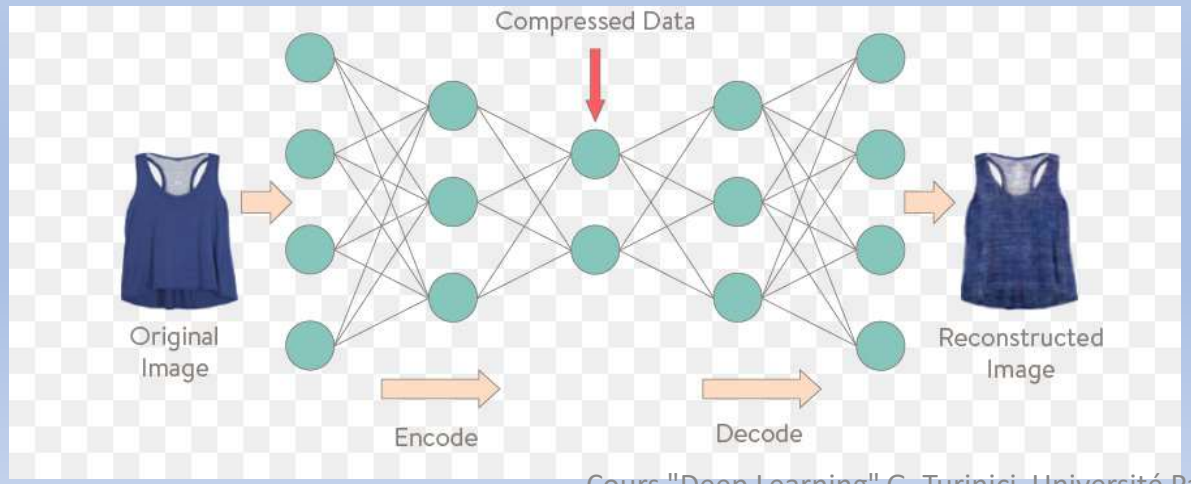
intuition



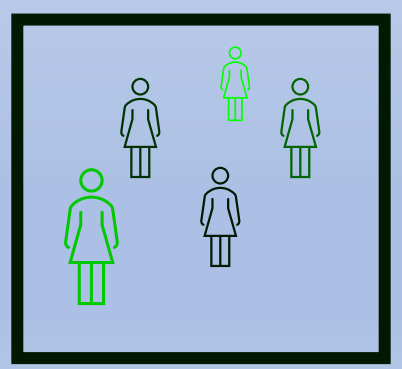
Principes de fonctionnement de l'IA génératif: réduction de dimension par architecture de type réseau neuronal

Une architecture typique (VAE) : encodeur-décodeur: réduction de dimensionnalité : tout est encodé (c'est-à-dire réduit) à un petit ensemble de nombres réels (2-3... 200 – 300 p. ex. mots dans le vocabulaire);

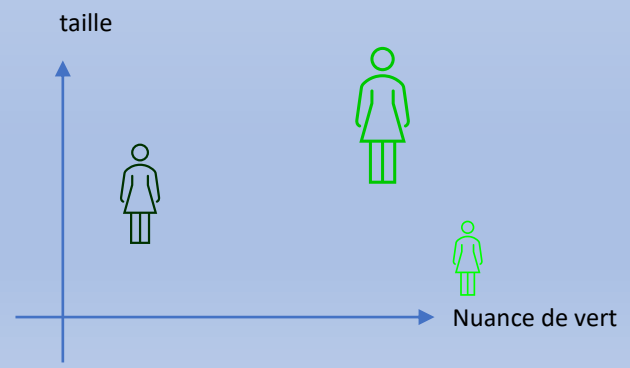
ok pour les images mais pas pour tous les aspects de la vie.



Dataset d'entrainement



Vision espace latent



Ne génèrera jamais autre couleur, autre personne ou un objet !

Principes de fonctionnement des réseaux neuronaux et IA: sampling conditionnel par régularisation

Comment une architecture de type NN peut prédire des choses ?

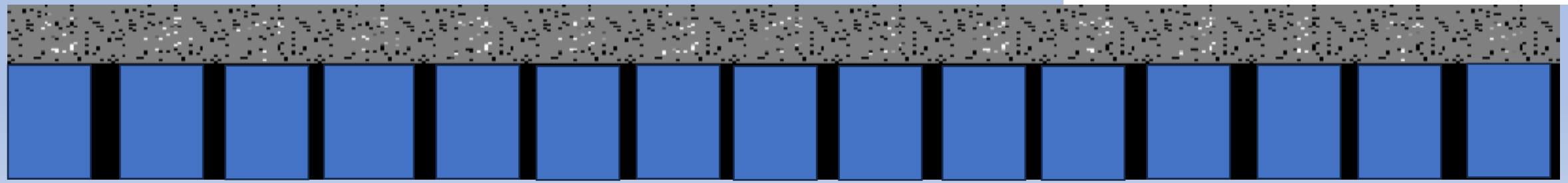
Exemple 2 (conférence G.T. sept 2023

DOI: 10.1007/978-3-031-53036-4_24)

Base d'entrainement MNIST (chiffres manuscrites)

15 chiffres à deviner ; points gris = info effacée à deviner

FAITES-LE !



Principes de fonctionnement des réseaux neuronaux et IA: sampling conditionnel par régularisation

Comment une architecture de type NN peut prédire des choses ?

Exemple II (conférence G.T. sept 2023)

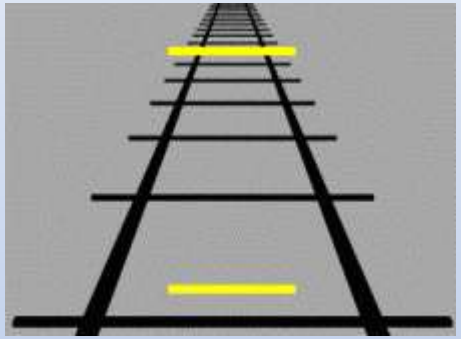
Base d'entrainement MNIST (chiffres manuscrites)

15 chiffres à deviner ; points gris = info effacée à deviner

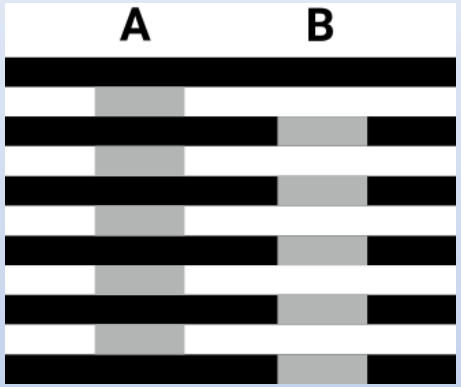
FAITES-LE !

Vision humaine : des « bogues » ?

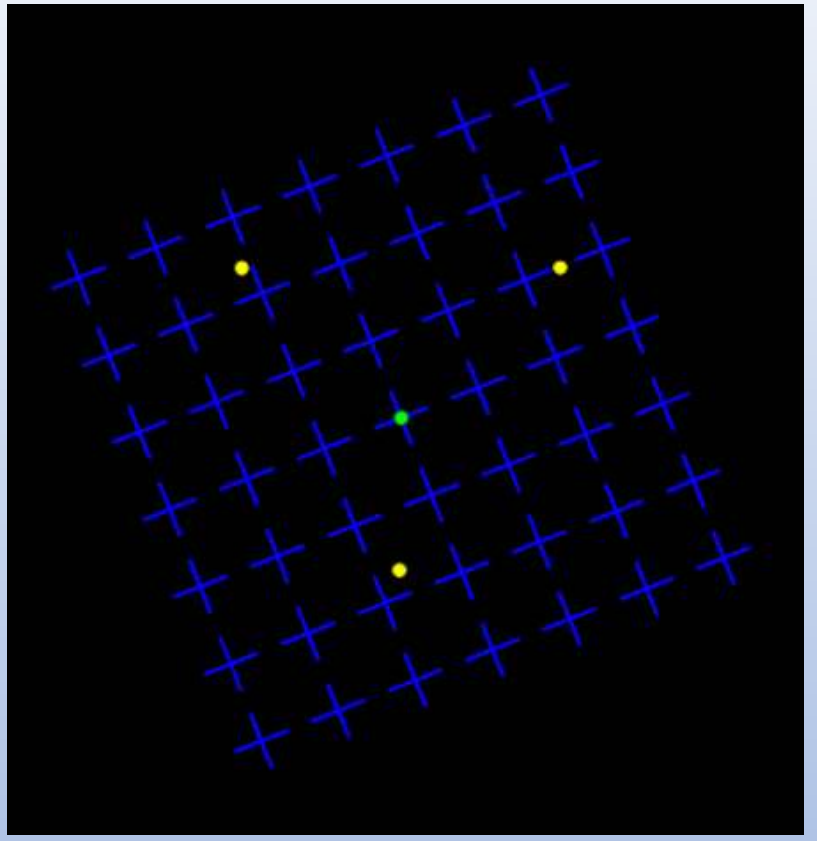
La vision humaine simplifie aussi la réalité …



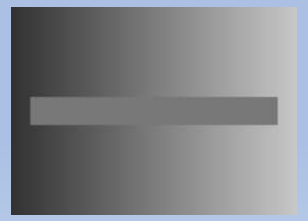
https://en.wikipedia.org/wiki/Ponzo_illusion sept 2023



https://en.wikipedia.org/wiki/White%27s_illusion sept 2023



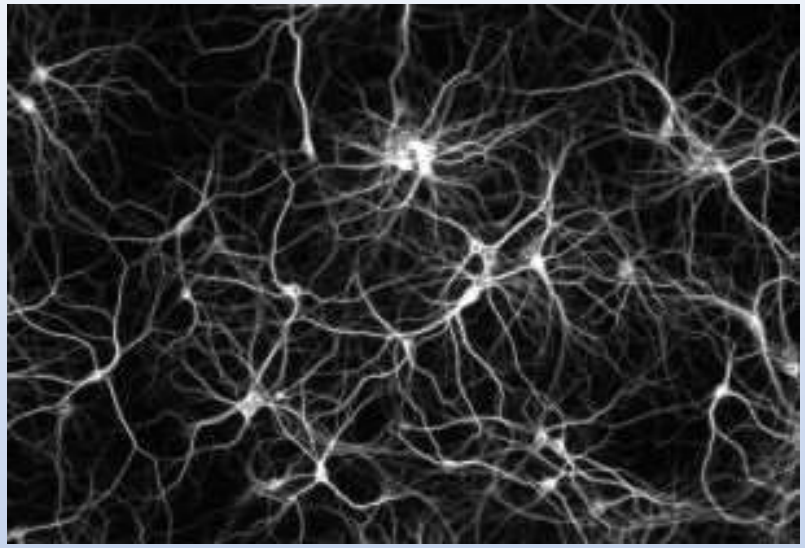
Ref. : Antoine Danchin



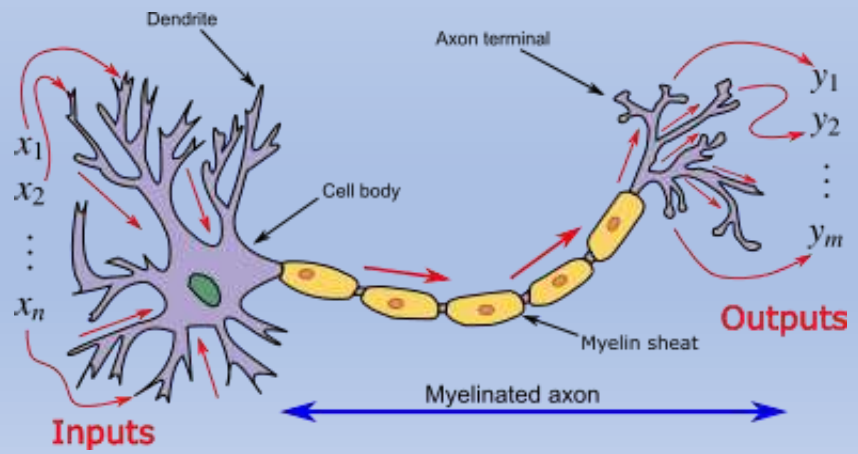
ATTENTION: IL FAUT UTILISER LA VERSION VIDEO PAS IMAGE

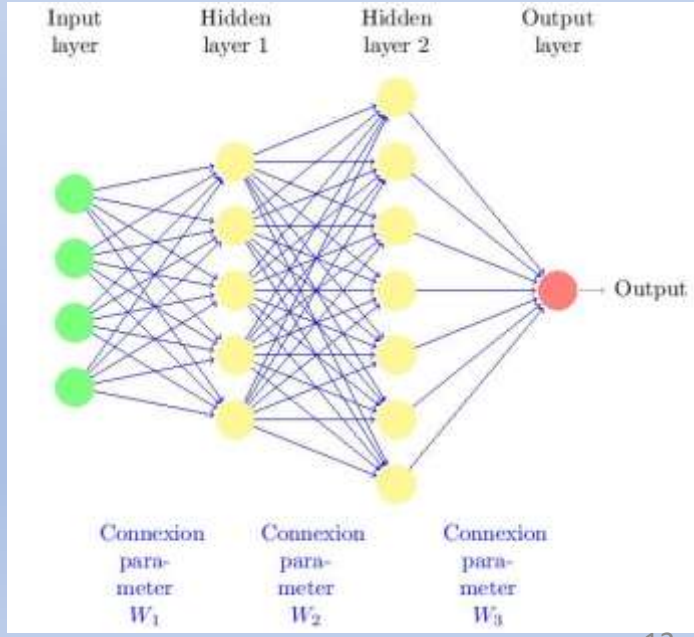
L'observateur se concentre sur le point vert vacillant au milieu. Après environ 10 secondes, l'observateur voit un, deux ou les trois points jaunes statiques disposés aux coins d'un triangle équilatéral imaginaire disparaître puis réapparaître. Ces disparitions et réapparitions se poursuivent de manière pseudo-aléatoire aussi longtemps que l'observateur se soucie de regarder.
https://en.wikipedia.org/wiki/Motion-induced_blindness, sept 2023
Si vidéo ne marche pas sur wikipedia utiliser :
https://turinici.com/wp-content/uploads/research/MotionBlindnessf.gif

