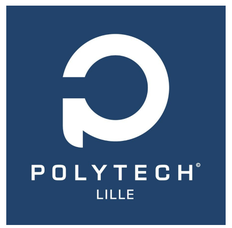
Introduction au Deep Learning

par Thomas Besnier, d'après le cours de Kevin Feghoul

CRIStAL - Université de Lille

2023-2024



Contact

Mail: thomas.besnier@univ-lille.fr

Support cours/TPs: https://tbesnier.github.io/teaching/

Objectifs du cours (22h)

- Être capable d’expliquer comment fonctionne le deep learning dans les grandes lignes

- Identifier des cas d’utilisation: quand faire du deep ?

- Etre capable d’identifier les blocs principaux d’un algo + coder des modèles de base

Objectifs du cours (22h)

- Être capable d’expliquer comment fonctionne le deep learning dans les grandes lignes

- Identifier des cas d’utilisation: quand faire du deep ?

- Etre capable d’identifier les blocs principaux d’un algo + coder des modèles de base

Évaluation: **TPs (40%) + Exam (60%)**

Programme du cours

- Introduction et contexte
- Réseaux de neurones et optimisation
- Deep Computer Vision
- Deep Generative model
- Deep Sequence model

Programme du cours

- Introduction et contexte
- Réseaux de neurones et optimisation
- Deep Computer Vision
- Deep Generative model
- Deep Sequence model

Qu’est ce que “le deep” ?

Qu’est ce que “le deep” ?

- Une représentation complexe des données + grande quantité

Qu’est ce que “le deep” ?

- Une représentation complexe des données + grande quantité

- Un (très) gros modèle statistique: réseaux de neurones

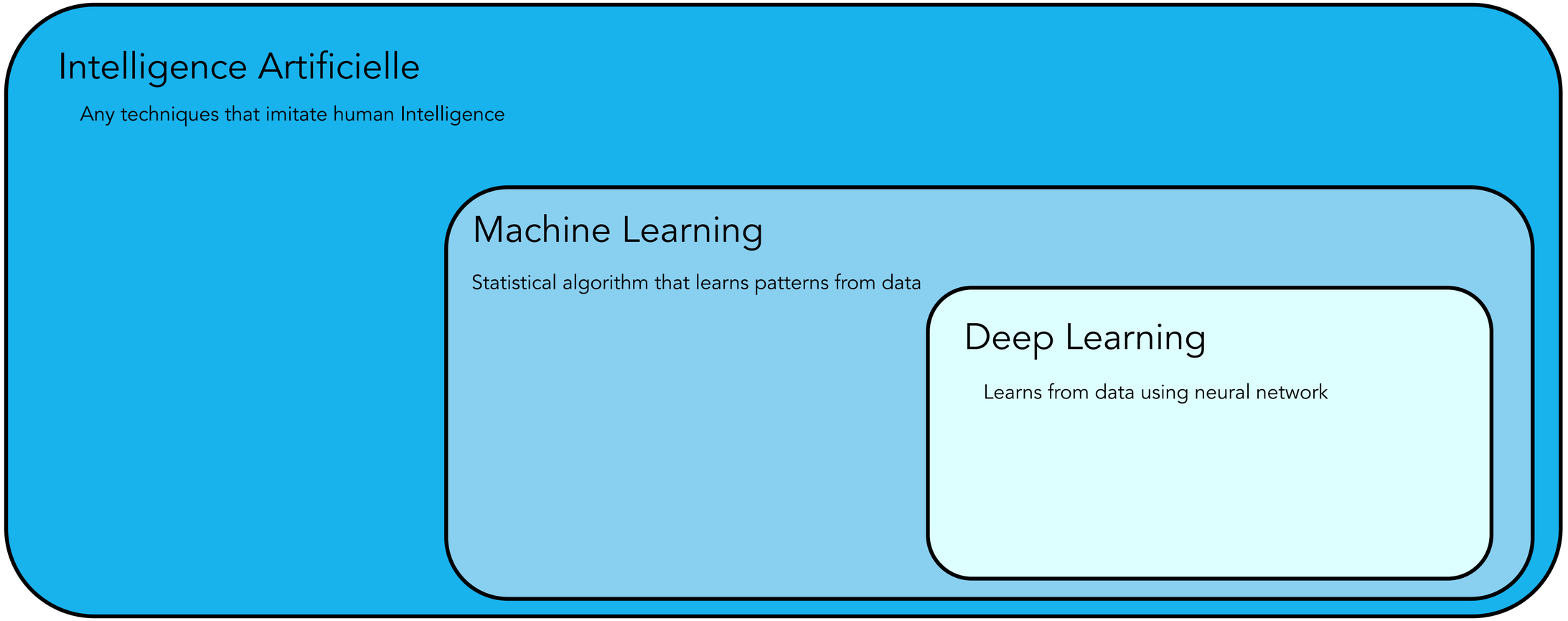
Qu’est ce que “le deep” ?

- Une représentation complexe des données + grande quantité

- Un (très) gros modèle statistique: réseaux de neurones

- Differentiation automatique

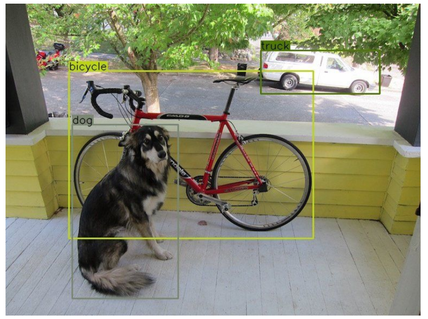
Qu’est ce que “le deep” ?

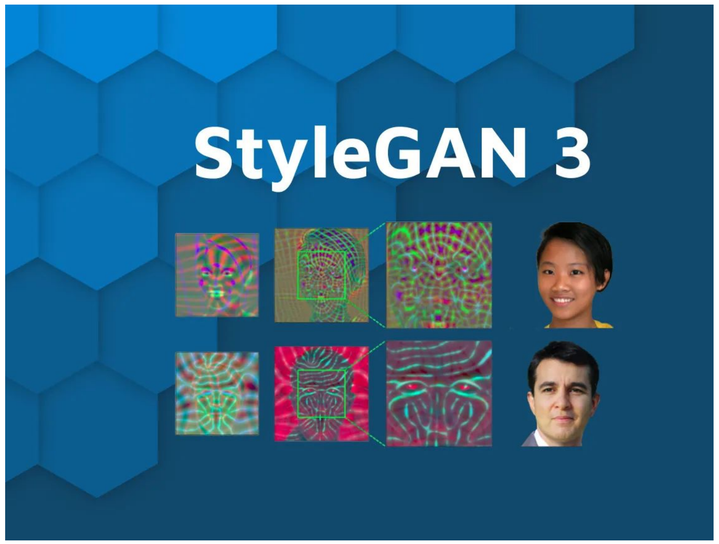


Pourquoi maintenant ?

Pourquoi maintenant ?

- Meilleurs algo





Pourquoi maintenant ?

- Meilleurs algo

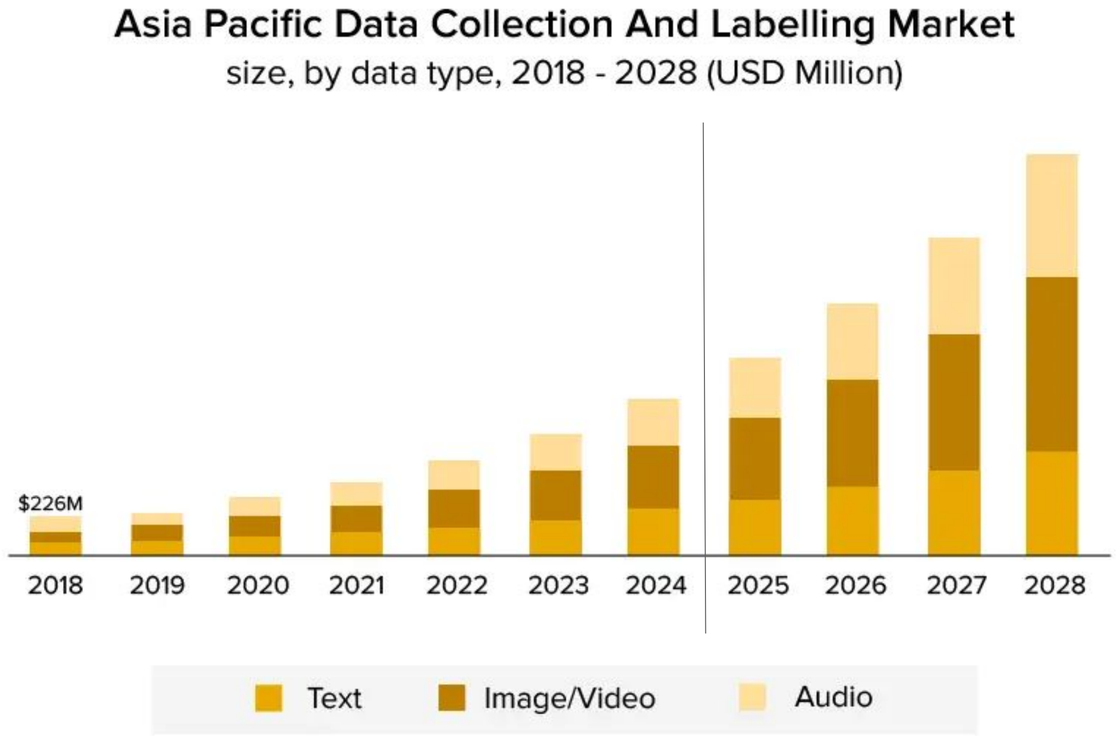
- Meilleurs hardware

Pourquoi maintenant ?

- Meilleurs algo

- Meilleurs hardware

- **Plus de données labellisées**



appinventiv

Pourquoi maintenant ?