Introduction: deep learning



- importance de ne pas trop utiliser de connaissances du domaine mais plus générales
- Partie technique : organisé en réseau de neurones artificiels (ANN ‘artificial neural networks’)
- Neurone:





Opération des neurones: transformer l'input

Deux étapes:

- combinaison linéaire des inputs: $x_k^{l+1} = \sum_{j=1}^{J} W_{jk}^l x_j^l + b_k^l$ donc, en forme vectorielle $x^{l+1} = W^l x^l + b^l$ (b^l = biais)

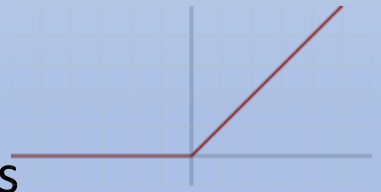
PARAMETRES DU RESEAU = LES POIDS DE LA COMBINAISON LINEAIRE (CONNEXIONS), A OPTIMISER

- fonction de transformation (dite « d'activation »): elle doit être non-linéaire sinon tout l'opération est linéaire

Exemples de fonctions d'activation (cf. https://en.wikipedia.org/wiki/Activation_function)

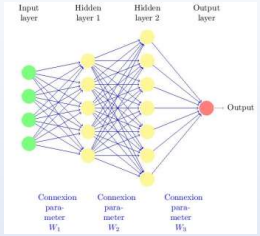
- Logistique $\sigma(x) = \frac{1}{1+e^{-x}}$ pour arriver dans (0,1)

- Rectified linear unit (ReLU) ReLU(x)=x_+ nonlineaire, gradients assez grands



- Softmax(x)= $\frac{e^{x_k}}{\sum_{l=1}^{K} e^{x_l}}$ pour sortir une loi de proba (e.g. classification):

Opération globale: optimisation du réseau, fonction loss



$\mathcal{L}$(paramètres réseau = X) = E_ω[Loss(aléa = ω, paramètres réseau = X)]

aléa = ω = ex. les images à classifier

X = poids du réseau + biais (W,b)

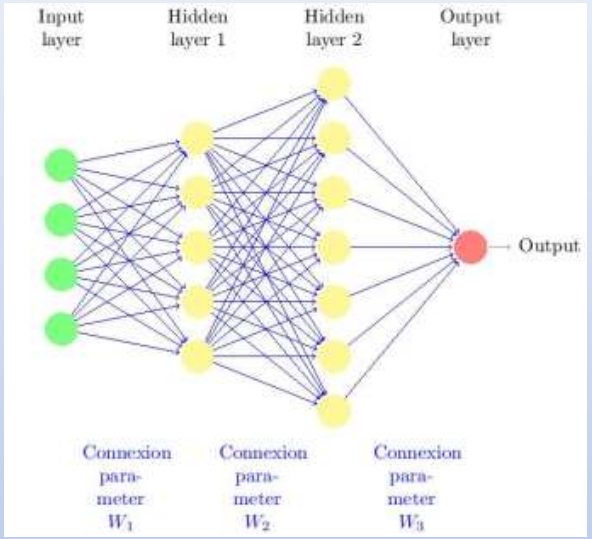
Fonction loss : exemple, classification chats/chiens

Activation = sigmoide: loss = [activation – label(0/1)(ω)]^2

Ou sinon d'autres métriques, e.g., « cross-entropy »

Opération globale: optimisation du réseau, fonction loss

Loss: $\mathcal{L}(\text{paramètres réseau} = X) = E_\omega[L(\omega, X)]$



En pratique: on a besoin des gradients $\nabla_x \mathcal{L}$ de la loss par rapport à X.

Difficile à obtenir, on aura juste une moyenne sur quelques ω_k, k=1,…K (« batch size ») en utilisant des PARALLELISATION PAR BATCH

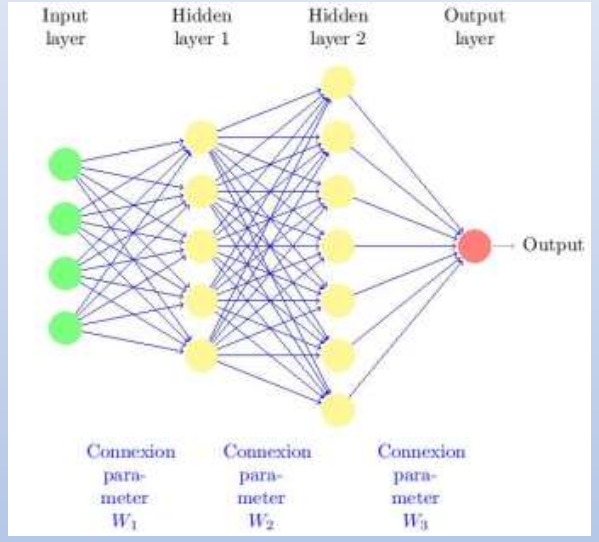
Optimisation: avec Stochastic Gradient Descent (SGD) ou similaires (Adam, momentum, Nesterov, SGD-G2, …) $X_{n+1} = X_n - \rho_n \nabla_x L(\omega_n, X_n)$

Paramètres, hyperparamètres, architecture

Loss: $\mathcal{L}(\mathrm{X}) = E_{\omega}[L(\omega, \mathrm{X})]$

Optimisation: SGD: $X_{n+1} = X_n - \rho_n \nabla_x L(\omega_n, X_n)$

Paramètres	Hyperparamètres	Architecture
X (W,b)	ρ_n	les dimensions du réseau, les type de couche, la régularisation, etc…

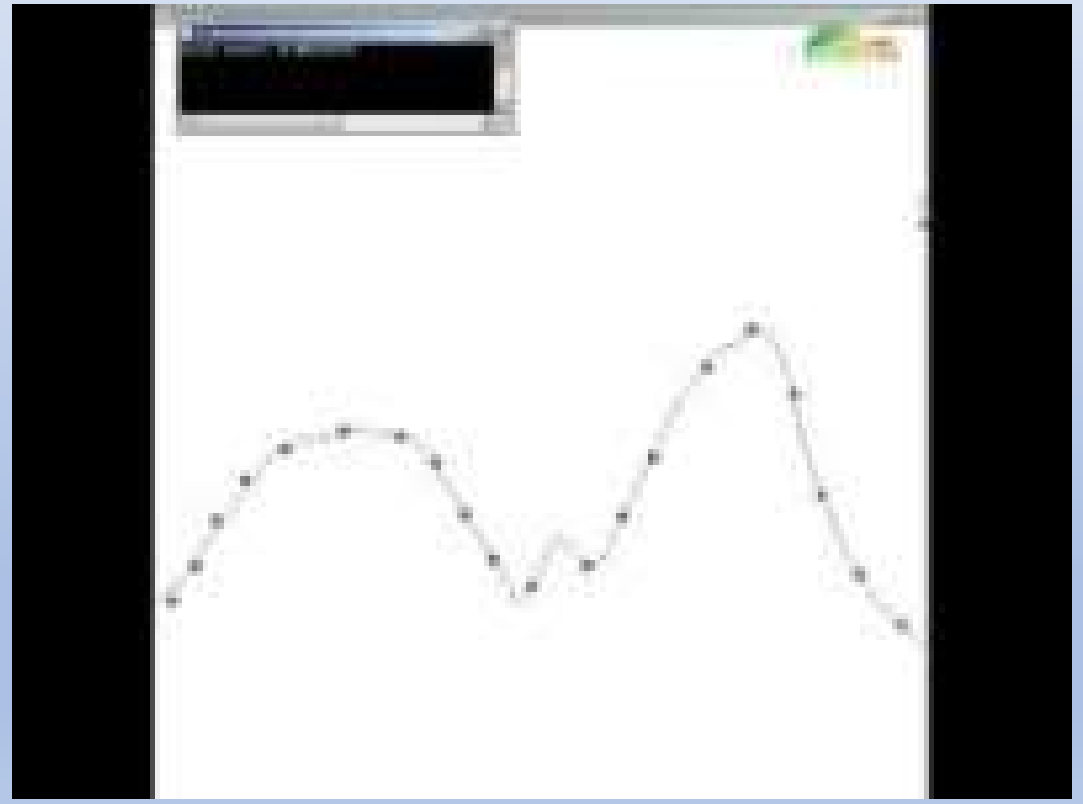


Questions théoriques en NN

Question (expressivité): est il possible d'approcher toute fonction arbitraire ?

Réponse: oui, à condition d'avoir de la non-linéarité (e.g. fonctions d'activation)

Exemple: approximation de fonction arbitraire 1D



Ref : https://www.youtube.com/watch?v=cCevQsEp9Hw

Algorithmes d’optimisation stochastiques et leur convergence

Questions théoriques en NN

SGD: $X_{n+1} = X_n - \rho_n \nabla_x L(\omega_n, X_n)$

PROBLEME: on ne dispose pas du VRAI gradient

$$\nabla_x \mathcal{L}(\mathrm{X}) = \nabla_x E_\omega L(\omega, \mathrm{X})] = E_\omega [\nabla_x L(\omega, \mathrm{X})]$$

mais de sa version bruitée $\nabla_x L(\omega, \mathrm{X})$ = $\nabla_x \mathcal{L}(\mathrm{X})$ + bruit !!

Question: SGD converge avec si peu d'information ?

Performance: overfitting, généralisation, robustesse: weight decay, dropout, …

Autres considérations fonction loss: « vanishing gradients »: si trop de couches avec activations exponentielles (e.g. sigmoïde) : gradients trop petits/ grands, pas d'apprentissage.

Convergence du SGD

Origine de la variabilité: vient des objets de la training set e.g. on peut calibrer un réseau pour 100% qualité du label sur une image donnée, mais les paramètres dépendent de l'image, changeront pour une autre image, il faut faire une sorte de moyenne

Exemple simple: on veut retrouver la moyenne d'une normale qu'on peut échantillonner: $L(\omega = Y, X) \sim \frac{(X-Y)^2}{2}$ avec $Y \sim N(m, \sigma^2)$

Exo 1

a/ écrire SGD: $X_{n+1} = X_n - \rho_n \nabla_x L(Y_n, X_n)$ pour ce cas,

b/ Pour $\rho_n = \rho$(constante), décrire X_n comme variable aléatoire: calculer moyenne, variance, loi

c/ calculer $\lim_{n \to \infty} X_n$

Convergence du SGD

Exo 1 –corrigé

SGD: $X_{n+1} = X_n - \rho_n(X_n - Y_n) = (1-\rho_n)X_n + \rho_n Y_n$ avec Y_n indépendant des autres ($\perp$)

Il s'agit d'une EMA (exponential moving average)

b/ $X_{n+1} = \rho Y_n + (1-\rho)X_n = \rho Y_n + (1-\rho)\rho Y_{n-1} + (1-\rho)^2 X_{n-2} = \ \ldots$

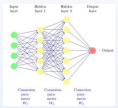
Moyenne: $m_{n+1} = (1-\rho)m_n + \rho m$, variance $\sigma_{n+1}^2 = (1-\rho)^2\sigma_n^2 + \rho^2\sigma^2$

c/ $\lim_{n\to\infty} X_n$: moyenne m (cv. exp si $0 < \rho < 2$), variance limite $\frac{\rho\sigma^2}{2-\rho}$ (pas nulle !), loi: normale

« Réparation » de la variance finale non-nulle: $\rho_n \to 0$: DECAY SCHEDULE

Exo 2: Question: à quelle formule pour ρ_n correspond la moyenne empirique usuelle ?? $X_n = \frac{\sum_{k=0}^{n-1} Y_k}{n}$?

Convergence du SGD



Exo 2 –corrigé

Exo 2: Question: à quelle formule pour ρ_n correspond la moyenne empirique usuelle ? $X_n = \frac{\sum_{k=0}^{n-1} Y_k}{n}$?