- Meilleurs algo

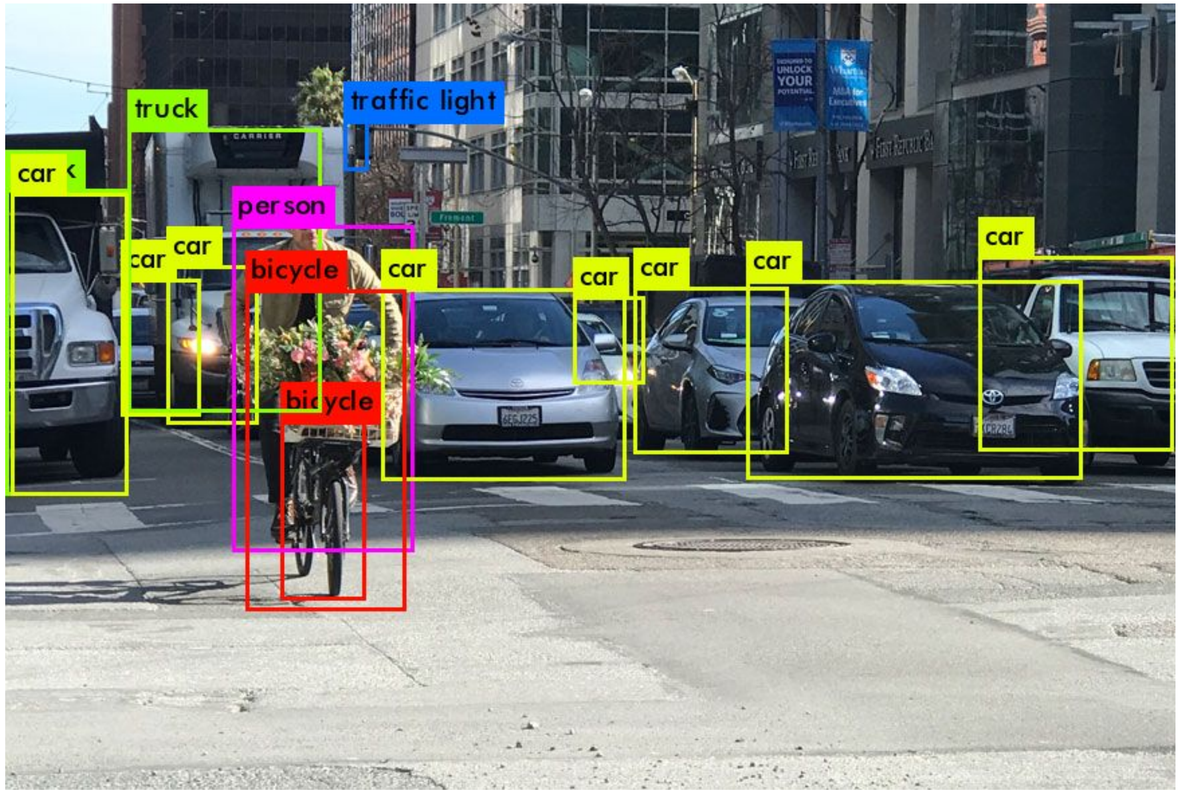
- Meilleurs hardware

- Plus de données labellisées

- Outils open source et accessibles

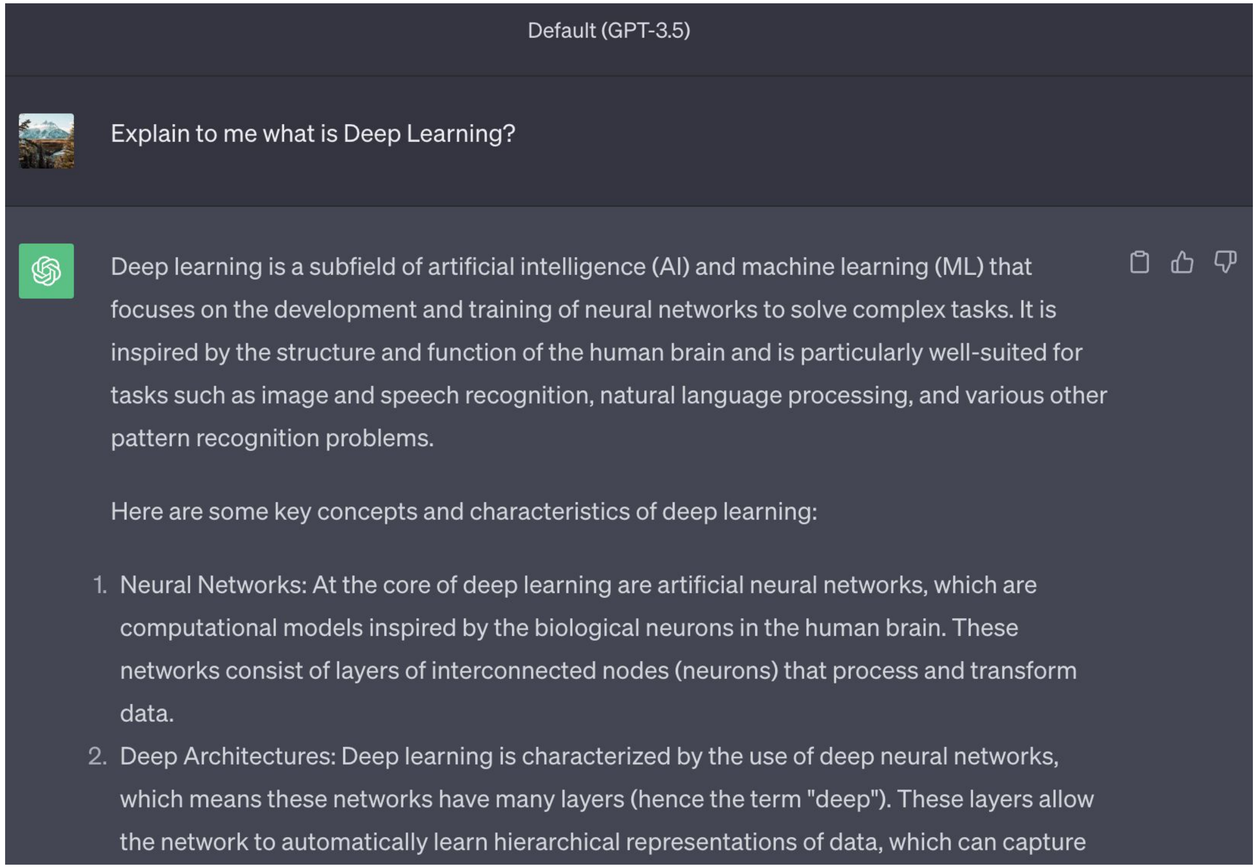
Progrès récents

Compréhension d’images



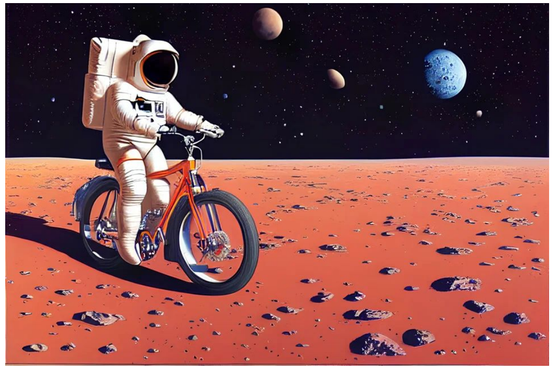
Modèles génératif: ChatGPT

Prompt to text generation



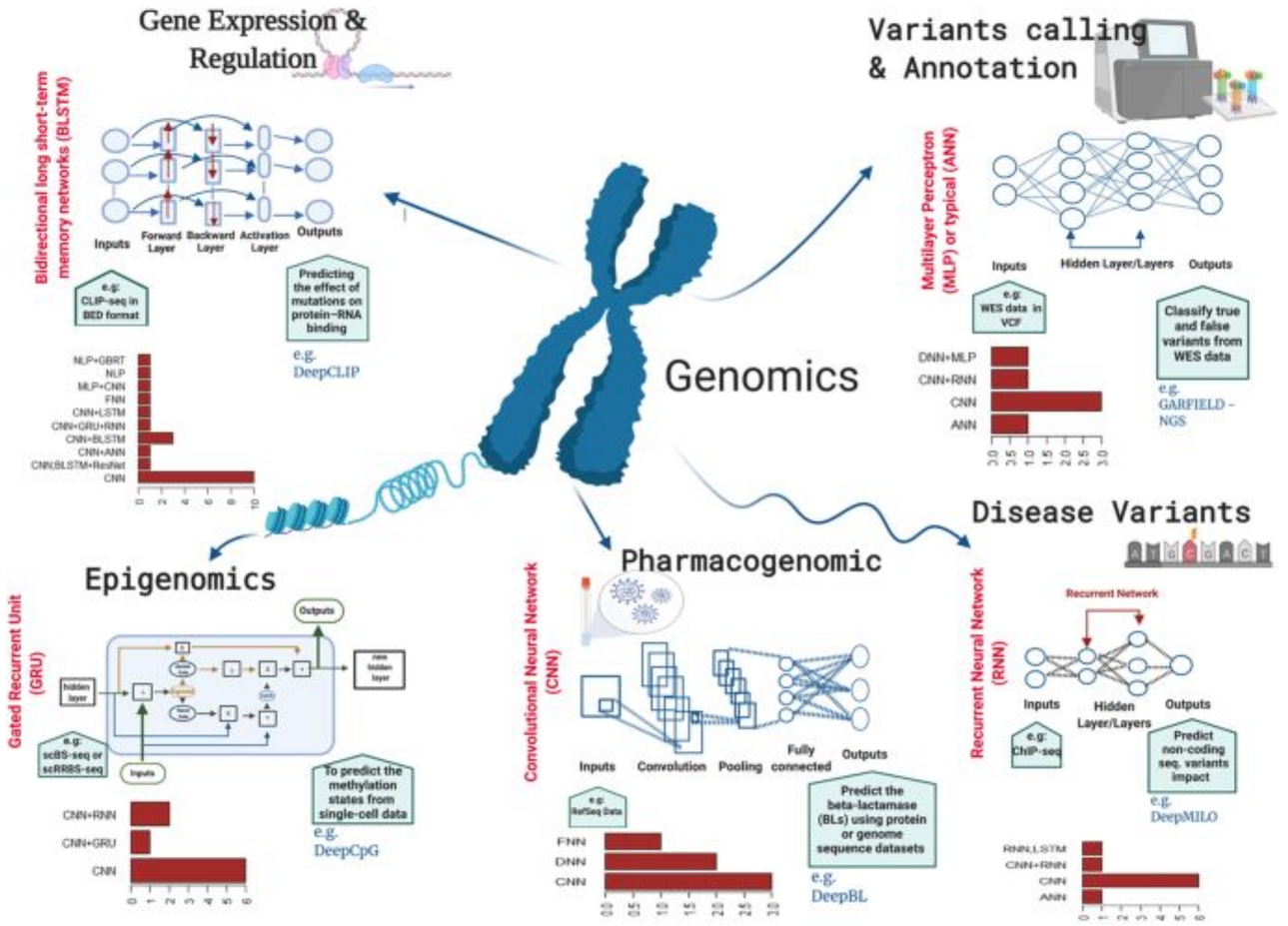
Modèle génératif: Stable Diffusion

Text-to-image generation



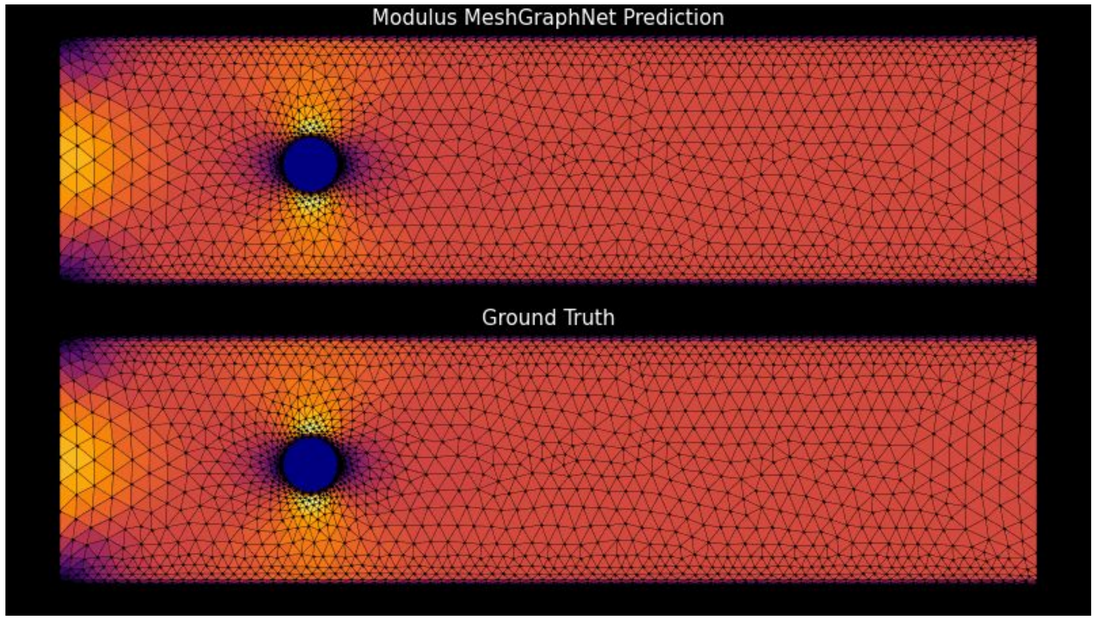
Utilisation en sciences

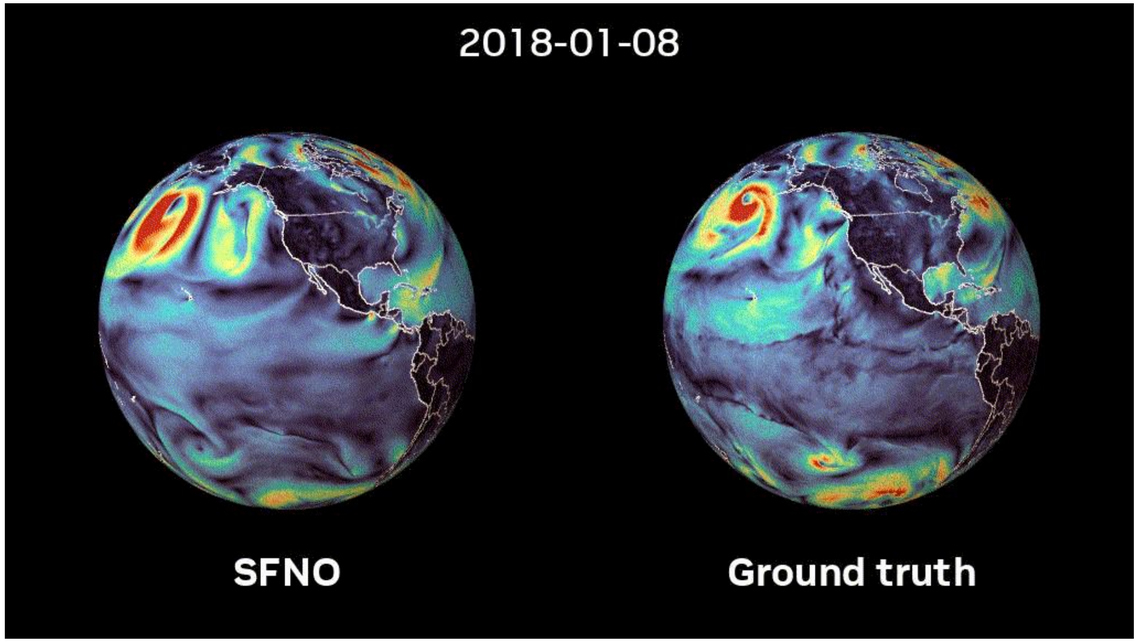
Génomique



Utilisation en sciences

Simulation en chimie / Physique





Comment ça marche ?

Le but de l’apprentissage supervisé est de construire une fonction de décision f qui fonctionne bien, vite et garantie de bonne performance :

Comment ça marche ?

Le but de l'apprentissage supervisé est de construire une fonction de décision f qui fonctionne bien, vite et garantie de bonne performance :

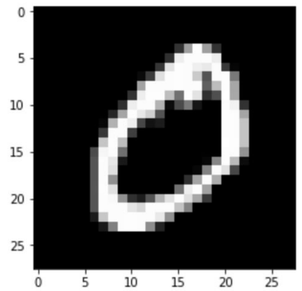
$$\begin{array}{rccc} f & : & \mathcal{X} & \rightarrow & \mathcal{Y} \\ & & x & \mapsto & y = f(x) \end{array}$$

$\mathcal{Y} = \mathbb{R}$ **Problème de régression**

$\mathcal{Y} = \{0, 1\}$ **Problème de classification binaire**

$\mathcal{Y} = \{1, 2, \ldots, C\}, C > 2$ **Problème de classification multi-classe**

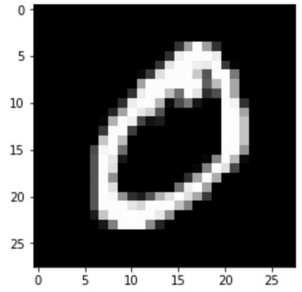
Comment ça marche ?



$x \in$

$y \in$

Comment ça marche ?



$$x \in \{0, ...1, .., 0\}^{28 \times 28}$$

$$y \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

Comment ça marche ?

Cat

Dog

$$y = f(x)$$

$$x \in$$

$$y \in$$

Comment ça marche ?

Cat

Dog

$$y = f(x)$$

$$x \in \{0, ...1, .., 0\}^{28 \times 28 \times 3}$$

$$y \in \{0,1\}$$

Comment ça marche ?

f

$$y = f(x) = 254,500 \text{ €}$$

$$x \in$$

$$y \in$$

Comment ça marche ?

f

$$y = f(x) = 254,500 \text{ €}$$

$$x \in \{0, 1, ..., 255\}^{4 \times 64 \times 64 \times 3}$$

$$y \in \mathbb{R}_+$$

Comment ça marche ?

f

$x \in$

$y \in$

Comment ça marche ?

f

$$x \in \{0, \ldots, 255\}^{n \times m} \qquad y \in \{0, 1\}^{n \times m}$$

Comment ça marche ?

Recette de l’entraînement d’un modèle

Comment ça marche ?

Recette de l’entraînement d’un modèle

1. Préparer la donnée → Dataloader

Comment ça marche ?

Recette de l’entraînement d’un modèle

1. Préparer la donnée → Dataloader

1. Définir / construire le modèle

Comment ça marche ?

Recette de l’entraînement d’un modèle

1. Préparer la donnée → Dataloader

1. Définir / construire le modèle

1. Définir un objectif

Comment ça marche ?

Recette de l’entraînement d’un modèle

1. Préparer la donnée → Dataloader

1. Définir / construire le modèle

1. Définir un objectif

Comment ça marche ?

Recette de l’entraînement d’un modèle

1. Préparer la donnée → Dataloader

1. Définir / construire le modèle

1. Définir un objectif

1. Optimiser les paramètres

Comment ça marche ?

Recette de l’entraînement d’un modèle

1. Préparer la donnée → Dataloader

1. Définir / construire le modèle

1. Définir un objectif

1. Optimiser les paramètres

1. Evaluer

Comment ça marche ?

Recette de l’entraînement d’un modèle

1. Préparer la donnée → Dataloader
1. Définir / construire le modèle
1. Définir un objectif
1. Optimiser les paramètres
1. Evaluer

Programme du cours

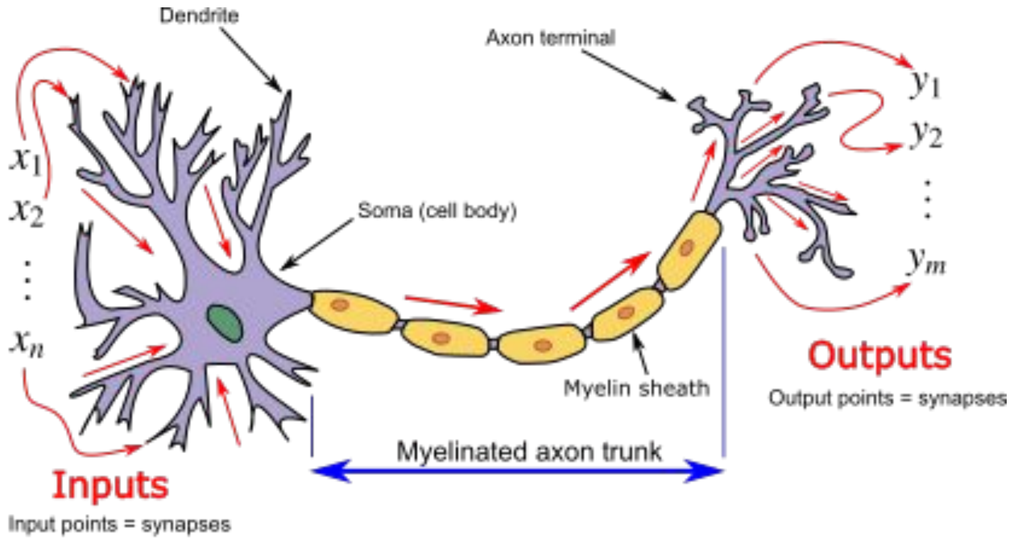
- Introduction et contexte
- Réseaux de neurones et optimisation
- Deep Computer Vision
- Deep Generative model
- Deep Sequence model

Comment ça marche ?

Neurone artificiel / Perceptron

Comment ça marche ?

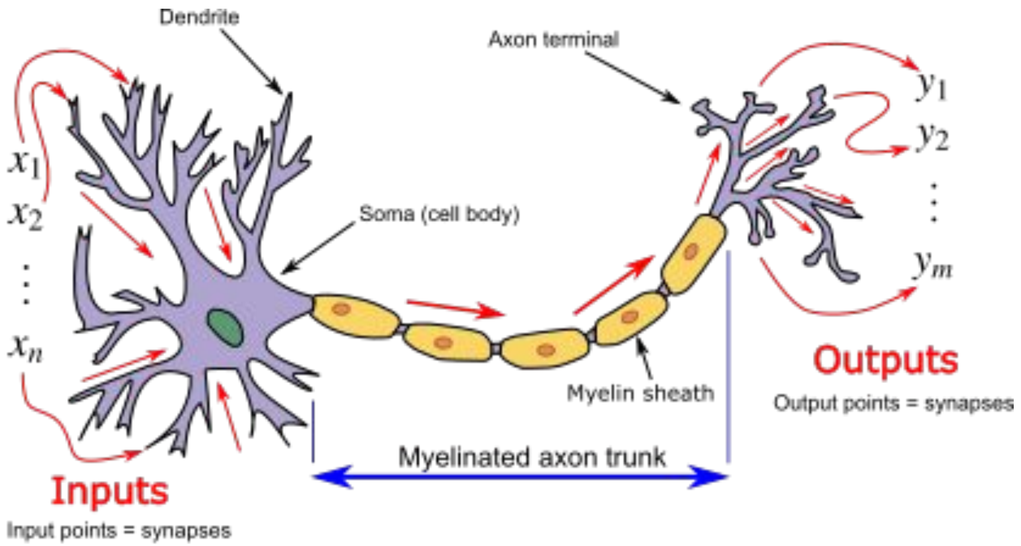
Neurone artificiel / Perceptron

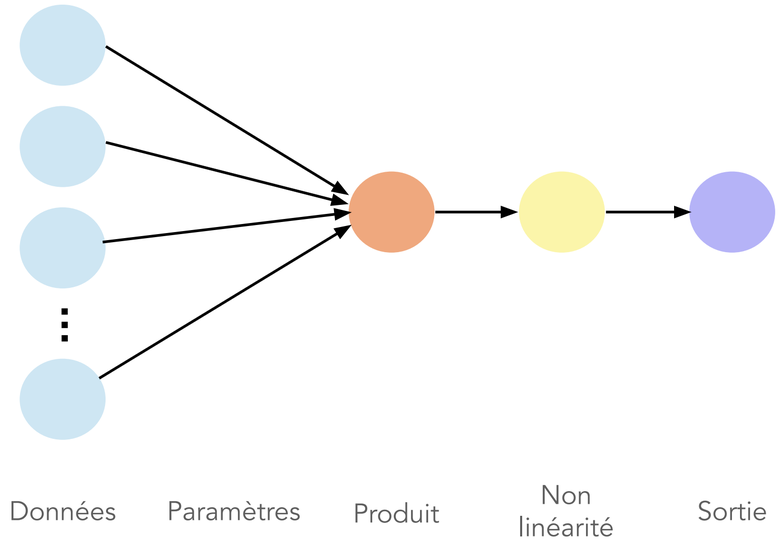


- Soma: intègre les signaux provenant d'autres neurones.
- Dendrites : reçoivent les signaux d'autres neurones.
- Axone : transporte le signal vers d'autres neurones.
- Potentiel d'Action : signal électrique qui détermine si le neurone transmet de l'information à d'autres neurones ou non.
- Synapses : sont les connexions entre les neurones où les neurotransmetteurs sont libérés. Ces connexions déterminent la force du signal transmis entre les neurones.

Source : https://en.wikipedia.org/wiki/Biological_neuron_model

Analogie entre neurone biologique et formel



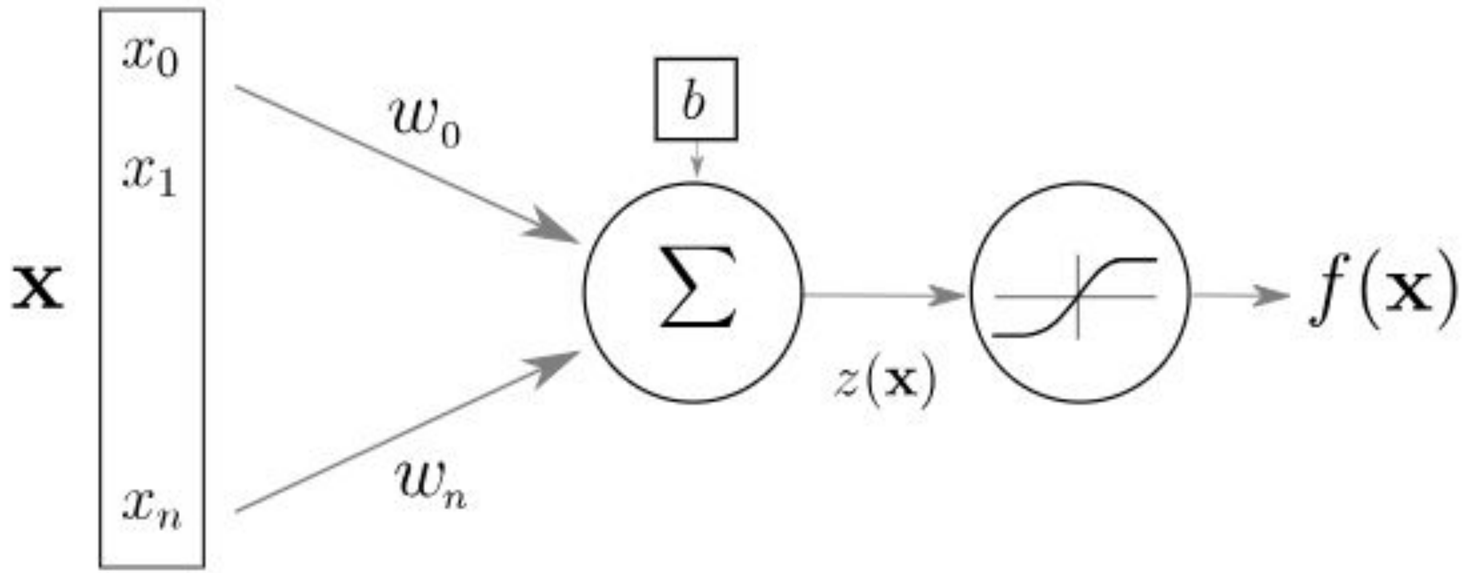


Le Perceptron

Source : https://en.wikipedia.org/wiki/Biological_neuron_model

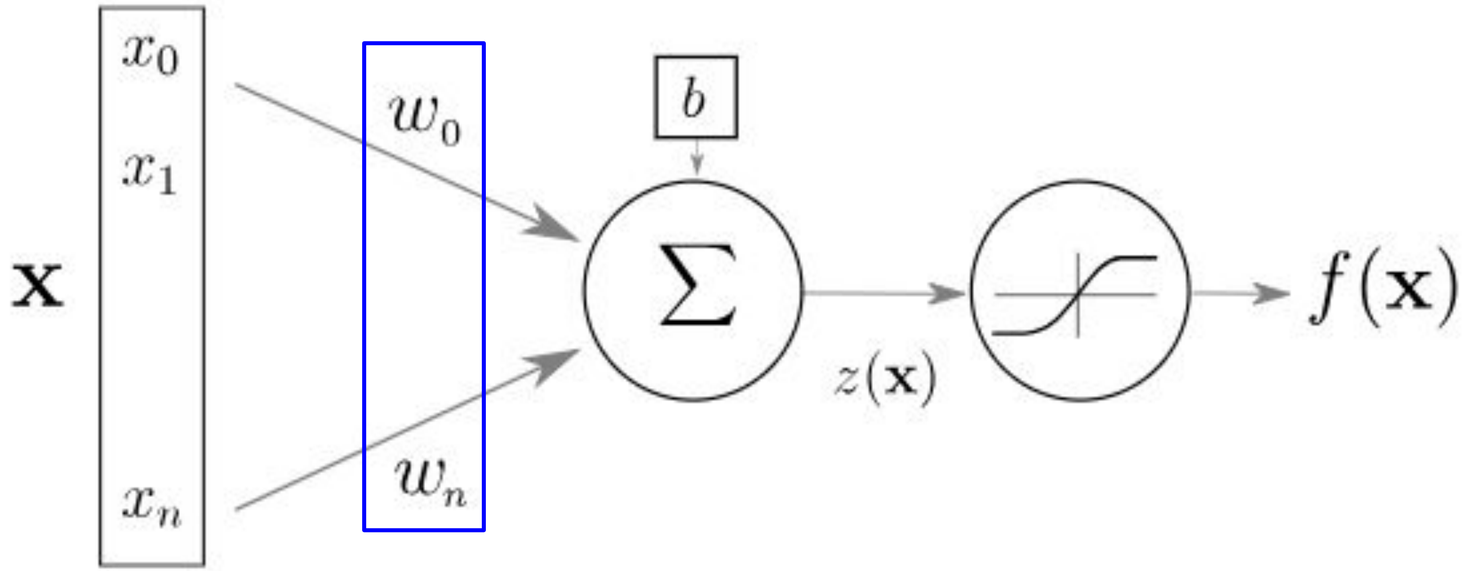
Comment ça marche ?

Neurone artificiel / Perceptron



Comment ça marche ?

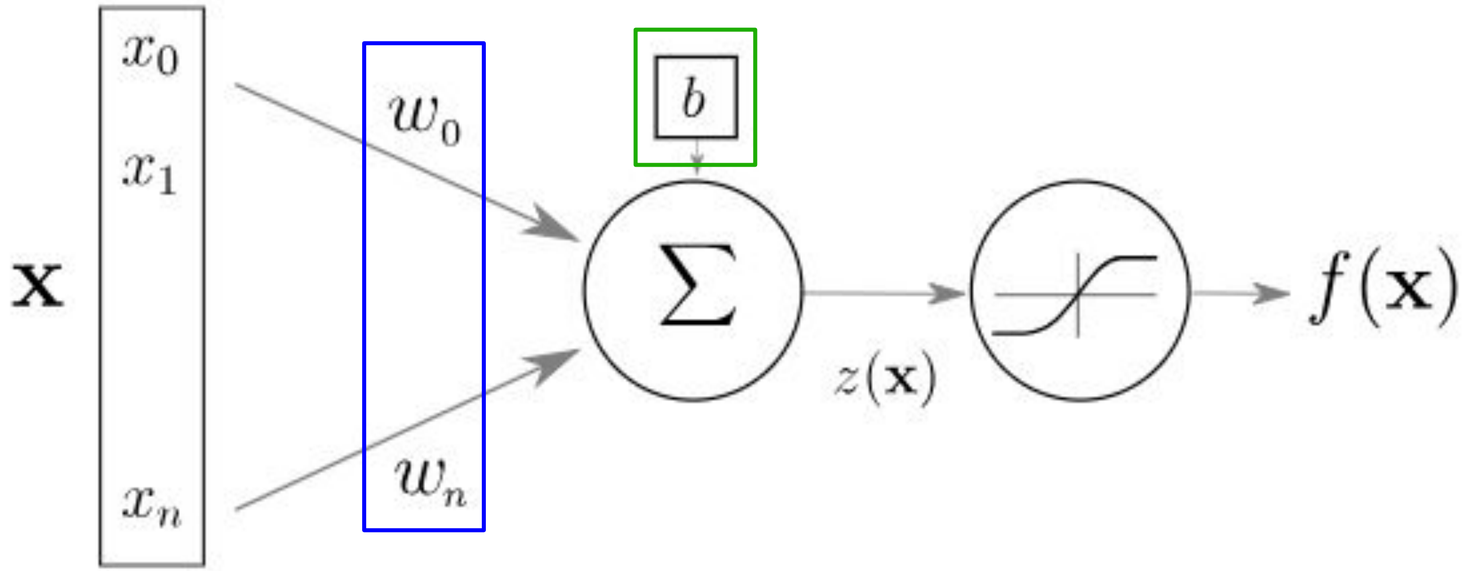
Neurone artificiel / Perceptron



$$f(x) = w^T x$$

Comment ça marche ?