On aura

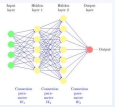
$X_{n+1} = \frac{n \cdot X_n + Y_n}{n+1}$ avec $Y_n \perp (\mathrm{X_n}, \mathrm{X_{n-1}}, \ldots, \mathrm{X_0})$

En comparant avec $X_{n+1} = \rho_n Y_n + (1 - \rho_n) X_n$ on obtient $\rho_n = \frac{1}{n+1}$

$\lim_{n \to \infty} X_n$: moyenne m (cv. exp), variance limite 0 (nulle !), loi: normale

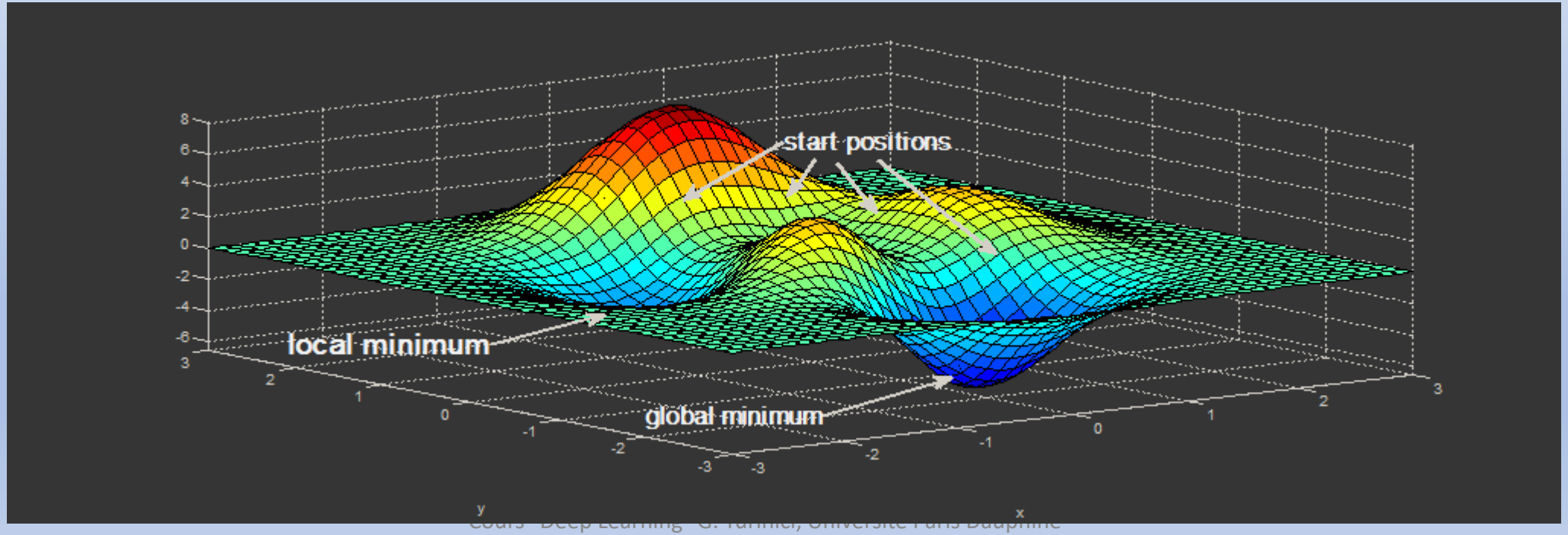
En pratique on utilise parfois des « decay schedules » adaptatifs (cf. SGD-G2 ☺) car on passe successivement par des étapes d'exploration et exploitation

Convergence du SGD

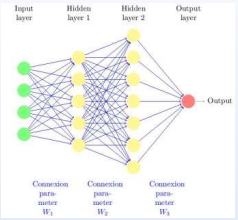


En pratique on utilise parfois des decay schedules adaptatifs (cf. SGD-G2 ☺) car on passe successivement par des étapes d'exploration et exploitation

Ref: https://i.stack.imgur.com/EGx2a.png



Convergence du SGD



SGD: $X_{n+1} = X_n - \rho_n \nabla_x L(\omega_n, X_n)$

- Convexité : $\mathcal{L}(y) \geq \mathcal{L}(x) + \langle \nabla \mathcal{L}(x), y - x \rangle$
- Convexité forte : $\mathcal{L}(y) \geq \mathcal{L}(x) + \langle \nabla \mathcal{L}(x), y - x \rangle + \frac{\mu}{2} \|x - y\|^2$
- Lipschitz: $\mathcal{L}(y) - \mathcal{L}(x) \leq C_{Lip} \|x - y\|$
- Gradient borné $\mathrm{E}_\omega[\|\nabla_X \mathrm{L}(\omega, X)\|^2] \leq C_0 + C_1 \|X\|^2$

Théorème Supposons que $\mathcal{L}$ est une fonction strictement convexe, Lipschitz, à gradient borné (quadratiquement, cf. lien arxiv). Alors:

a) le minimum de $\mathcal{L}$ est unique, il sera noté X_*

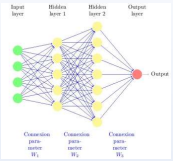
b) $\mathrm{E}[\|X_{n+1} - \mathrm{X}_*\|^2] \leq (1 - \rho_n \mu + \rho_n^2 c_1) \mathrm{E}[\|X_n - \mathrm{X}_*\|^2] + \rho_n^2 \, c_0$

c) En particulier, si $\rho_n \to 0$ et $\sum_{n=1}^{\infty} \rho_n = \infty$ (ex: $\rho_n = \frac{c}{n}$) alors SGD converge.

- Preuve: voir le document en ligne (anglais) DOI: http://arxiv.org/abs/2103.14350

Calcul du gradient et implémentation en « pure python »

Calcul du gradient: backpropagation



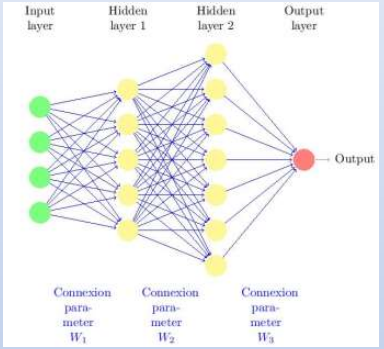
Back-propagation et nécessité de la diff. automatique: pour le SGD on a besoin du gradient

- Les différences finies ne sont pas efficaces, il faut utiliser la propagation rétrograde, technique bien plus ancienne que le ‘machine learning’, venant de la théorie du contrôle optimal.

Document à consulter: cours de M1 (cf. turinici.com, ensuite cours « deep learning » etc) ou livre sur Amazon https://www.amazon.fr/dp/B09QP6QCNQ

Calcul du gradient: exemple de backpropagation

Document à consulter: cours de M1 (cf. turinici.com, ensuite cours « deep learning » etc) ou livre sur Amazon https://www.amazon.fr/dp/B09QP6QCNQ



$Y_0 \to \widetilde{Y_1} = W_1 Y_0 + b_1 \to Y_1 = A_1(\widetilde{Y_1})(RELU)$, …, softmax, loss

Backpropagation: $\delta L = Id, \delta Y_3 = \frac{\partial L}{\partial Y_3}$, $\partial \widetilde{Y_3} = \frac{\partial L}{\partial \widetilde{Y_3}} = \frac{\partial L}{\partial Y_3}\frac{\partial Y_3}{\partial \widetilde{Y_3}}$= …

But: obtenir les gradients $\delta W_k = \frac{\partial L}{\partial W_k}$ et $\delta b_k = \frac{\partial L}{\partial b_k}$ pour k=1,2,3.

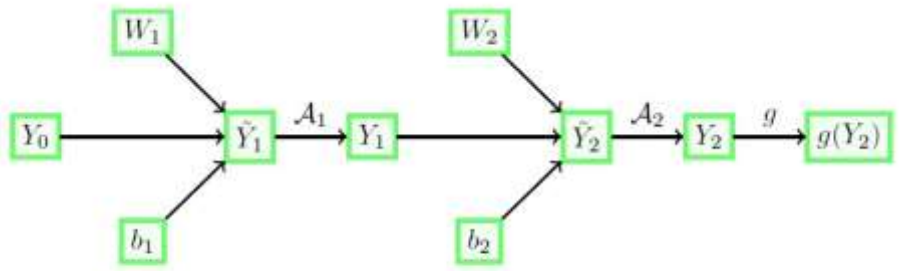


FIGURE 3.5 – Graphe de propagation forward du réseau de neurones

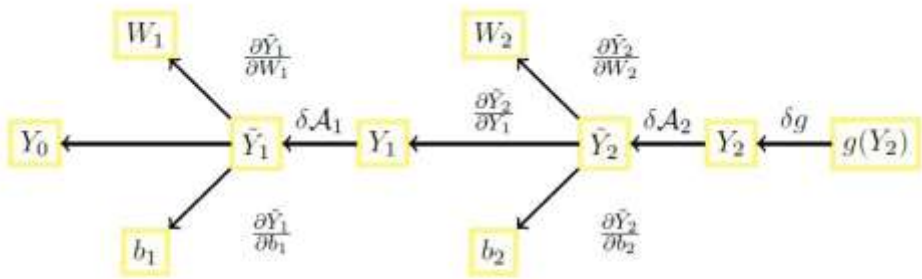
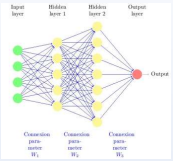


FIGURE 3.6 – Graphe de propagation backward du réseau de neurones

Backprop: formules (démontrées en classe)



Rappel notation $\partial(Tom) = dérivée\ du\ loss\ final\ par\ rapport$ à "Tom"

Gradient de softmax+cross-entropy

opération: Y -> S= np.exp(Y)/np.sum(np.exp(Y)) (softmax)

S -> Loss(Y)=- np.sum(P*np.log(S)) (P= vraies étiquettes)

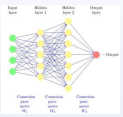
$Exo: démontrer\ la\ formule :\ \partial Y = S - P\ (noté\ \tilde{Y}\ dans\ le\ code)$

Gradient de ReLu opération $\widetilde{Y_k} \mapsto Y_k = ReLu(\widetilde{Y_k}), formule: \delta\tilde{Y}_k = \delta Y_k \cdot 1_{\tilde{Y}_k \geq 0}$

Gradient de la partie linéaire $opération$: $\widetilde{Y}_{k+1} = W_{k+1}Y_k + b_{k+1}$

$formule: \delta Y_k = W_{k+1}^T \cdot \delta\tilde{Y}_{k+1}, \delta W_{k+1} = \delta\tilde{Y}_{k+1} \cdot Y_k^T, \delta b_{k+1} = \delta\tilde{Y}_{k+1}$

Implémentation python (sans tensorflow, pytorch etc)



Etapes: 0/ chargement BD (iris), 1/création réseau, 2/ itérations: sampling, « fwd. pass », « backprop » (calcul du gradient), « SGD step »

0/ chargement iris

```
from sklearn import datasets
#Load the iris dataset
iris = datasets.load_iris()
X,Y = iris.data,iris.target
# tranforms y in vector of len=3 with 0/1 : hot encoding
nb_classes = len(np.unique(y))
one_hot_targets = np.eye(nb_classes)[y]
```

1/intialisation couche dense x -> W*x+b

#function to create a dense layer

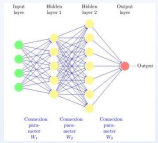
```
def create_dense_layer(n_input,n_output):
  weights=0.1*np.random.rand(n_output,n_input)
  biases=np.zeros((n_output,1))
  return weights, biases

def ReLU(input_array):
  return np.maximum(0,input_array)

dim1,dim2,dim3,dim4=4, 32, 16, 3
weights1, biases1 = create_dense_layer(dim1, dim2)
weights2, biases2 = create_dense_layer(dim2,dim3)
weights3, biases3 = create_dense_layer(dim3,dim4)
```

2/ Itérations …

Culture deep learning: présentations 2-3 min



- Sujet 1: donner des exemples d'acteurs institutionnels avec leur positionnement (applications, techniques) et contributions (année), et le cas échéant le type d'architecture de réseaux utilisé.
- Sujet 2: lister principales plateformes de développement logiciel et exemples de benchmarks (données), architectures réseaux
- Sujet 3: lister principaux algorithmes d'optimisation au delà du SGD, préciser si c'est dans un cadre supervisé ou pas

Acteurs: Meta, IAlabs, (ex Facebook, cf. Yann LeCun) ; 10cent, deepN (traduction), Netflix (recommandations)

Problèmes: interactions virtuel/réel, joueur vidéo, traduction automatique (NLP par phrase)

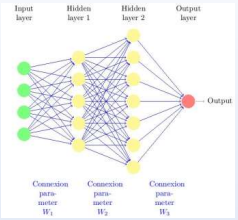
Benchmarks: images (MNIST, MS-coco -detection), NLP (base Wikipedia,…), 10cent database; Maryana;

Plateformes: pytorch (orienté Python), Tensorflow (opensource par Google; visualisation TensorBoard, version high-level python: Keras)

Architectures: LSTM (jeux vidéo), CNN, RNN (e.g. traduction, langage, ..), GAN, VAE,

Algorithmes d'optimisation: SGD, Adam (adaptation taux apprentissage), algo évolutionnaires (« générations » de solutions; e.g. GA), ADMM (directions alternées, cas particulier de l'algo proximal), Dijkstra (ds un graphe),

Culture deep learning: présentations 2 min



Version 2020/21

Exposé: donner des exemples d'acteurs 3 et chercheurs 2 avec leur positionnement (quel domaine, quelle technique) et contribution (année) et exemples de benchmarks (données) 3.