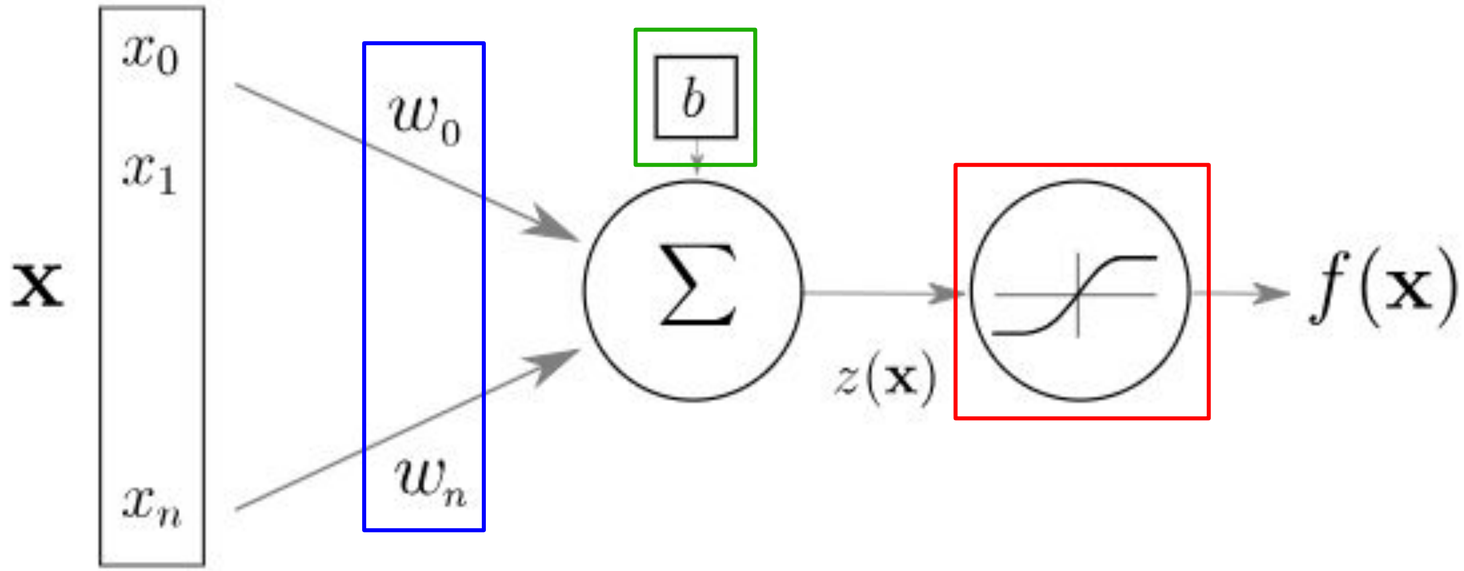
Neurone artificiel / Perceptron



$$f(x) = w^T x + b$$

Comment ça marche ?

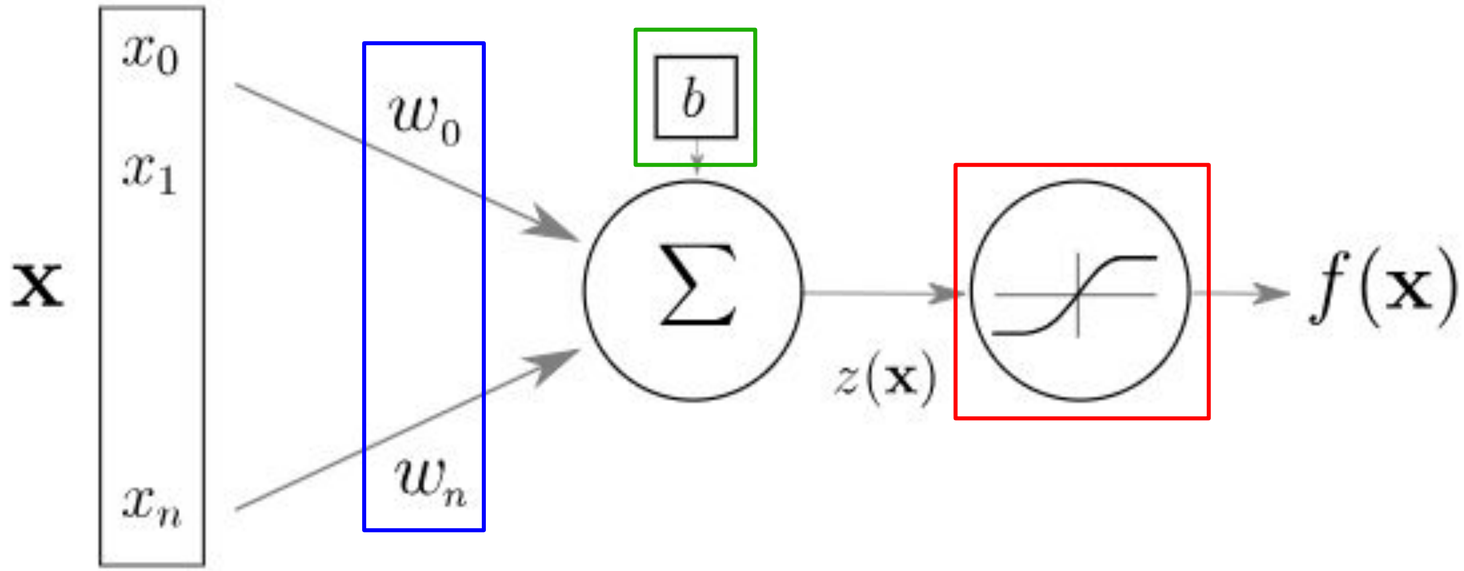
Neurone artificiel / Perceptron



$$f(x) = g(w^T x + b)$$

Comment ça marche ?

Neurone artificiel / Perceptron



$$f(x) = g(w^T x + b)$$

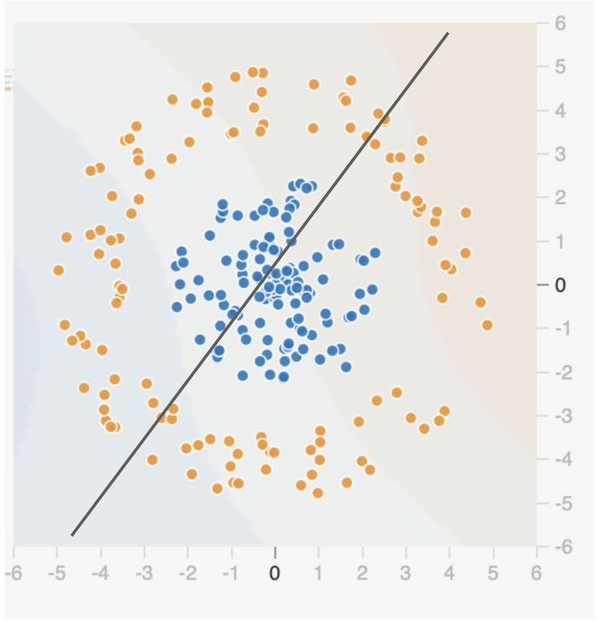
→ **Régression logistique**

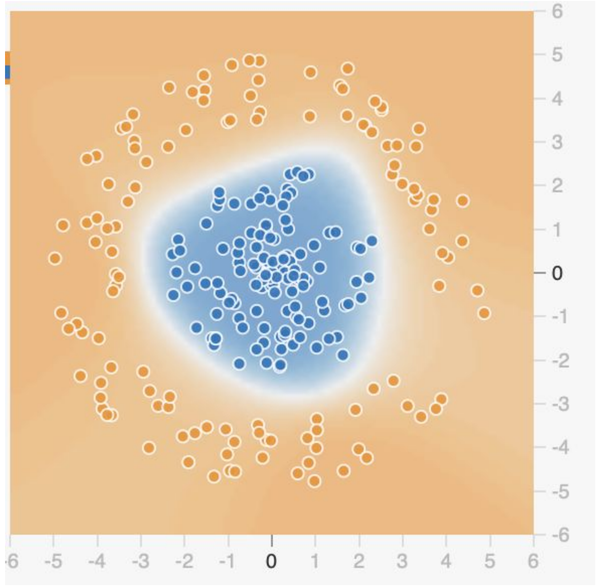
Importance des fonctions d’activation

- Introduisent de la non-linéarité, permettant au réseau de neurones d'identifier des motifs complexes au sein des données.
- Sans fonction d’activation, le réseau de neurones serait équivalent à un modèle linéaire.
- De nombreux problèmes du monde réel impliquent des modèles et des relations complexes qui ne peuvent être capturés que par des fonctions non linéaires.

Importance des fonctions d’activation

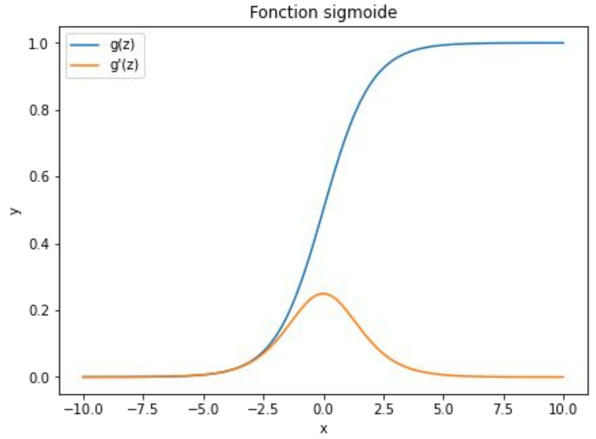
- Introduisent de la non-linéarité, permettant au réseau de neurones d'identifier des motifs complexes au sein des données.
- Sans fonction d’activation, le réseau de neurones serait équivalent à un modèle linéaire.
- De nombreux problèmes du monde réel impliquent des modèles et des relations complexes qui ne peuvent être capturés que par des fonctions non linéaires.





Importance des fonctions d’activation

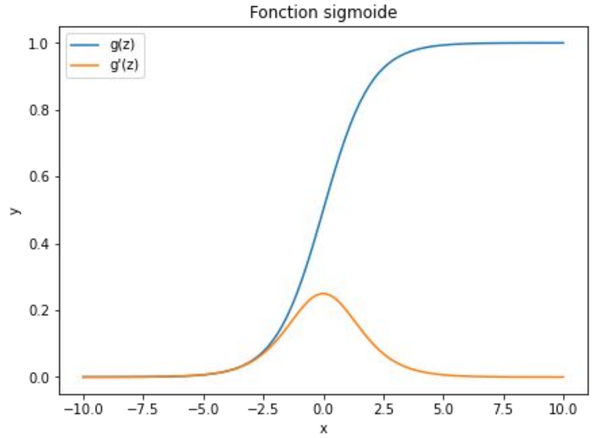
- Introduisent de la non-linéarité, permettant au réseau de neurones d'identifier des motifs complexes au sein des données.
- Sans fonction d’activation, le réseau de neurones serait équivalent à un modèle linéaire.
- De nombreux problèmes du monde réel impliquent des modèles et des relations complexes qui ne peuvent être capturés que par des fonctions non linéaires.



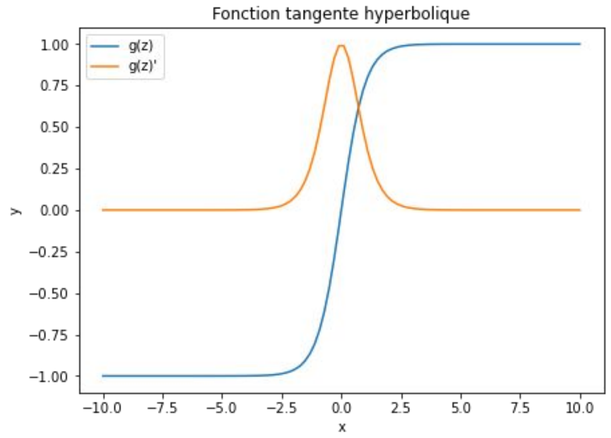

```
torch.nn.functional.sigmoid(z)
```

Importance des fonctions d’activation

- Introduisent de la non-linéarité, permettant au réseau de neurones d'identifier des motifs complexes au sein des données.
- Sans fonction d’activation, le réseau de neurones serait équivalent à un modèle linéaire.
- De nombreux problèmes du monde réel impliquent des modèles et des relations complexes qui ne peuvent être capturés que par des fonctions non linéaires.



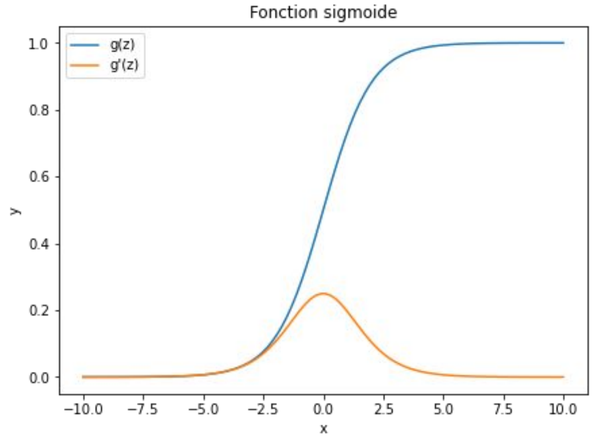

```
torch.nn.functional.sigmoid(z)
```



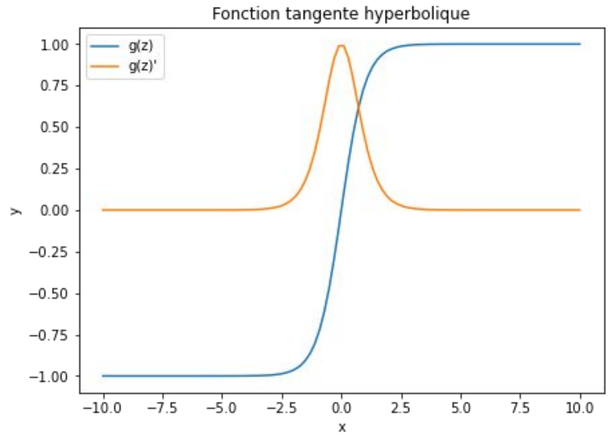

```
torch.nn.functional.tanh(z)
```

Importance des fonctions d’activation

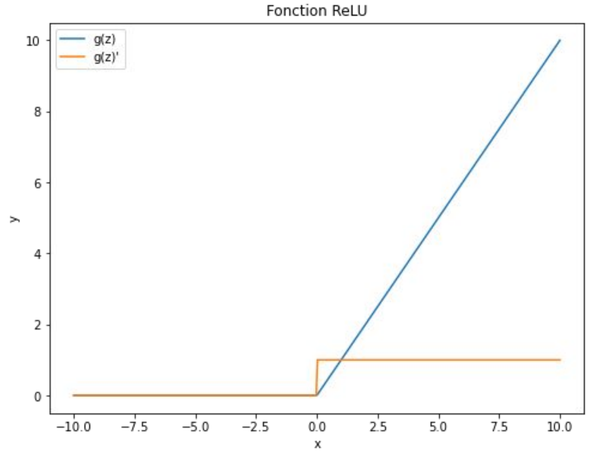
- Introduisent de la non-linéarité, permettant au réseau de neurones d'identifier des motifs complexes au sein des données.
- Sans fonction d’activation, le réseau de neurones serait équivalent à un modèle linéaire.
- De nombreux problèmes du monde réel impliquent des modèles et des relations complexes qui ne peuvent être capturés que par des fonctions non linéaires.




```
torch.nn.functional.sigmoid(z)
```



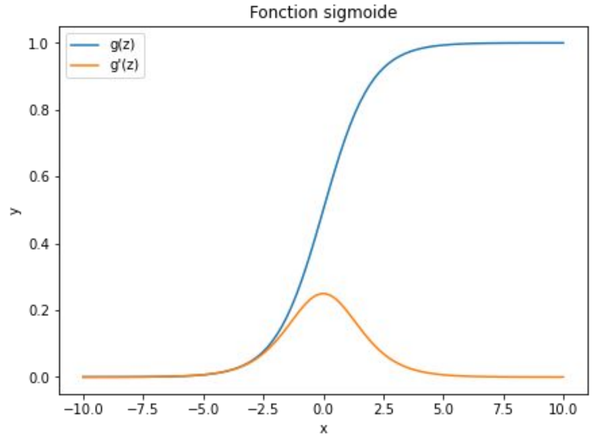

```
torch.nn.functional.tanh(z)
```



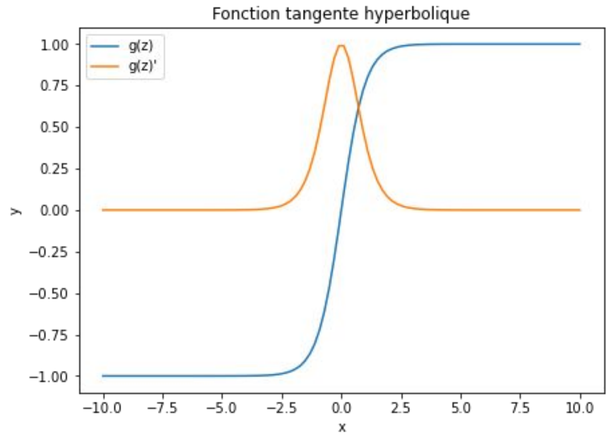

```
torch.nn.functional.relu(z)
```

Importance des fonctions d’activation

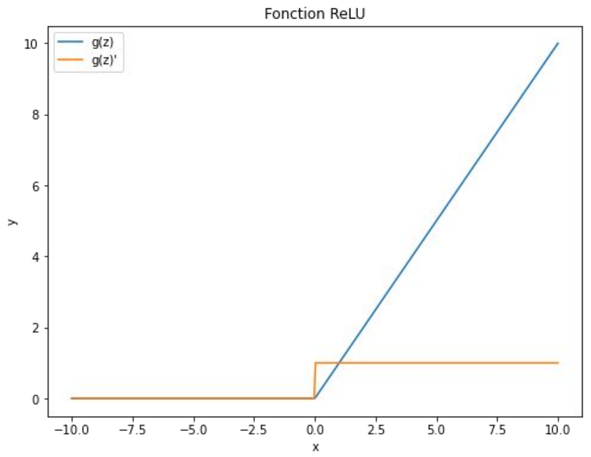
- Introduisent de la non-linéarité, permettant au réseau de neurones d'identifier des motifs complexes au sein des données.
- Sans fonction d’activation, le réseau de neurones serait équivalent à un modèle linéaire.
- De nombreux problèmes du monde réel impliquent des modèles et des relations complexes qui ne peuvent être capturés que par des fonctions non linéaires.



`torch.nn.functional.sigmoid(z)`

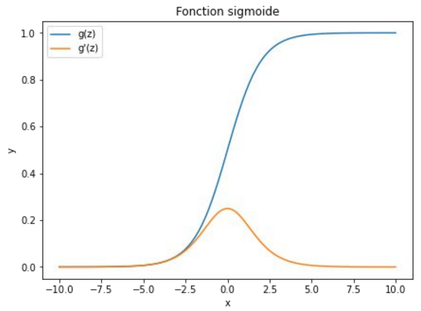
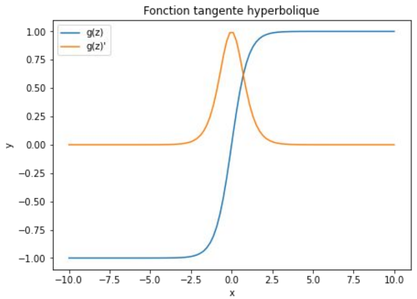
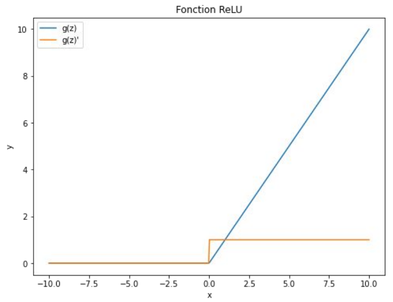


`torch.nn.functional.tanh(z)`



`torch.nn.functional.relu(z)`

Importance des fonctions d'activation

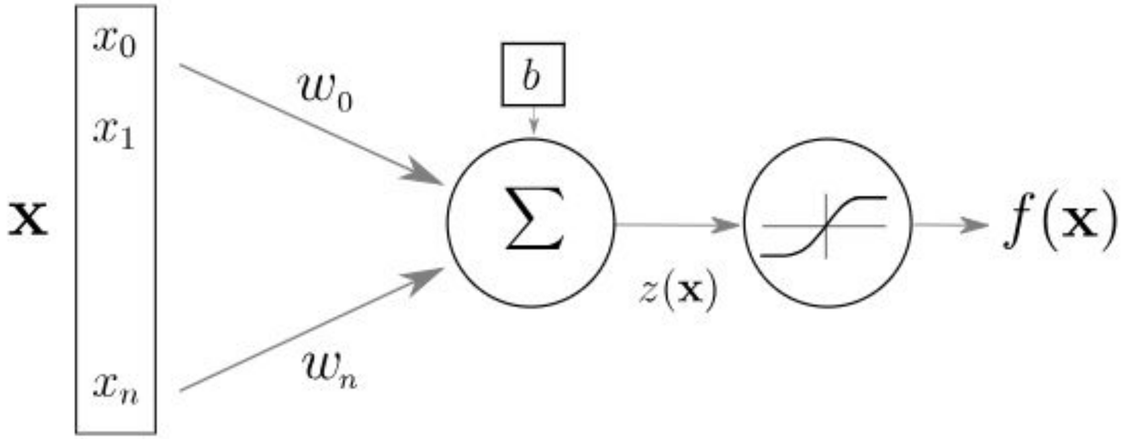
	Avantages	Inconvénients
	- Valeurs comprises entre 0 et 1, ce qui la rend adapté dans le cas de l'utilisation d'un neurone de sortie pour des tâches de classification binaire. - La sigmoïde est une fonction lisse et différentiable. Bonne propriété pour le processus d'optimisation.	- Calcul coûteux en raison de l'exponentiel. - Problème de saturation quand g(z) = 0 ou 1 car g'(z) = 0. Le gradient de la fonction de coût disparaît. C'est ce qu'on appelle le « Vanishing gradient ». - Fonction non centrée en zéro. Le gradient de tous les poids connectés au même neurone sera soit positif ou soit négatif.
	- Fonction centrée en zéros.	- Calcul coûteux en raison des exponentiels. - Problème de vanishing gradient. Quand la fonction de rapproche de -1 ou 1, la dérivée se rapproche de 0.
	- Calcul et optimisation plus rapide - Ne souffre pas du problème de vanishing gradient car non bornée du côté positive.	- Problème de « ReLU mourants » car les valeurs négatives restent toujours à zéro, ce qui entraîne un gradient nul pour les neurones concernés. - La fonction n'est pas différentiable en zéros.

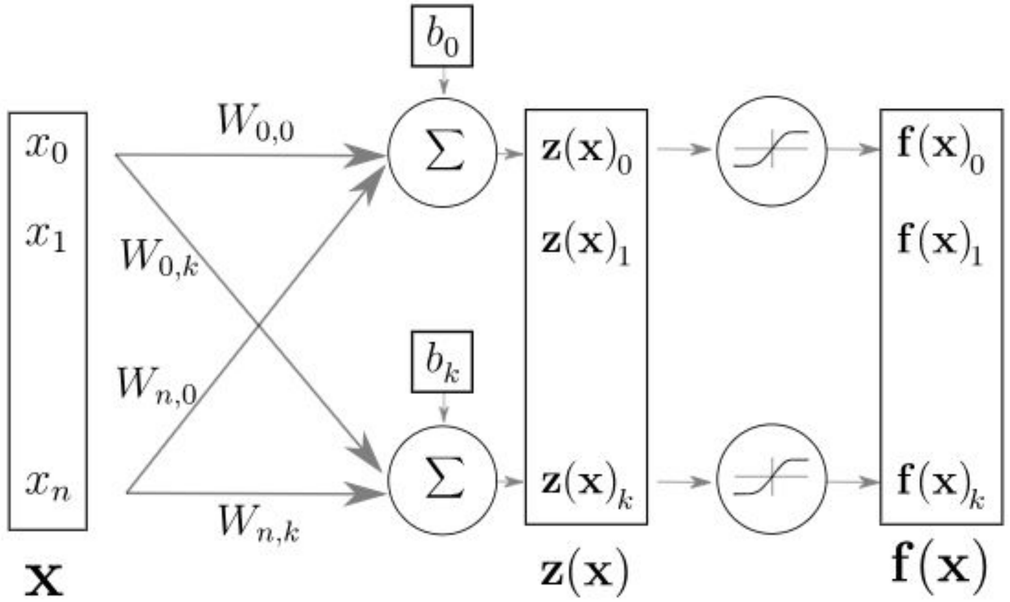
Perceptron avec PyTorch

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class Perceptron(nn.Module):
    def __init__(self, input_dim):
        super(Perceptron, self).__init__()