Acteurs Google (Tensorflow, Keras), Facebook, OpenAI (GPT3), Netflix (suggestions), NVIDIA, Airbus (détection nuages image satellites), DeepMind-Google, Tesla, DeepL, IBM, Baidu, Alibaba, Tencent, Xiaomi

Chercheurs: Yann le Cun, Youshua Bengio, Ian Goodfellow (GAN), Geofrey Hinton (prix TURING), Karpathy (Tesla, Stanford), V. Vapnik, J. Schmidhuber, …

Problèmes: alphaFold (prédiction repliement protéines)

Benchmarks: CIFAR10/100, MNIST, Wikiner (traduction), Sentiment140 (tweets), Imagenet (images) , Ember (virus +), Feret (reconnaissance faciale), SpaceNet (images satellites) , JFLEG (grammaire), NORB (objets 3D)

Rq: beaucoup de concentration, mais le futur et assez imprévisible.

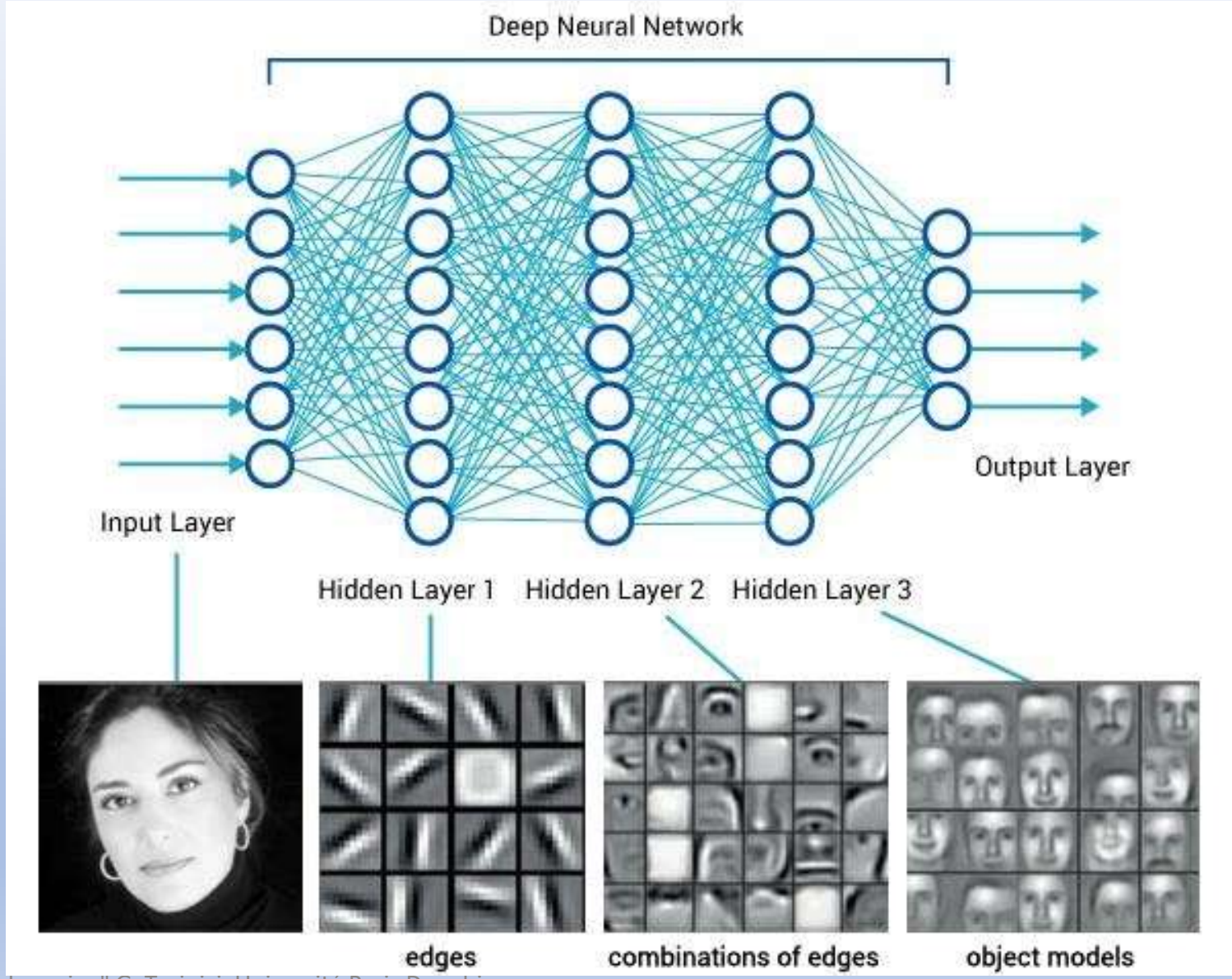
Techniques de construction de réseaux, architectures de réseaux

https://sci2lab.github.io/ml_tutorial/cnn/index.html33

Réseaux convolutifs: CNN

Réseaux convolutifs: selon un modèle naturel (les couches neuronales du cortex visuel).

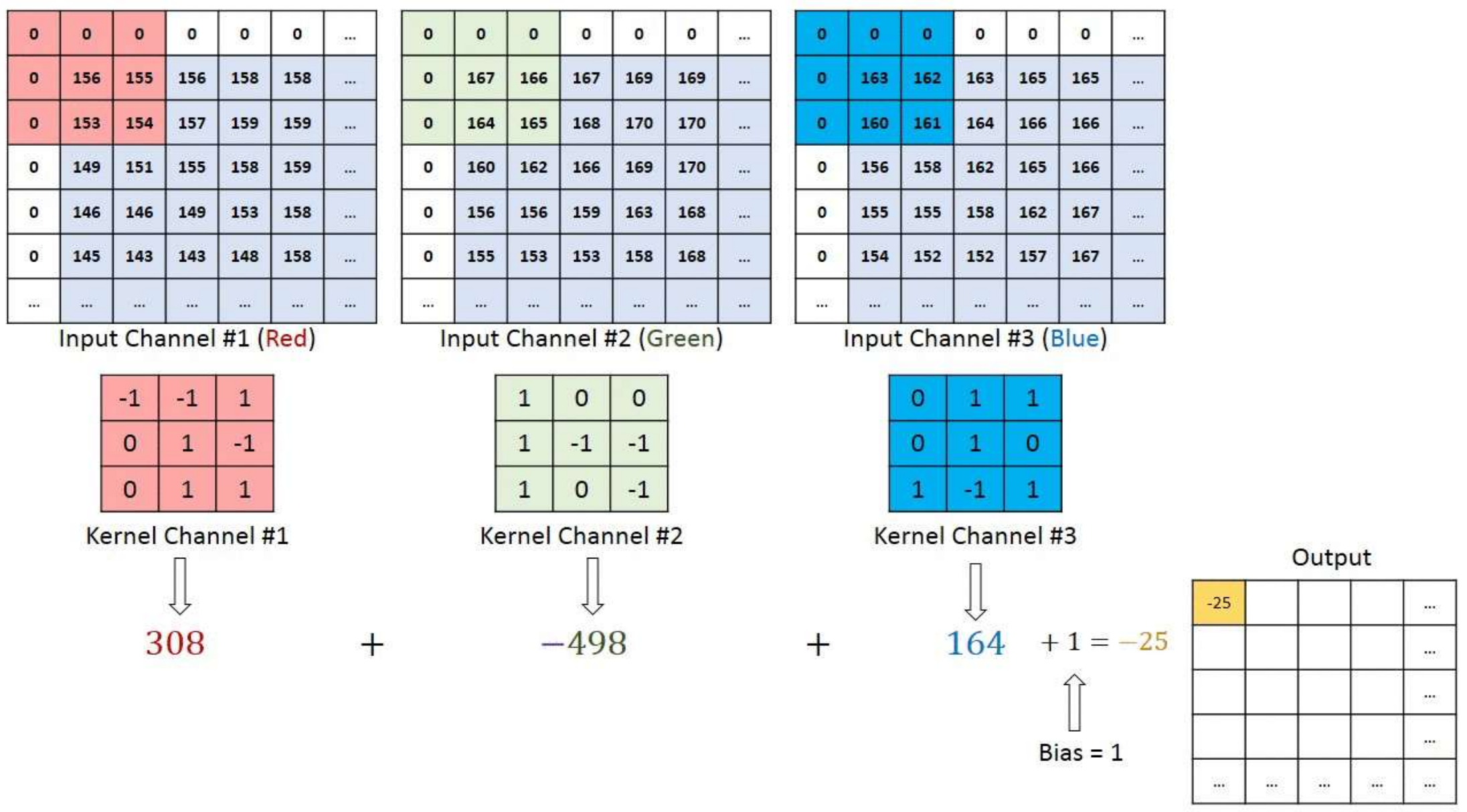
En général c'est un filtre, structure plus simple et efficace (traitement local de l'information donc PARALLELISABLE / TENSORISABLE / CREUX!!!)



https://sci2lab.github.io/ml_tutorial/cnn/index.html34

Réseaux convolutifs: CNN

Remarque théorique: ceci garde la même expressivité (possible d'approcher toute fonction arbitraire) que les couches denses (« fully connected »)

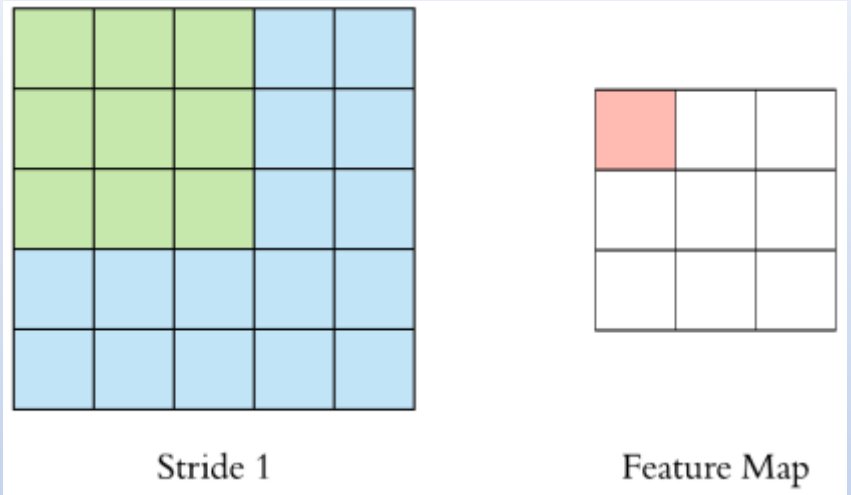


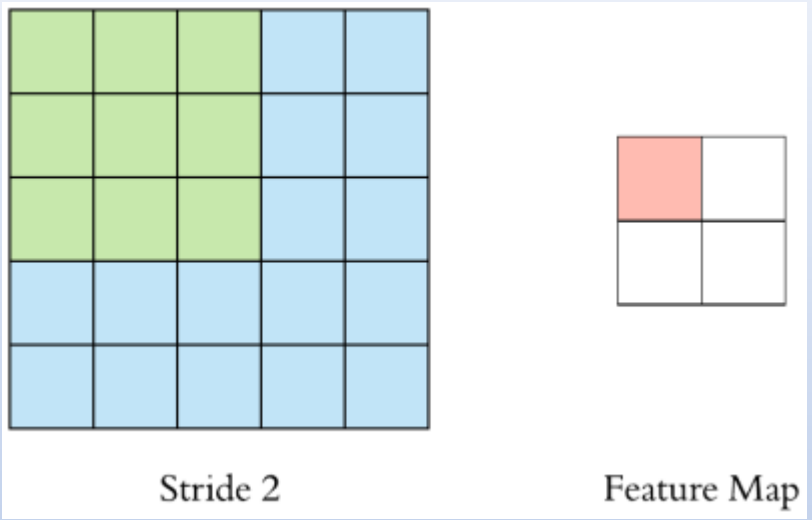
https://medium.datadriveninvestor.com/convolutional-neural-networks-3b241a5da51e

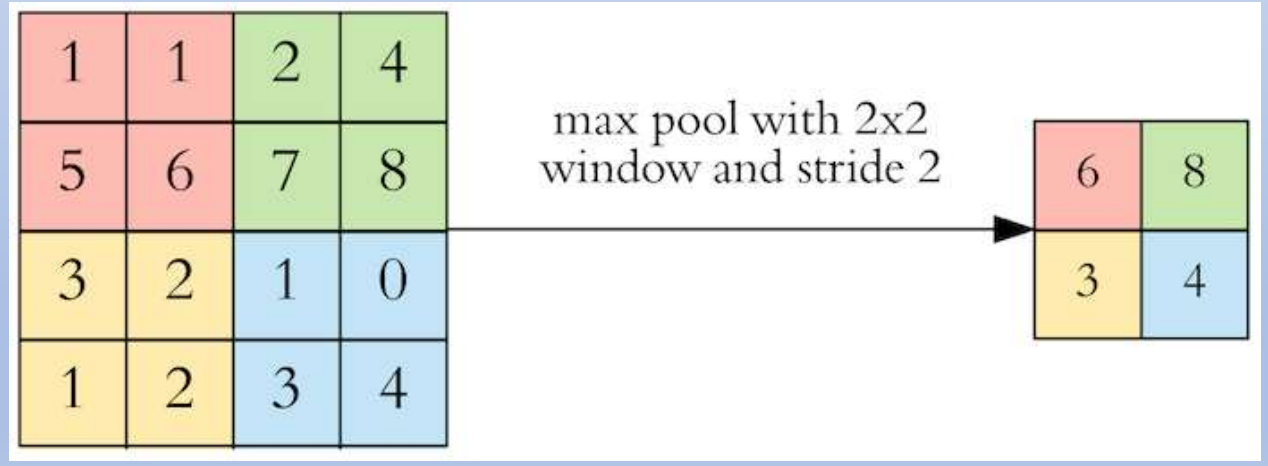
Réseaux convolutifs: CNN

- **Hyperparametres**

1. “Filter size” (kernel size): 3x3 sont souvent vus, sinon 5x5 et 7x7
2. Combien de filtres utiliser, généralement puissance de 2 entre 32 and 1024. Plus c’est mieux (mais + cher en calcul, risque d’overtiffing).
3. Stride: ex: 1, 2, …
4. Padding: oui ou non
5. Max pooling ou pas







Réseaux convolutifs: CNN: exemple de calcul de tailles des tenseurs

AlexNet layers

1.Input: Color images of size 227x227x3. The AlexNet paper mentions the input size of 224×224 but that is a typo in the paper.