        # Définition d'une couche entièrement connectée
        # Cette couche est composé de input_dim neurones
        # en entrée et de un neurone en sortie
        self.fc = nn.Linear(input_dim, 1)

    def forward(self, x): # x correspond aux données en entrée
        # Propagation forward à travers la couche fc
        x = self.fc(x)
        # Application de la fonction d'activation ReLU
        # à la sortie de fc
        x = F.relu(x)
        return x
```

Comment ça marche ?

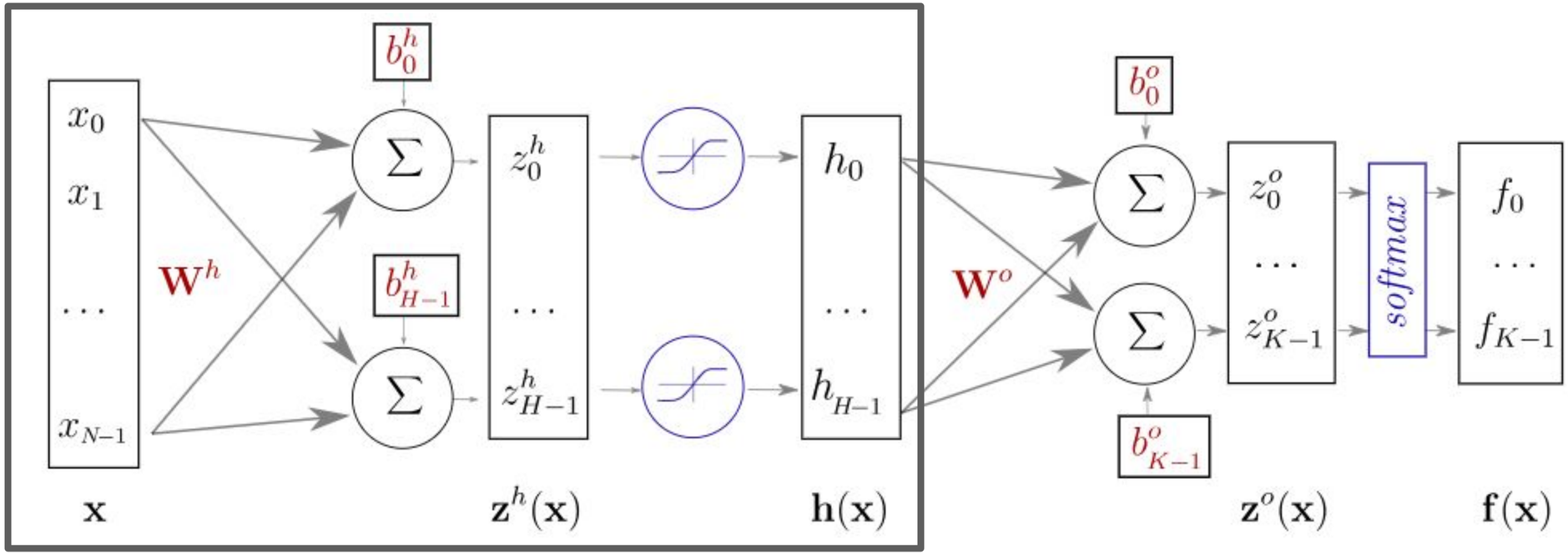
Réseau de neurones à 1 couche cachée





Comment ça marche ?

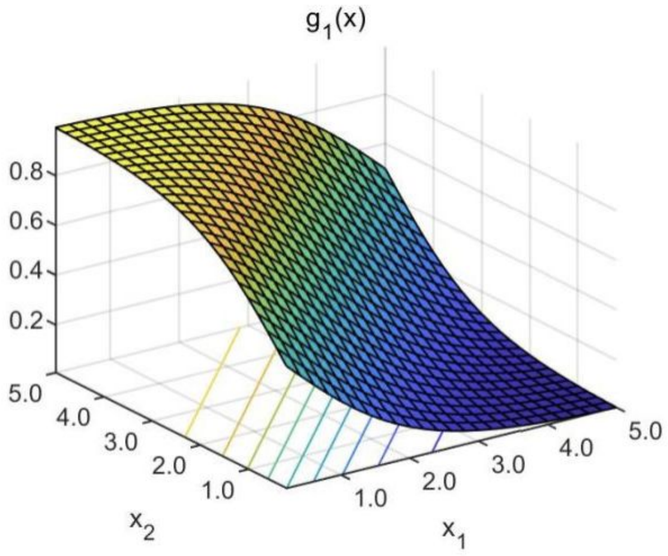
Réseau de neurones à 1 couche cachée

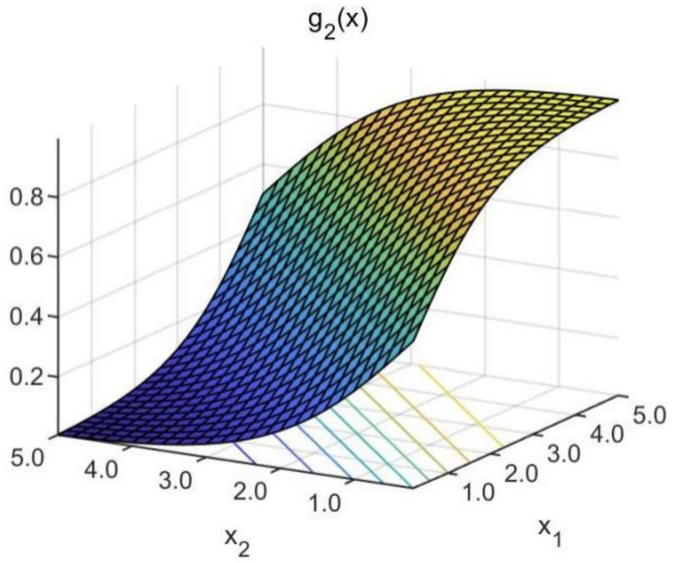


Comment ça marche ?

Fonction SOFTMAX

$$g_j(x) = \frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}}, \quad j = 1 : n$$



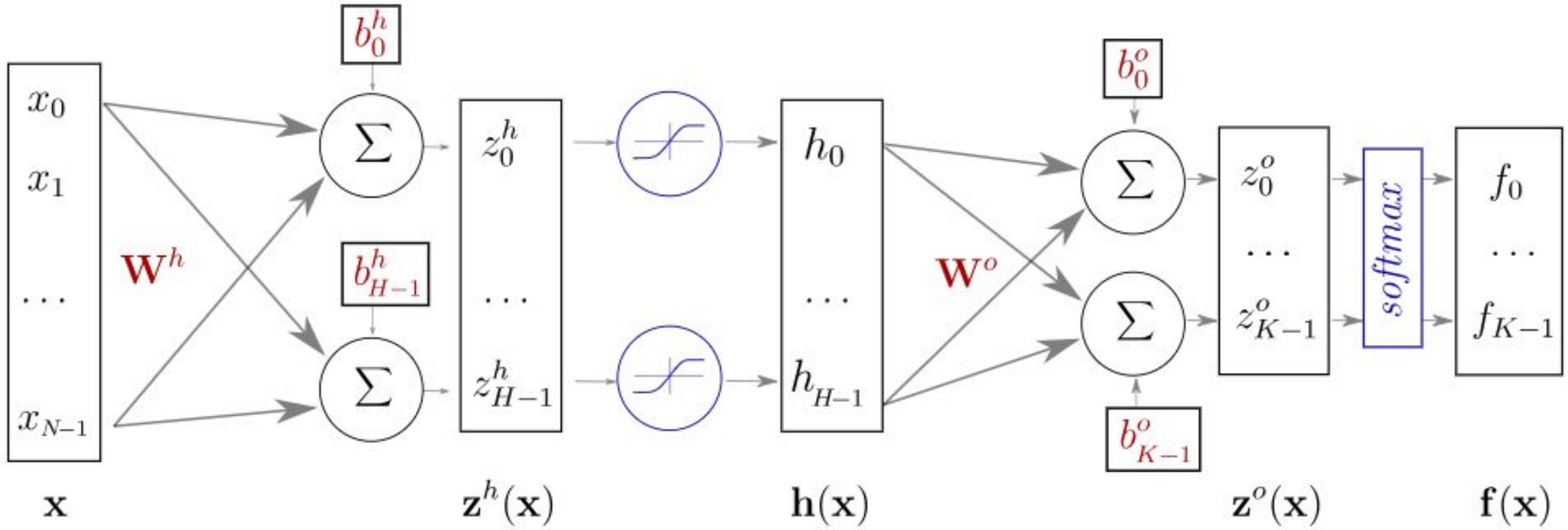


→ **Adapté à la classification multi-classe**

→ **Transforme la sortie du réseau en distribution de probabilité**

Comment ça marche ?

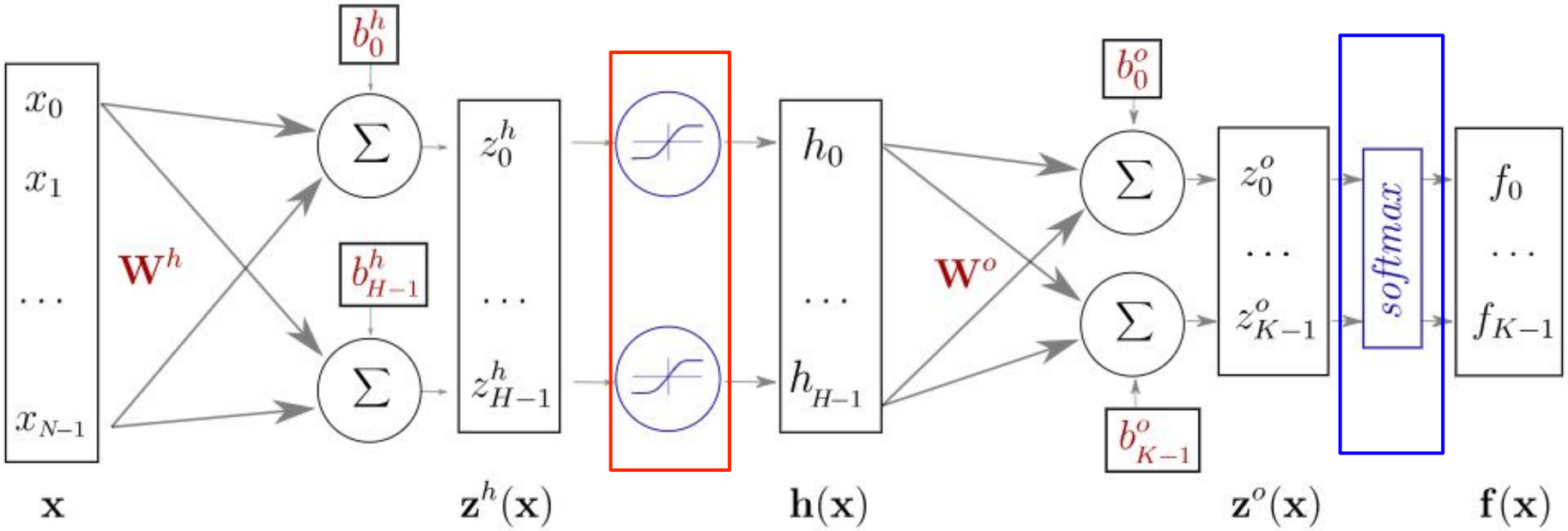
Réseau de neurones à 1 couche cachée



$$h(x) = g(W^h x + b^h)$$

Comment ça marche ?

Réseau de neurones à 1 couche cachée

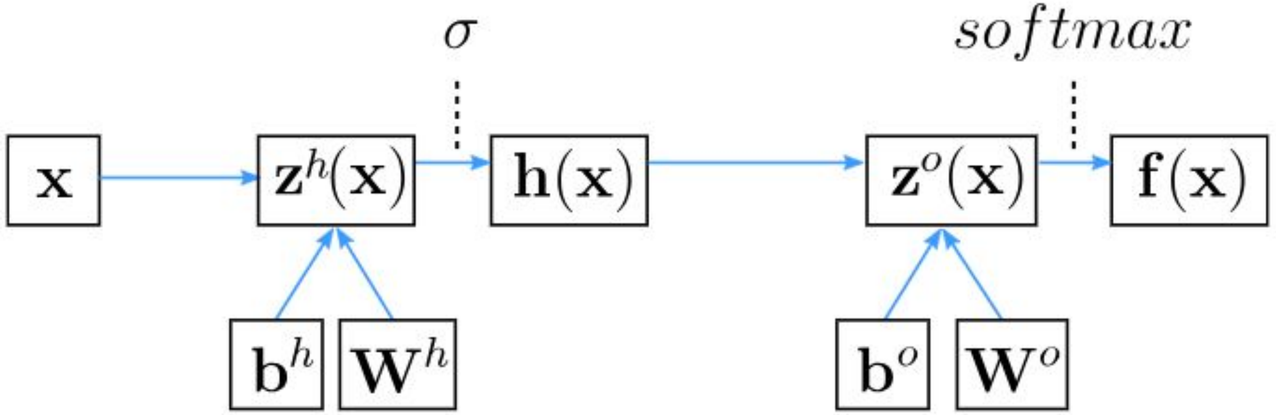


$$h(x) = g(W^h x + b^h)$$

$$f(x) = softmax(W^o h(x) + b^o)$$

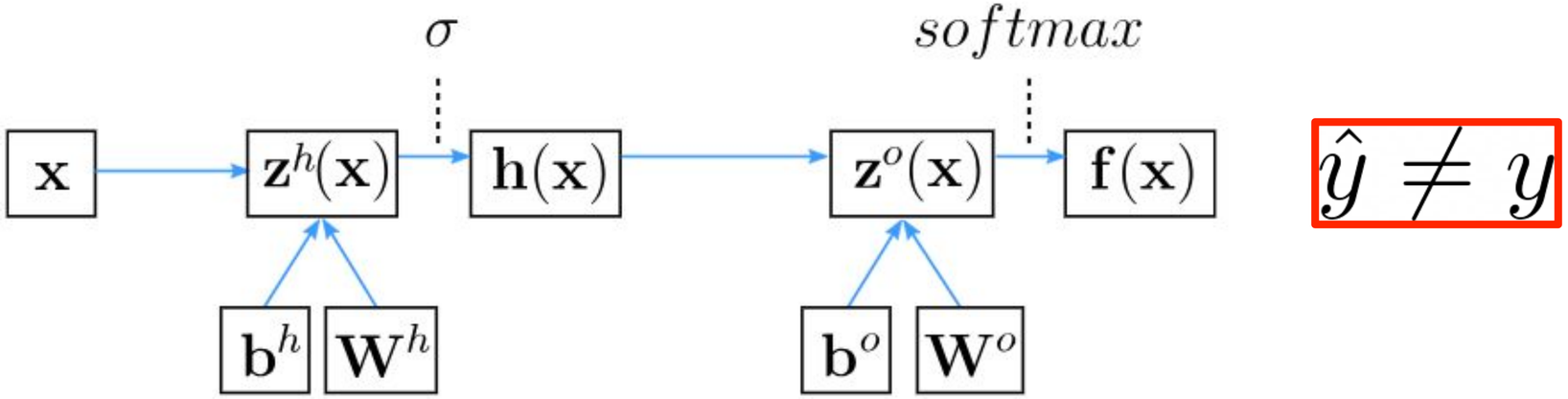
Comment ça marche ?

Réseau de neurones à 1 couche cachée



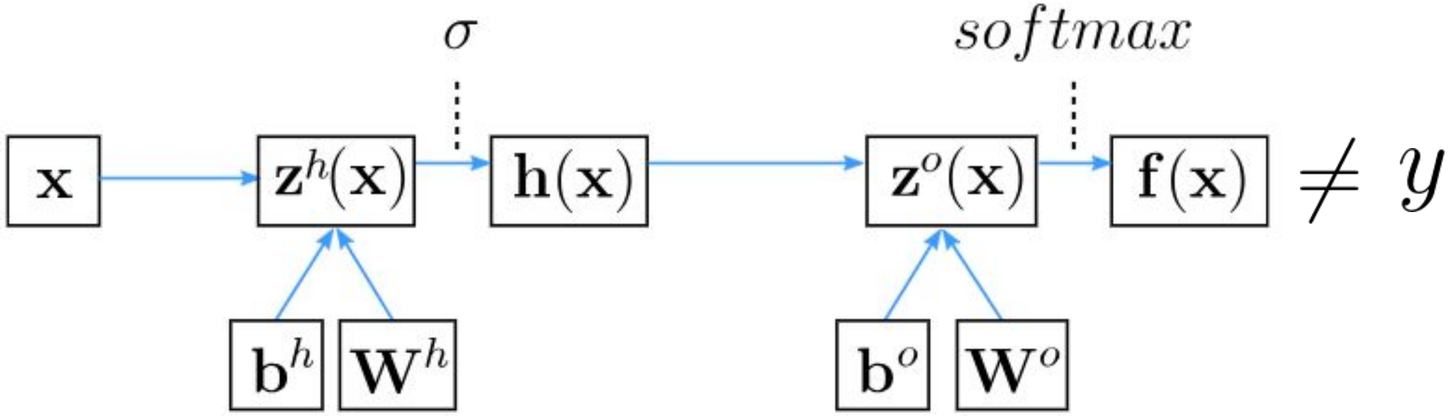
Comment ça marche ?

L’apprentissage: un problème d’optimisation



Comment ça marche ?

L’apprentissage: un problème d’optimisation



Objectif: Trouver les meilleurs paramètres $\theta = (W^h, b^h, W^o, b^o)$

pour minimiser une **fonction coût** l

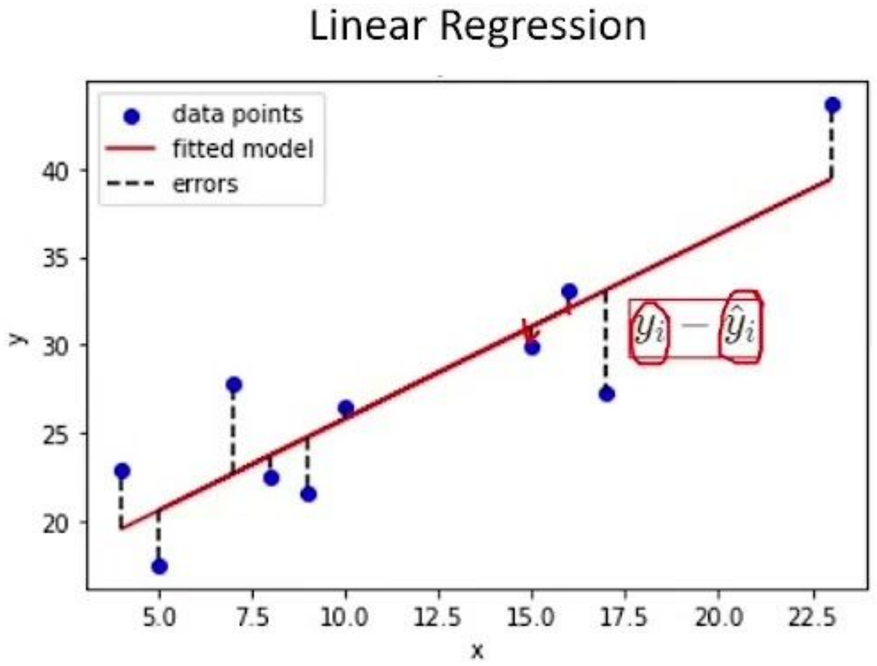
Fonctions de coût classiques

Fonctions de coût classiques

Mean Squared Error (MSE)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

→ **Régression**



Fonctions de coût classiques

Mean Squared Error (MSE)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

→ **Régression**

Categorical Cross-Entropy (CCE)

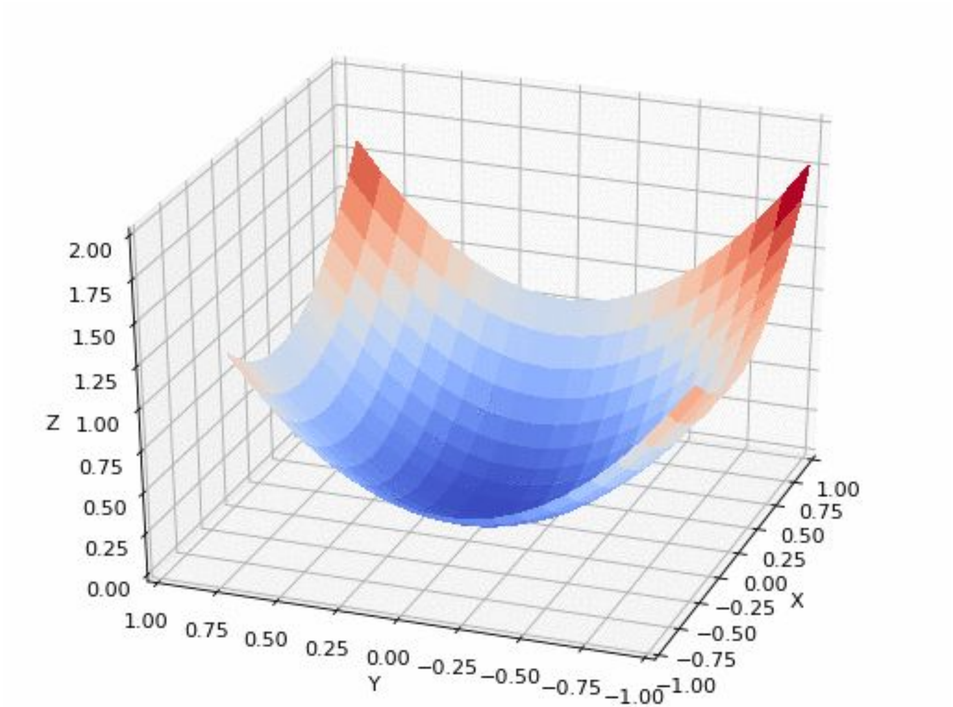
$$CCE = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{M} y_{ij}\,\log(\hat{y}_{ij})$$

→ **Classification**

Binary Cross-Entropy (BCE)

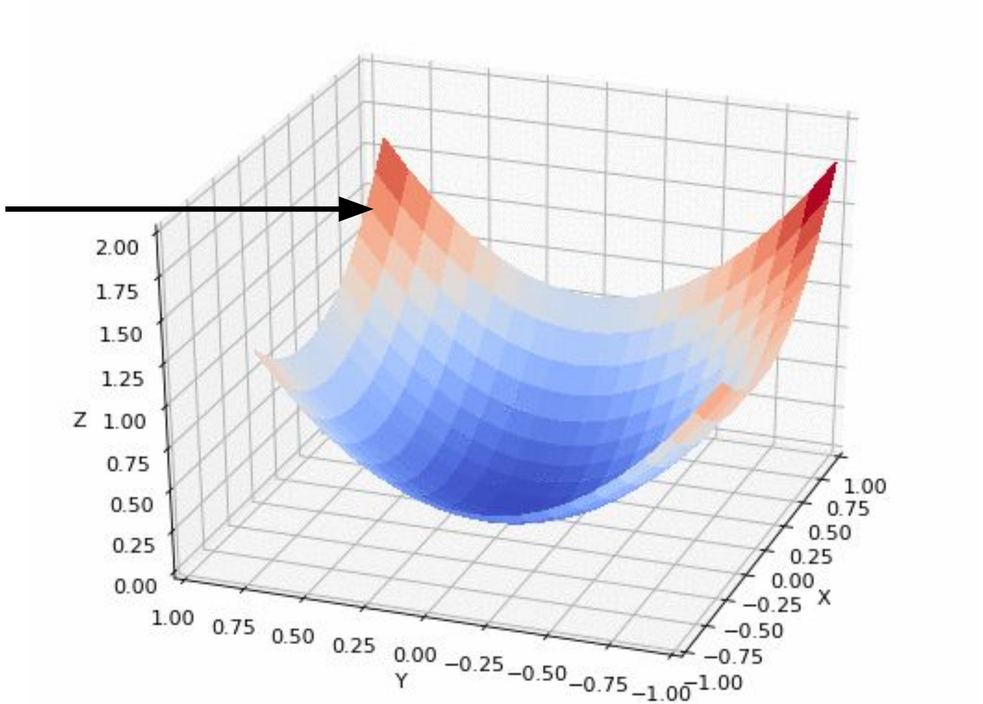
$$BCE = -\frac{1}{n}\sum_{i=1}^{n}[y_i\,\log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

Descente de Gradient



Descente de Gradient

Point aléatoirement déterminé



Initialisation aléatoire des paramètres

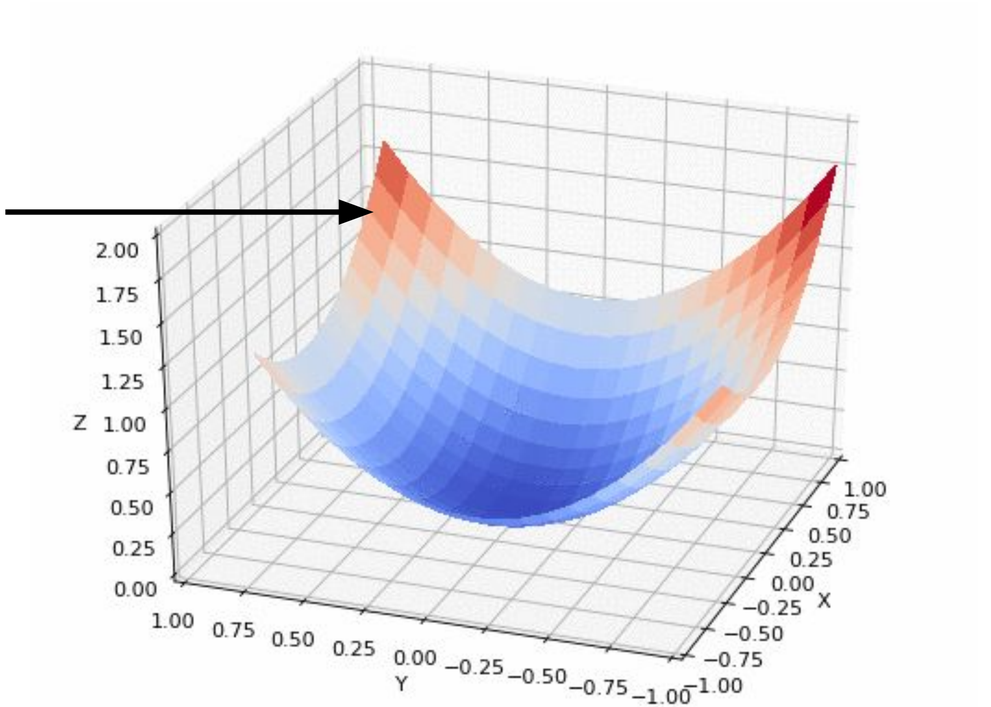
Calcul du gradient

Faire un petit pas dans la direction opposée au gradient

Continuer jusqu'à convergence

Descente de Gradient

Point aléatoirement déterminé



Initialisation aléatoire des paramètres

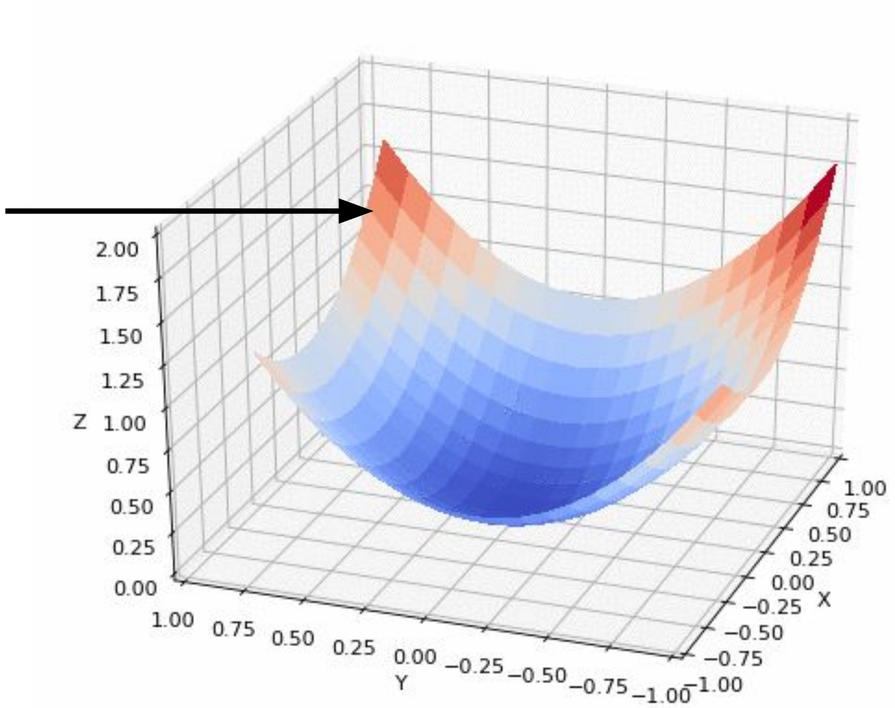
Calcul du gradient de la fonction coût $\delta = \nabla_\theta l(f(x), y)$

Faire un petit pas dans la direction opposée au gradient $\theta \leftarrow \theta - \eta\delta$

Continuer jusqu'à convergence

Descente de Gradient

Point aléatoirement déterminé



Initialisation aléatoire des paramètres

Calcul du gradient de la fonction coût $\delta = \nabla_\theta l(f(x), y)$

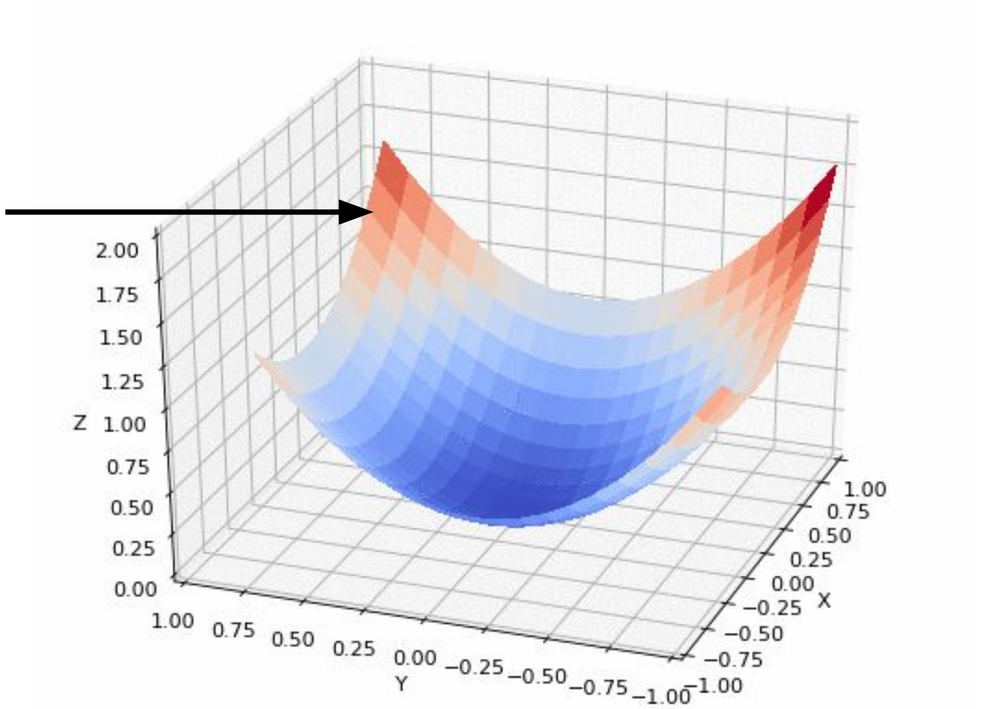
Faire un petit pas dans la direction opposée au gradient $\theta \leftarrow \theta - \eta\delta$

Continuer jusqu'à convergence

* Learning rate

Descente de Gradient

Point aléatoirement déterminé



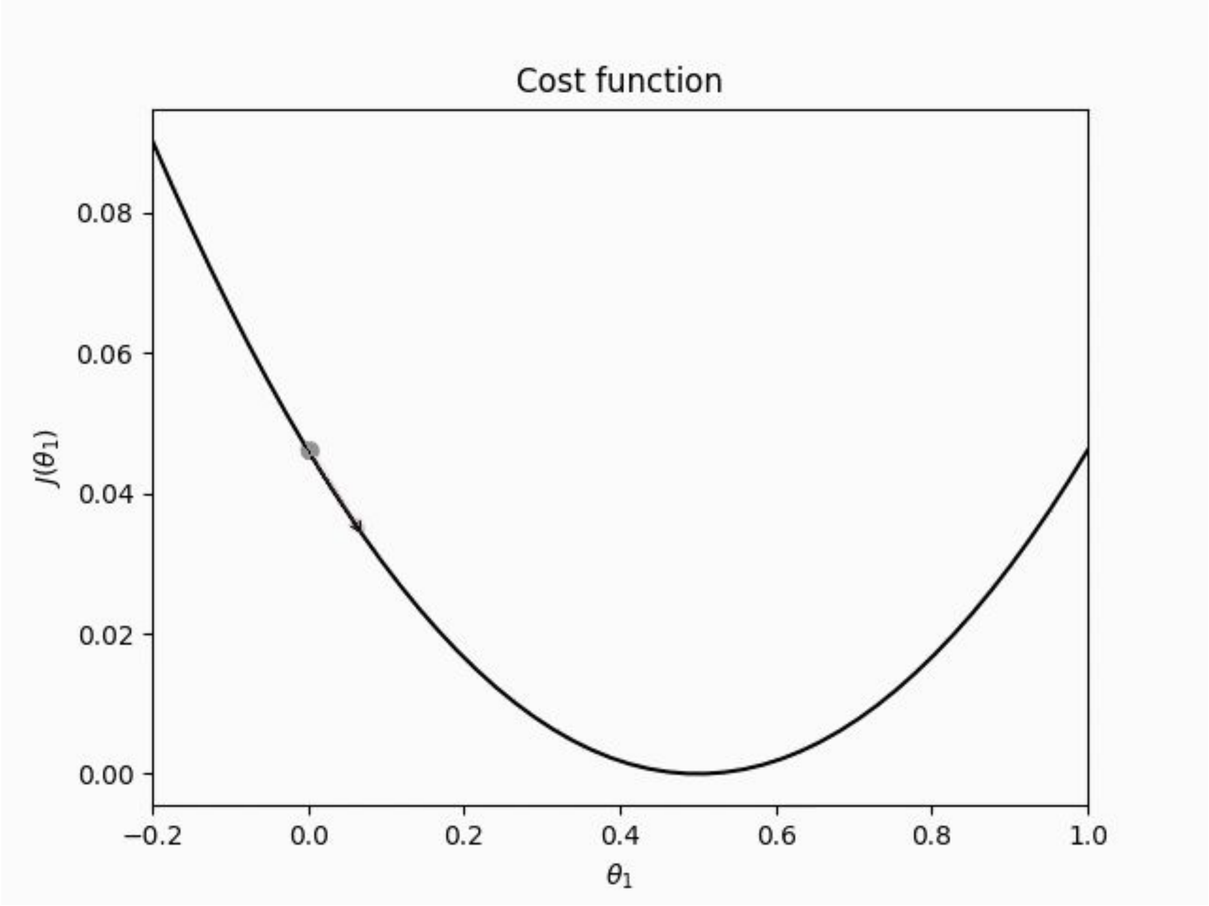
Initialisation aléatoire des paramètres

Calcul du gradient de la fonction coût $\delta = \nabla_\theta l(f(x), y)$

Faire un petit pas dans la direction opposée au gradient $\theta \leftarrow \theta - \eta\delta$

Continuer jusqu'à convergence

Descente de Gradient



Initialisation aléatoire des paramètres

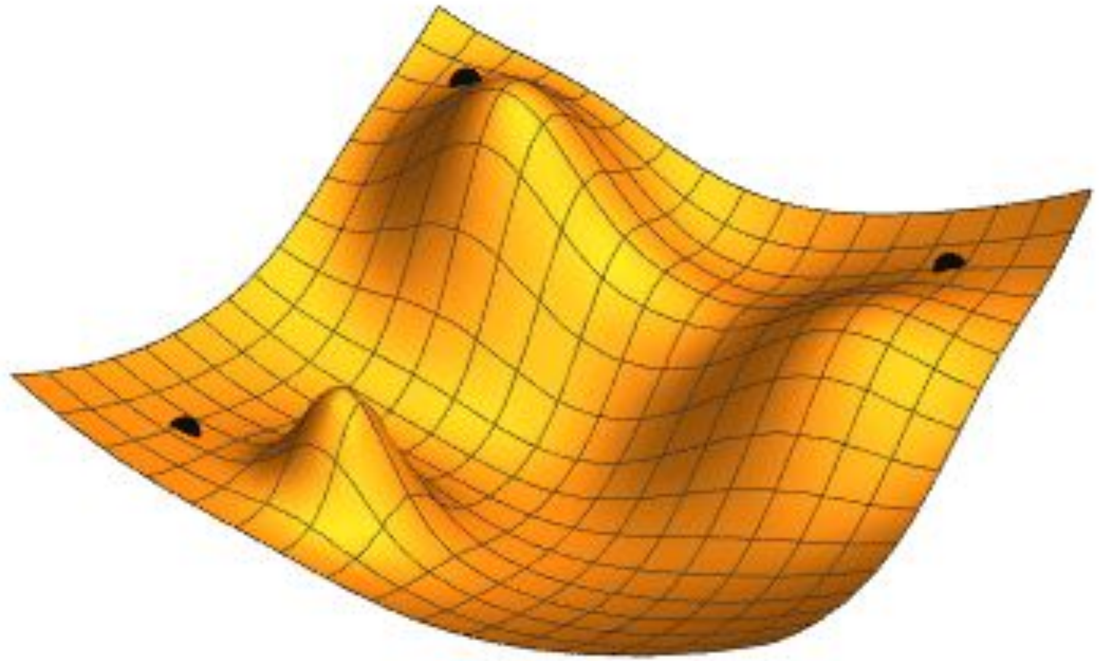
Calcul du gradient de la fonction coût $\delta = \nabla_\theta l(f(x), y)$