2.Conv-1: The first convolutional layer consists of 96 kernels of size 11×11 applied with a stride of 4 and padding of 0.

3.MaxPool-1: The maxpool layer following Conv-1 consists of pooling size of 3×3 and stride 2.

4.Conv-2: The second conv layer consists of 256 kernels of size 5×5 applied with a stride of 1 and padding of 2.

5.MaxPool-2: The maxpool layer following Conv-2 consists of pooling size of 3×3 and a stride of 2.

6.Conv-3: The third conv layer consists of 384 kernels of size 3×3 applied with a stride of 1 and padding of 1.

7.Conv-4: The fourth conv layer has the same structure as the third conv layer. It consists of 384 kernels of size 3×3 applied with a stride of 1 and padding of 1.

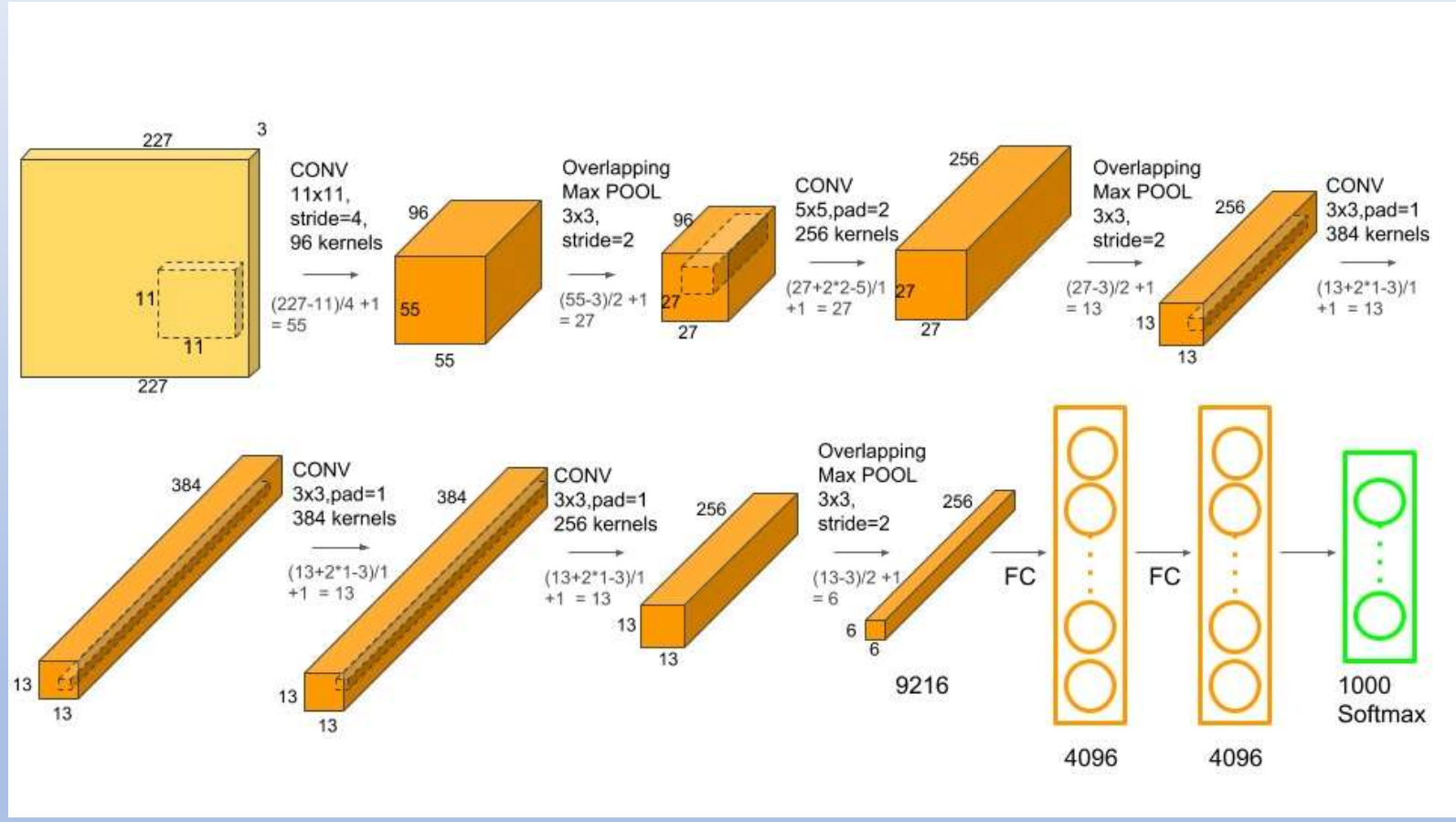
8.Conv-5: The fifth conv layer consists of 256 kernels of size 3×3 applied with a stride of 1 and padding of 1.

9.MaxPool-3: The maxpool layer following Conv-5 consists of pooling size of 3×3 and a stride of 2.

10.FC-1: The first fully connected layer has 4096 neurons.

11.FC-2: The second fully connected layer has 4096 neurons.

12.FC-3: The third fully connected layer has 1000 neurons.



https://learnopencv.com/number-of-parameters-and-tensor-sizes-in-convolutional-neural-network/

Réseaux convolutifs: CNN

CNN conclusion: calcul de filtre localisé qui donne des bonnes performances en pratique (images, audio)

Calcul numérique aisé avec la tensorisation (Tensorflow!)

Exemple pratique (CIFAR10/100): https://www.tensorflow.org/tutorials/images/cnn

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928
=================================================================
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0
```

Exemple de vérification: une filtre 3x3 (x3 channels) utilise 3x3x3 + 1(biais) = 28 variables donc première couche « conv2D » aura 28*32 = 896 variables: la couche de convolution suivante aura 3x3x32 +1 variables par filtre et 64 filtres ce que donne 18496 variables à optimiser … et ainsi de suite.

Réseaux convolutifs: CNN

Exemple pratique (CIFAR10/100): https://www.tensorflow.org/tutorials/images/cnn

A faire: 1/ changer d'algorithme: regarder la doc « Keras / optimizers », e.g. prendre SGD, Adamax, RMSProp, etc.

2/ exécuter plusieurs fois la prédiction et voir où les erreurs sont faites

3/ ajouter une couche convolutionnelle (CNN) avant le dernier Dense

4/remplacer la dernière couche dense par un CNN

Réseaux convolutifs: CNN

Exemple pratique (CIFAR10/100): https://www.tensorflow.org/tutorials/images/cnn

A faire: 1/ changer d'algorithme: regarder la doc « Keras / optimizers », e.g. prendre SGD, Adamax

Mot clé: « adadelta »

2/ exécuter plusieurs fois la prédiction et voir où les erreurs sont faites, avec model.predict(…)

```
j=2405#erreur dans base initiale !! Note: ca peut varier si la base a été melangée
j=11
import numpy as np
model.predict( test_images[j][None,...] )
id_pred=np.argmax( model.predict( test_images[j][None,...] ) , axis=-1)[0]
pred_name=class_names[id_pred]
plt.imshow(test_images[j])
plt.xlabel(class_names[test_labels[j][0]]+ "/"+pred_name)
```

3/ A faire: ajouter plus de filtres (64 à la place de 32) ou des filtres plus gros pour la 1ere couche

Applications simples

Autre exemple: IRIS dataset (keras), classification (cf. page du cours). A faire: utiliser la base scikit-learn « wines » à la place.

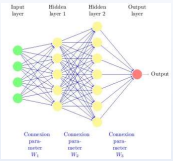
Attention: ici pas la peine de mettre des CNN car pas d'ordre naturel parmi les attributs des données.

Exemple pratique (MNIST) avec couches et optimisation plus en détail:

https://www.tensorflow.org/tutorials/quickstart/advanced

A faire: tester avec une couche qui saute un niveau (cf. Resnet)

Autres éléments d'architecture: régularisation

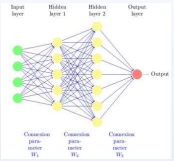


Phénomène d'overfitting / robustesse:

- Dans la fonction « loss » : « loss regularization » L2: « weight decay », L1: « sparsity » : adjoinction d'un terme contenant la norme L1/L2 des inconnues (poids du réseau)

- Dans les couches : dropout (composantes mis à 0 aléatoirement), « batch norm », ...

Exposé



Sujet: (1) plateformes logicielles, (2) applications, (3)architectures réseaux

Exposés:
Groupes 1 & 2:
(1): TensorFlow, Keras, pyTorch, Matlab, R
(2) : data en général par ML/DL (deep fakes, …), voiture autonome (CNN); traducteurs – NLP ex GPT3; musique (Chopin, Beatles, …), arrangement, applications en finances (risque de crédit), reinforcement learning; Waze/ Youtube; question de biais, générations : cinéma, jeux
(3) : ANN, CNN, RNN, au delà du séquentiel

Exemples d'architecture

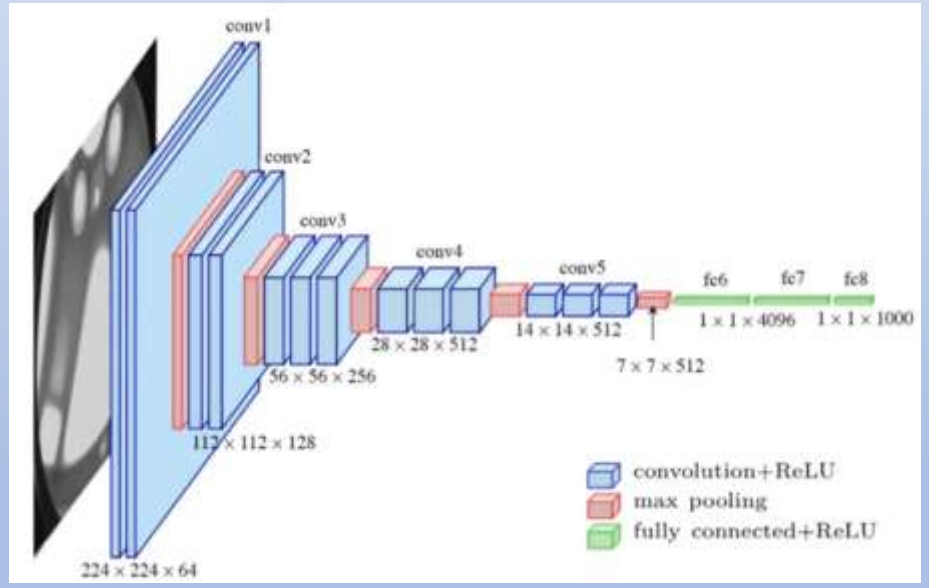
LeNet, AlexNet : Computer Vision (CV), convolutions (CNN),

https://en.wikipedia.org/wiki/AlexNet

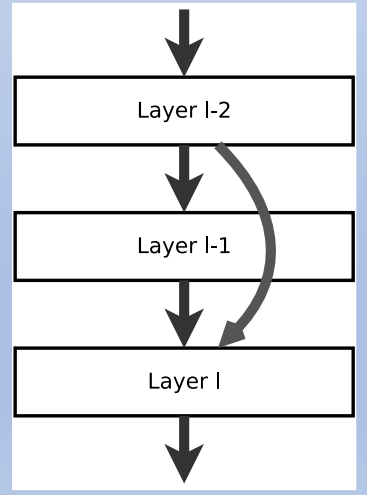
Exemples d'architecture

VGG-16 : CNN, bonne qualité, 16 couches, 138M param (grand!)

VGG 16
(Researchgate.net)



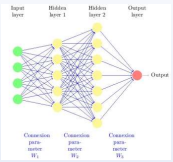
ResNet ('residual network' 2015) : pour mieux calculer le gradient, AVEC des couches qui « sautent », améliore l'apprentissage;



Resnet
(wikipedia)

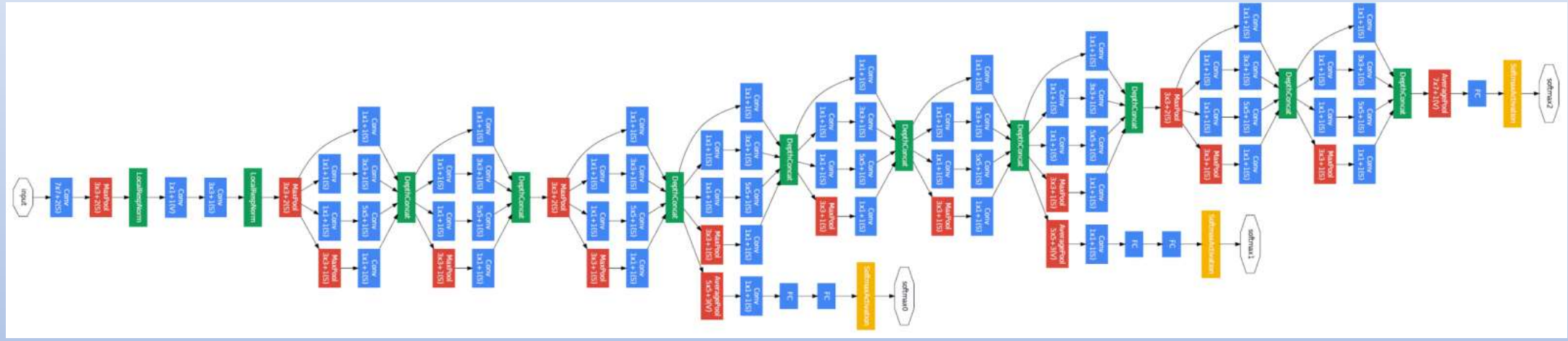
https://blog.floydhub.com/long-short-term-memory-from-zero-to-hero-with-pytorch/

Exemples d'architecture



Inception (GoogleNet) : plusieurs filtres de taille différente au même niveau, astuce k=1x1 (aggregation sur les canaux, analogue du « maxpool » en espace), réduit le cout; architectures similaires disponibles (les trois softmax entrent dans la fonction loss avec poids différents mais seulement le dernier est utilisé en inférence).

Inception



- **Utilisation en transfer learning (en enlevant la couche finale)**
- **TODO: adapter le code ci-dessous pour classifier MNIST utilisant MobileNetV2 (entrainé sur ImageNet)**

https://towardsdatascience.com/10-minutes-to-building-a-binary-image-classifier-by-applying-transfer-learning-to-mobilenet-eab5a8719525

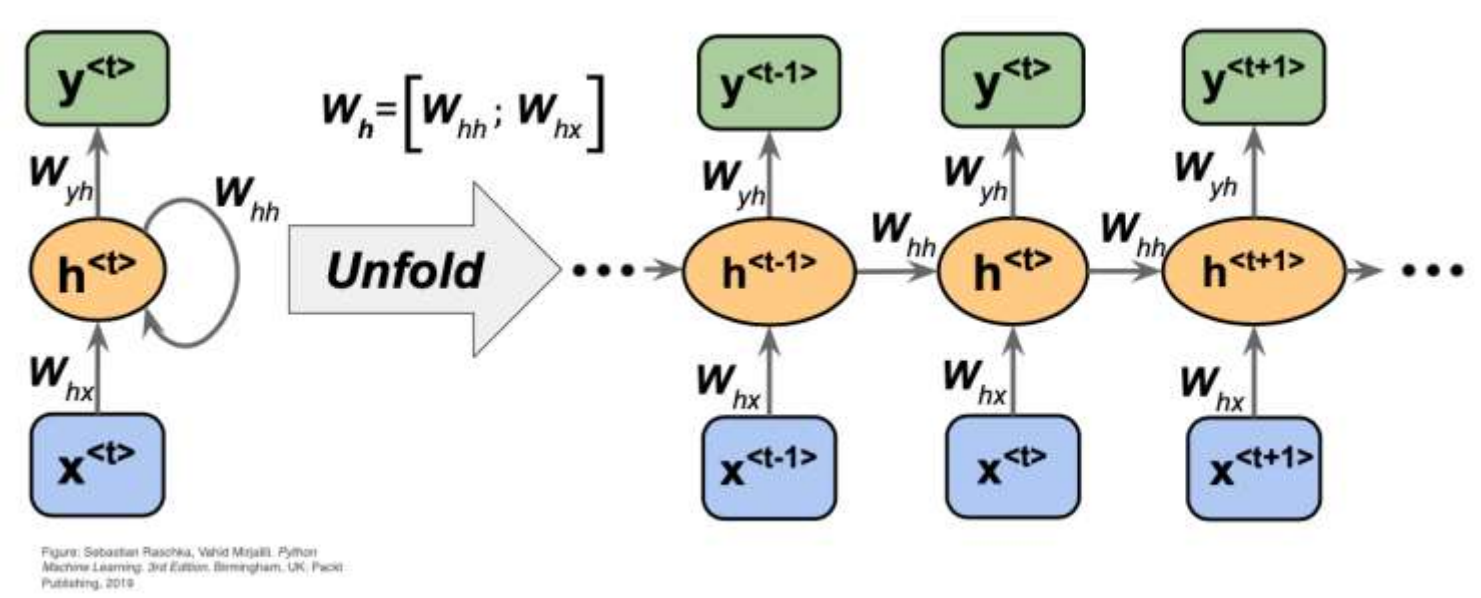
Version locale :

https://turinici.com/wp-content/uploads/cours/deep_learning/10_Minutes_to_Building_a_MobileNet_Image_Classifier.ipynb

Réseaux particuliers: réseaux récurrents (RNN)

Parfois nous avons besoin de traiter l'information qui est une suite de données, suite n'ayant pas de taille prédéfinie naturelle, ex. textes, audio, vidéo, langage (NLP), …

On utilise des réseaux récurrents (RNN)



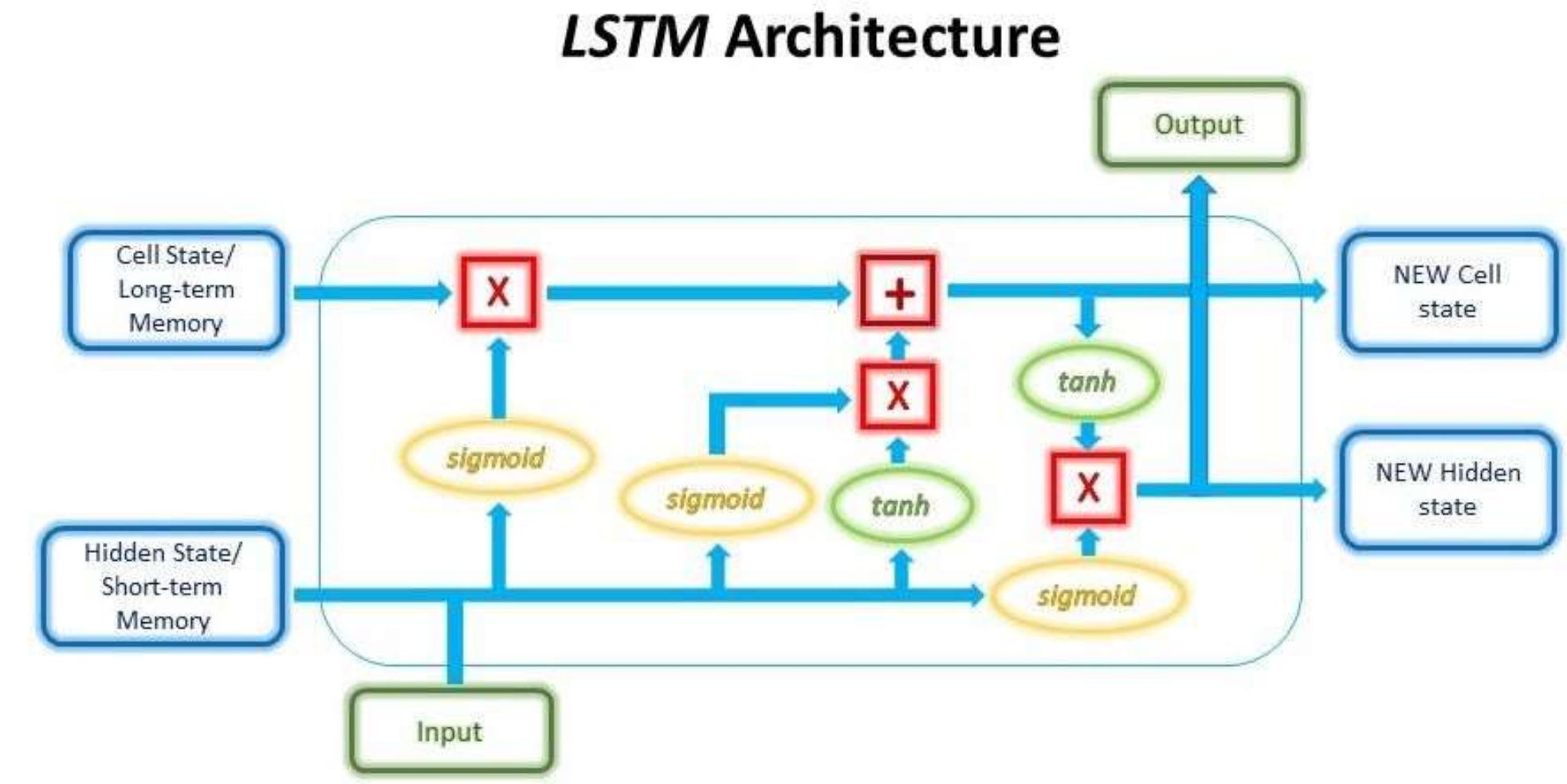
Ici les unités de base ne sont plus les neurones (« perceptrons ») mais des cellules RNN.

Idée générale: pour avoir de la mémoire il faut sortir du cadre Markovien et avoir donc des états internes.

L'opération de base sera donc (input X_t, état interne / caché h_t) -> (output Y_t, nouveau état interne h_{t+1}), plus précisément $h_{t+1} = f_W(h_t, X_t)$ même opération f_W à tout temps « t ».

Réseaux particuliers: réseaux récurrents **Long short-term memory (LSTM)** LSTM

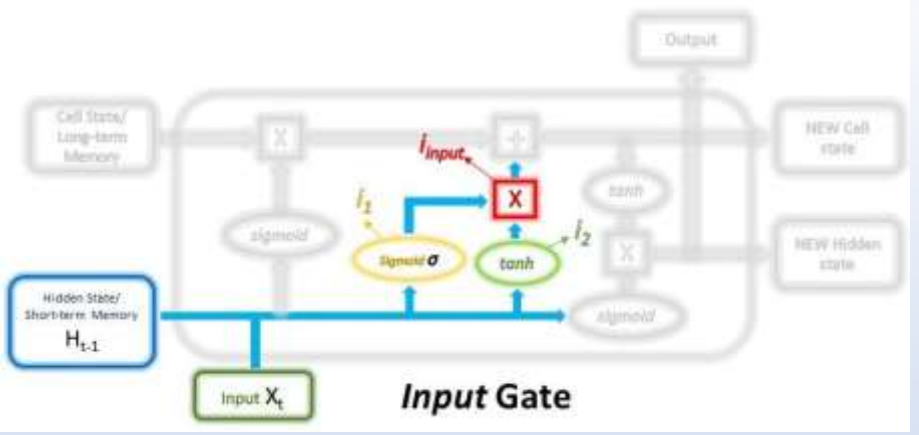
- Une architecture utilisée: **Long short-term memory (LSTM)**
- unité de base = cellule LSTM
- Deux types d'états internes: mémoire de longue durée (« état de la cellule » C_t) et de courte durée (« état caché » h_t)



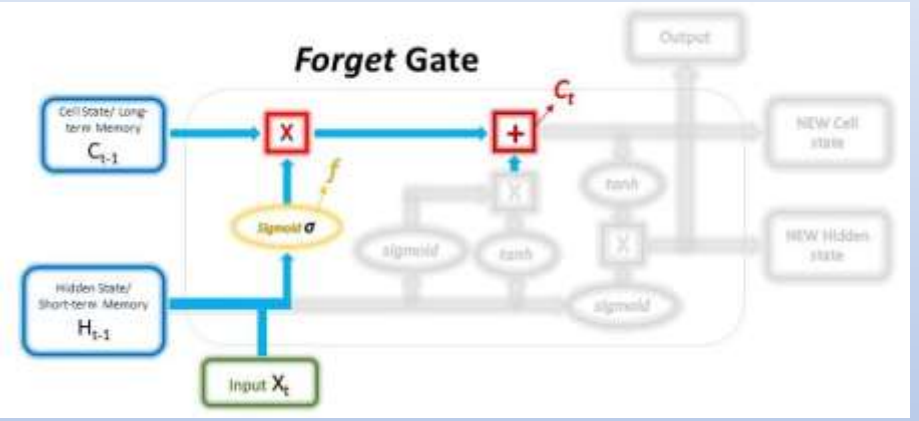
https://blog.floydhub.com/long-short-term-memory-from-zero-to-hero-with-pytorch/

LSTM

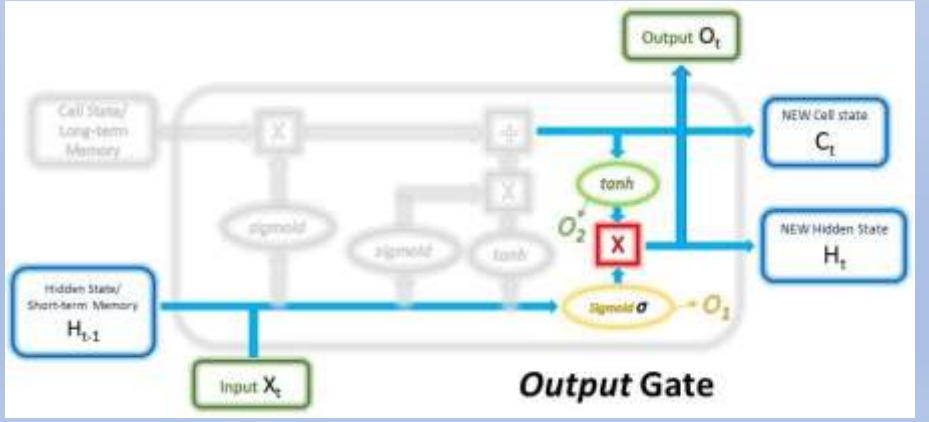
- **Input Gate**
- The input gate decides what **new information** will be stored in the **long-term memory**. It only works with the information from the current input and the short-term memory from the previous time step. Therefore, it has to **filter** out the information from these variables that are not useful.



- The **forget gate** decides which information from the **long-term memory** should be kept or discarded. This is done by multiplying the incoming long-term memory by a forget vector generated by the current input and incoming short-term memory.



- The output gate will take the current input, the previous short-term memory, and the newly computed long-term memory to produce the **new short-term memory**/hidden state which will be passed on to the cell in the next time step. The output of the current time step can also be drawn from this hidden state.



https://blog.floydhub.com/long-short-term-memory-from-zero-to-hero-with-pytorch/

LSTM

LSTM with a forget gate [edit]

The compact forms of the equations for the forward pass of an LSTM unit with a forget gate are:[1][9]

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

where the initial values are $c_0 = 0$ and $h_0 = 0$ and the operator $\circ$ denotes the Hadamard product (element-wise product). The subscript t indexes the time ste

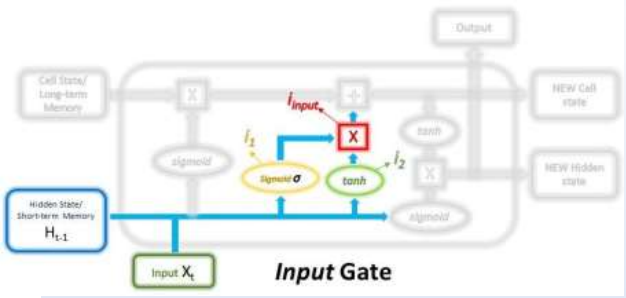
Variables [edit]

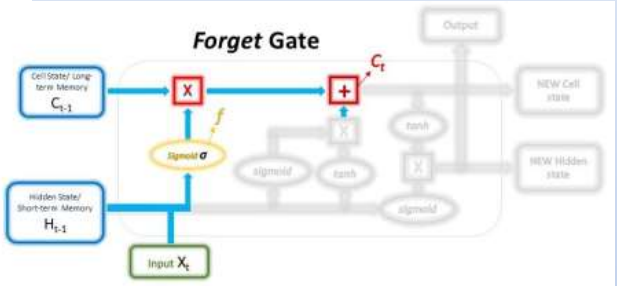
- $x_t \in \mathbb{R}^d$: input vector to the LSTM unit
- $f_t \in \mathbb{R}^h$: forget gate's activation vector
- $i_t \in \mathbb{R}^h$: input/update gate's activation vector
- $o_t \in \mathbb{R}^h$: output gate's activation vector
- $h_t \in \mathbb{R}^h$: hidden state vector also known as output vector of the LSTM unit
- $\tilde{c}_t \in \mathbb{R}^h$: cell input activation vector
- $c_t \in \mathbb{R}^h$: cell state vector
- $W \in \mathbb{R}^{h \times d}$, $U \in \mathbb{R}^{h \times h}$ and $b \in \mathbb{R}^h$: weight matrices and bias vector parameters which need to be learned during training

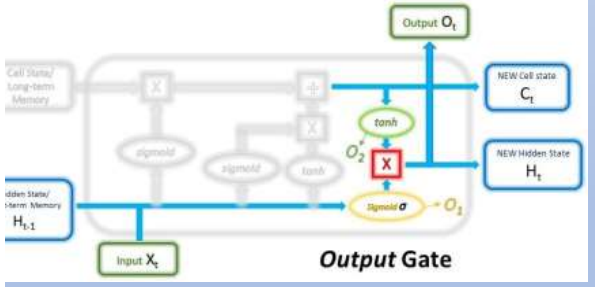
where the superscripts d and h refer to the number of input features and number of hidden units, respectively.

Activation functions [edit]

- σ_g: sigmoid function.
- σ_c: hyperbolic tangent function.
- σ_h: hyperbolic tangent function or, as the peephole LSTM paper[36][37] suggests, $\sigma_h(x) = x$.







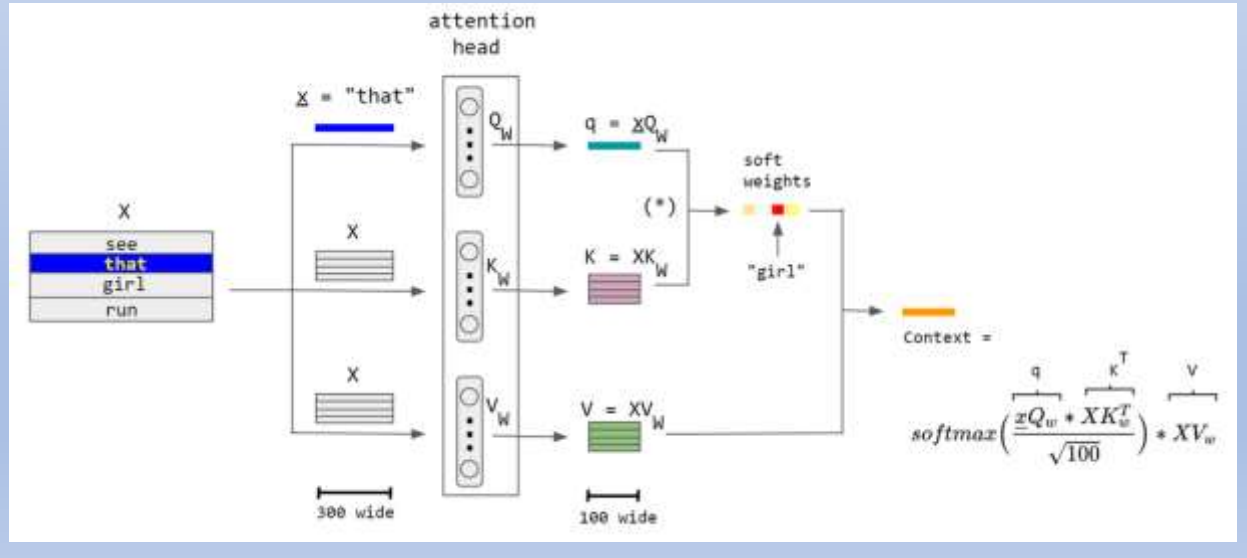
https://blog.floydhub.com/long-short-term-memory-from-zero-to-hero-with-pytorch/

Autres types de couche: « attention », « transformers »

- Cf. projets de C. Vincent

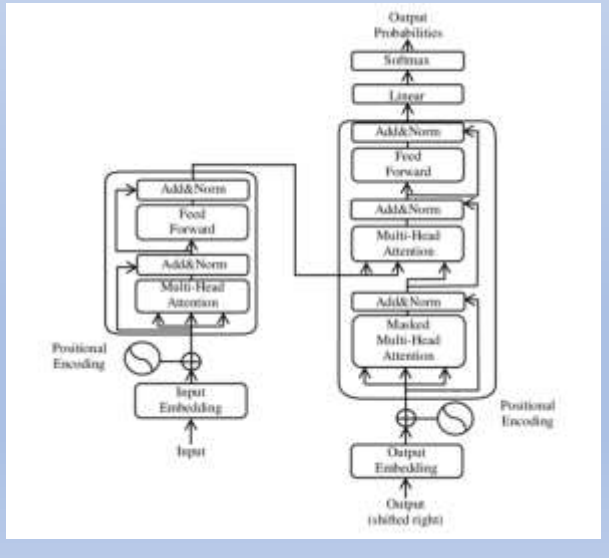
Attention / Transformers : pour des données de type séquence, afin de faire « attention » aux mots et au « contexte », prend comme modèle l'attention humaine (séquentielle), rend plus interprétable le modèle.

Attention



https://en.wikipedia.org/wiki/Attention_(machine_learning)

Transformer



https://en.wikipedia.org/wiki/Transformer_(machine_learning_model)#Scaled_dot-product_attention

Autres types de couches: « attention », « transformers »

Cf. exposés …

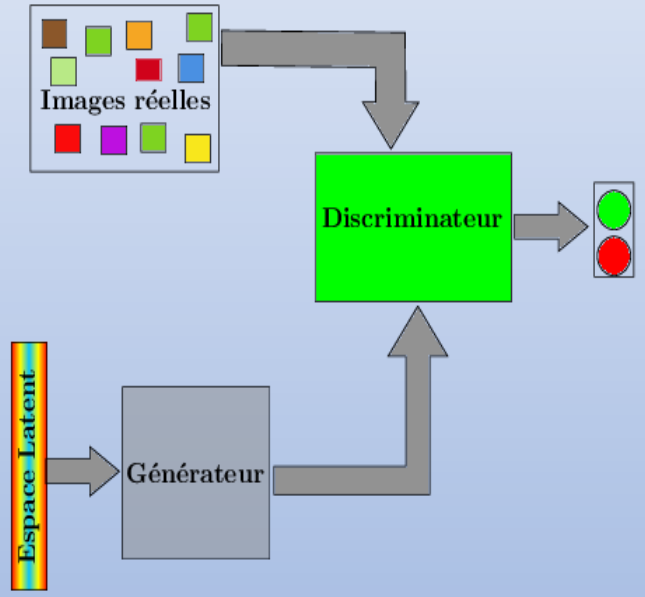
	Attention	Transformers
Architecture	Cf. slide précédent, input un ensemble d'éléments, output = un softmax de taille du nombre d'éléments	assez souvent encodeur/décodeur ou un de deux, plusieurs « têtes » attention, et « encoding positionnel »
Applications	Langage, segmentation d'image, séquences, …	Toutes les précédentes …

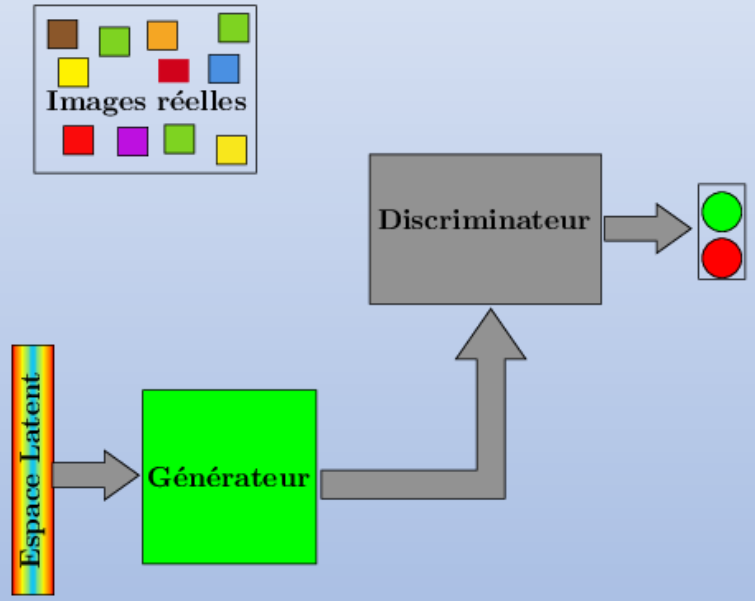
https://en.wikipedia.org/wiki/Attention_(machine_learning)

https://en.wikipedia.org/wiki/Transformer_(machine_learning_model)#Scaled_dot-product_attention

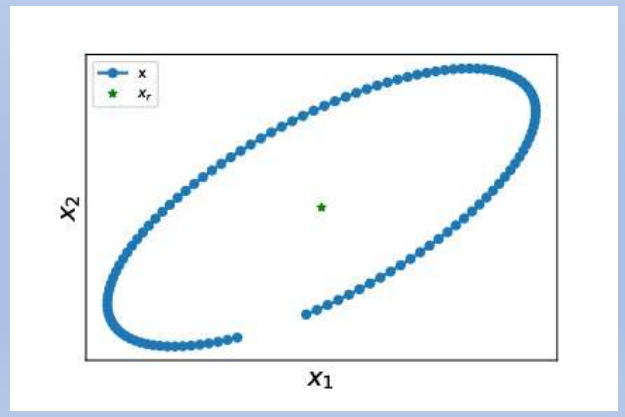
Réseaux particuliers: génératifs, GAN, stable diffusion

Generative Adversarial Networks : deux réseaux adversaires, le générateur qui crée des nouveaux objets, le discriminateur (critique) qui tente de trouver le faux





GANs: peuvent poser des problèmes de convergence, par exemple le « mode collapse ».



Source Gabriel Turinici :

[Convergence dynamics of Generative Adversarial Networks: the dual metric flows]

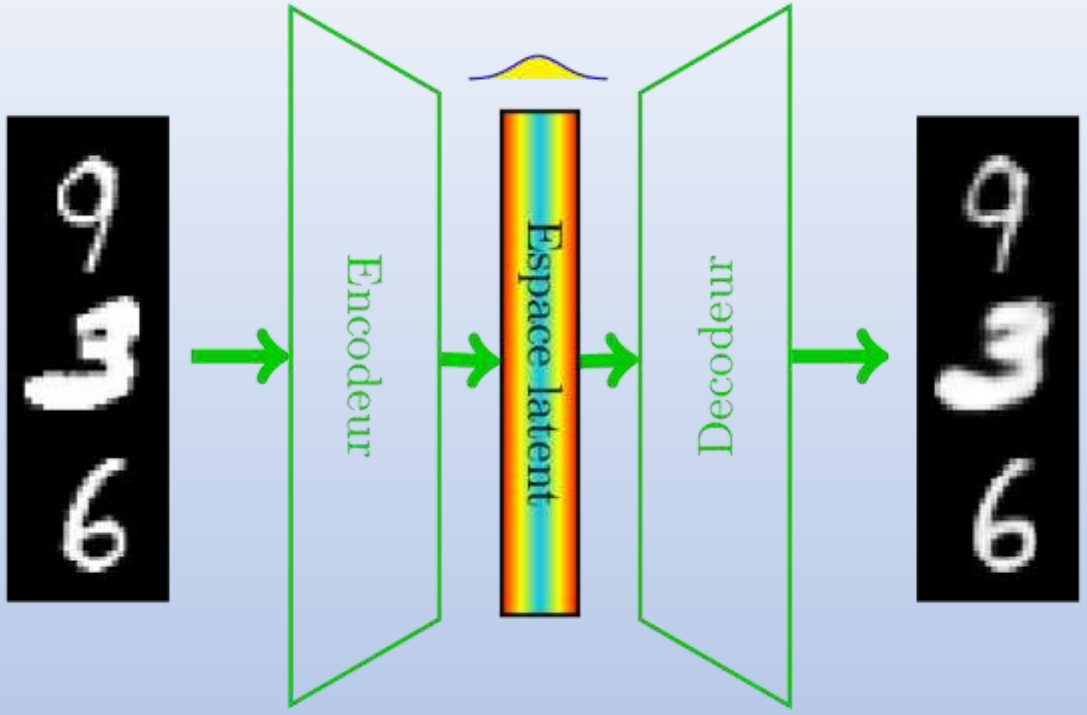
[Deepfakes & Algorithmes: Menace ou Opportunité?]

Réseaux particuliers: Variational AutoEncoders (VAE)

Ingrédients - encodeur $E(.)$ / décodeur $D(.)$: trouve une représentation condensée à l'aide de l'espace latent.

Espace latent: distribution gaussienne multi-dimensionnelle i.e. indépendance des traits, normalité

Training: besoin d'une fonction distance entre lois de proba (e.g. Kolmogorov-Smirnov, divergence, Radon-Sobolev (G.T.)

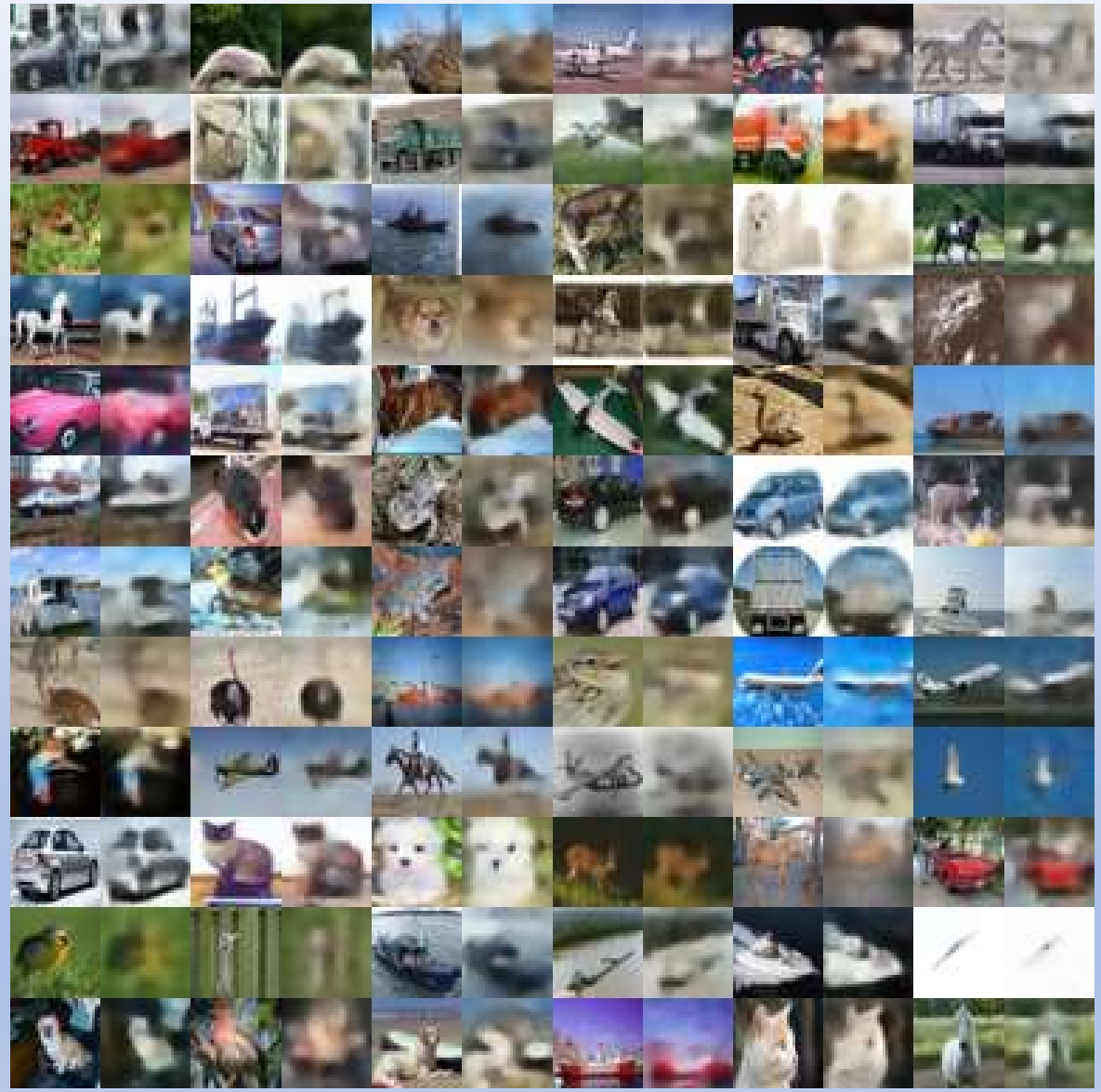


Fonction loss totale : moyenne sur les images en entrée $\omega \in \Omega$ représentées par une v.a. X de loi uniforme sur Ω de :

$$\underbrace{E_X\left[dist\left(X, D\left(E(X)\right)\right)^2\right]}_{\text{erreur de reconstruction sur l'espace réel}} + \mathrm{C}\ \underbrace{dist(loi\ E(X), loi\ cible\ e.g.\ Gausienne)^2}_{\text{conformite à la loi cible sur l'espace latent}}$$

Réseaux particuliers: Variational AutoEncoders (VAE)

- Deux étapes: reconstruction et génération



Source Gabriel Turinici : [X-Ray Sobolev Variational Auto-Encoders]

Réseaux particuliers: Variational AutoEncoders (VAE)

Deux étapes: reconstruction et génération, « side-effect » : interpolations

Source Gabriel Turinici : [X-Ray Sobolev Variational Auto-Encoders]

Réseaux particuliers: Variational AutoEncoders (VAE)

Exemple: **Convolutional Variational Autoencoder (du « Tensorflow tutorial »):** https://www.tensorflow.org/tutorials/generative/cvae

A faire:

- exécuter le code,

- ajouter une couche dans l'encodeur et le décodeur (attention à Conv2DTranspose, utiliser le bon stride, par ex stride=1)

- figurer (scatter plot) la distribution E(X) sur l'espace latent

```
def generate_latent_representation(model, epoch, test_sample):
  mean, logvar = model.encode(test_sample)
  z = model.reparameterize(mean, logvar)
  fig = plt.figure(figsize=(4, 4))
  plt.scatter(z[:,0],z[:,1],s=0.5)
  plt.savefig('latent_at_epoch_{:04d}.png'.format(epoch))
  plt.show()
```

Autre réseaux génératifs: Stable diffusion

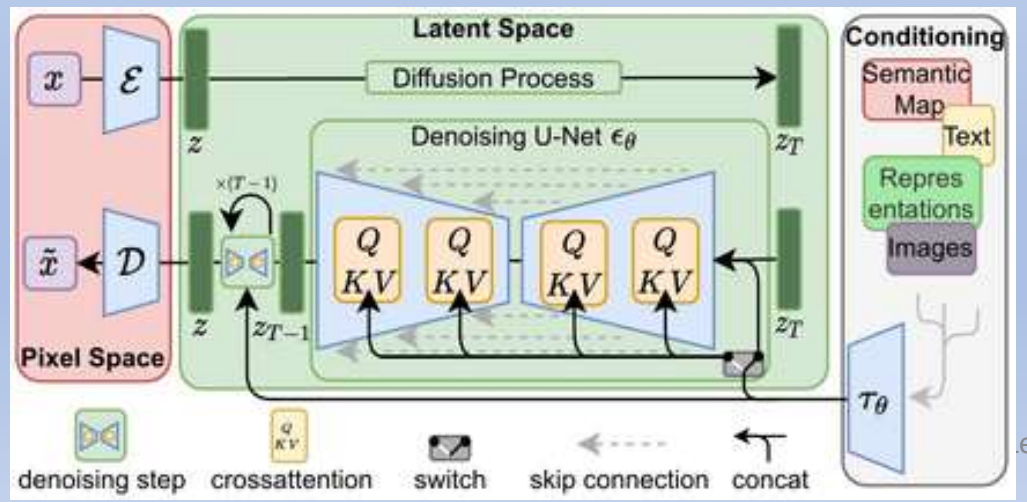
Idée: inverser le processus qui détériore une image par adjonction de bruit blanc.

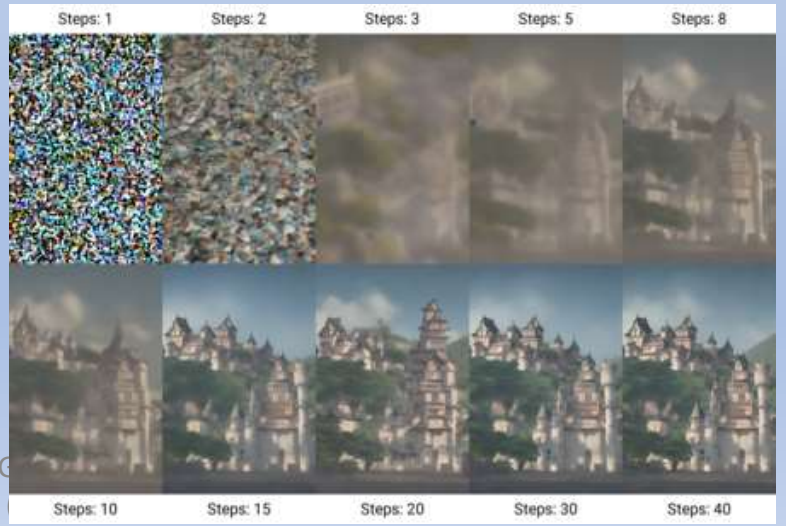
Mathématiquement une SDE est résolue chaque fois pour générer un objet nouveau.

La SDE est déterminée par un « potentiel » construit lors de l'entrainement.

Architecture: VAE+ denoising diffusion + prompt encoding (cf image) ou

- Un encodeur de texte : transforme le prompt en vecteur latent (une partie de celui-ci).
- Un modèle de diffusion: « débruite » l'image de manière itérative
- Un décodeur qui transforme l'image finale en une image de plus haute résolution (ex: 64x64-> 512x512





Stable Diffusion génère des images en débruitant de manière itérative.

Illustrations: wikipedia (jan 2024).

Autre réseaux génératifs: Stable diffusion

tutorial Tensoflow :
https://www.tensorflow.org/tutorials/generative/generate_images_with_stable_diffusion

```
!pip install keras_cv==0.8.1 tensorflow==2.15.0.post1 keras==2.15.0

import keras_cv
import keras
import matplotlib.pyplot as plt
model = keras_cv.models.StableDiffusion(img_width=512, img_height=512,
jit_compile=True)
images = model.text_to_image("photograph of an astronaut riding a horse",
batch_size=3,num_steps=50,seed=123)
def plot_images(images):
    plt.figure(figsize=(4*len(images),4))
    for i in range(len(images)):
        ax = plt.subplot(1, len(images), i + 1)
        plt.imshow(images[i])
        plt.axis("off")
plot_images(images)
```

Mettre ici votre prompt

Suite

- regarder l'actualité sur arxiv, conférences, journaux scientifiques
- Ex: adversarial examples cf. https://www.tensorflow.org/tutorials/generative/adversarial_fgsm?hl=en

- cf. aussi le cours de « reinforcement learning »