Faire un petit pas dans la direction opposée au gradient $\theta \leftarrow \theta - \eta\delta$

Continuer jusqu'à convergence

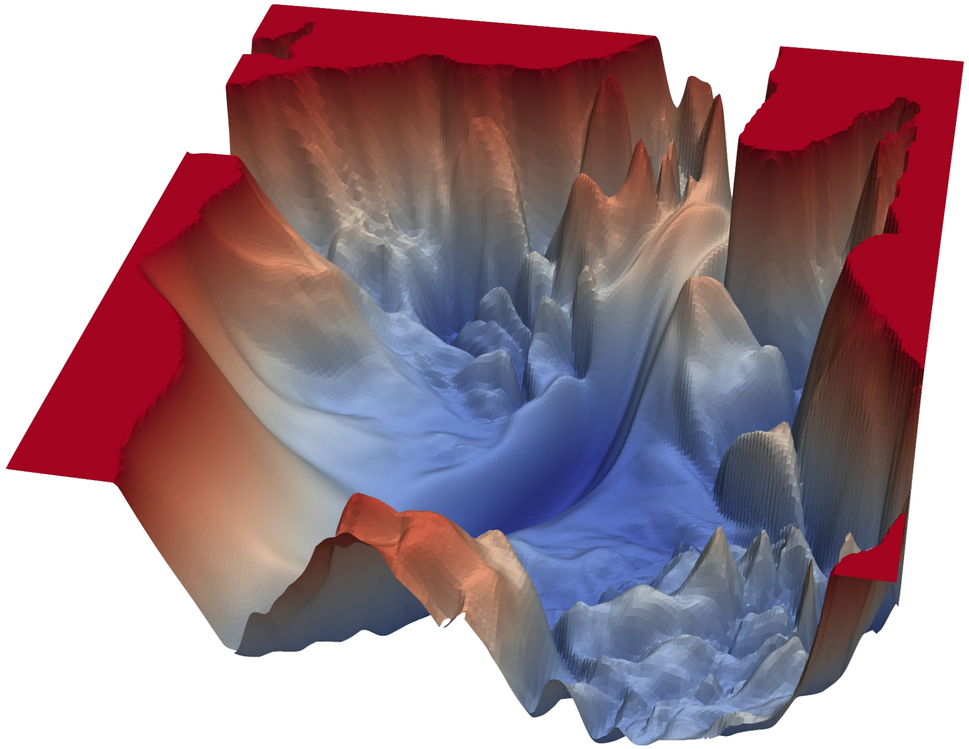
Défis en matière d’optimisation

Trouver le minimum global de la fonction de coût peut être difficile, car la descente de gradient peut se retrouver piégée dans des minima locaux.



Défis en matière d’optimisation

Exemple d’une représentation 3D des contours d’une fonction de coût relatif au réseau de neurones VGG16 sur l’ensemble de donnée Cifar-10.



Source : https://www.cs.umd.edu/~tomg/projects/landscapes/

Comment ça marche ?

Calculer le gradient: backpropagation

Problème: Comment calculer le gradient pour faire $\theta \leftarrow \theta - \eta\delta$?

Comment ça marche ?

Calculer le gradient: backpropagation

Problème: Comment calculer le gradient pour faire $\theta \leftarrow \theta - \eta\delta$?

- Utilisation de l'algorithme « **Backpropagation** » pour ajuster les paramètres des réseaux de neurones afin de minimiser l'erreur de prédiction.
- Cette méthode fonctionne en calculant les gradients par rapport à l'erreur à chaque couche du réseau, puis en propageant ces gradients de manière rétroactive pour mettre à jour tous les poids.

Comment ça marche ?

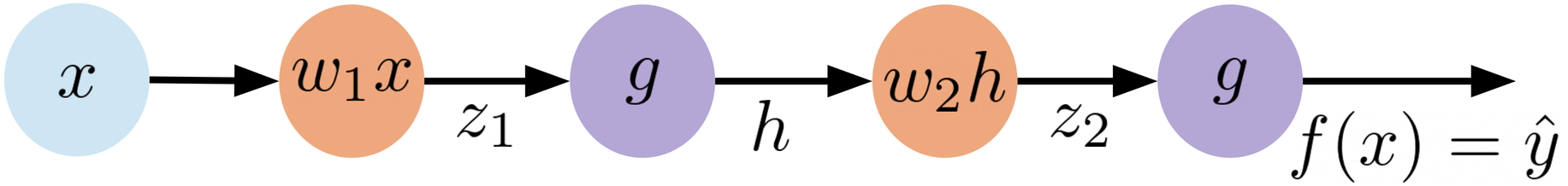
Calculer le gradient: backpropagation

<u>Problème:</u> Comment calculer le gradient pour faire $\theta \leftarrow \theta - \eta\delta$?

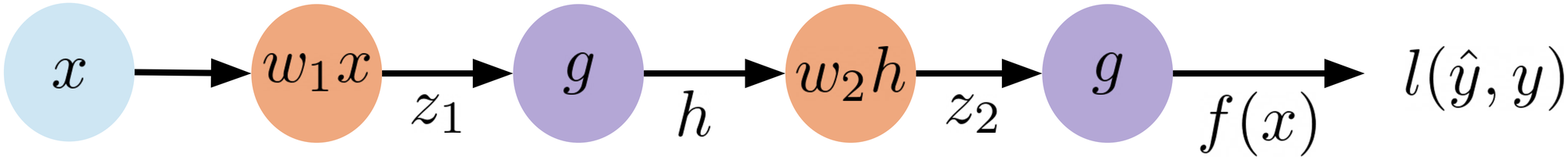
- Utilisation de l'algorithme « **Backpropagation** » pour ajuster les paramètres des réseaux de neurones afin de minimiser l'erreur de prédiction.
- Cette méthode fonctionne en calculant les gradients par rapport à l'erreur à chaque couche du réseau, puis en propageant ces gradients de manière rétroactive pour mettre à jour tous les poids.

Vidéo 3Blue1Brown: https://www.youtube.com/watch?v=Ilg3gGewQ5U

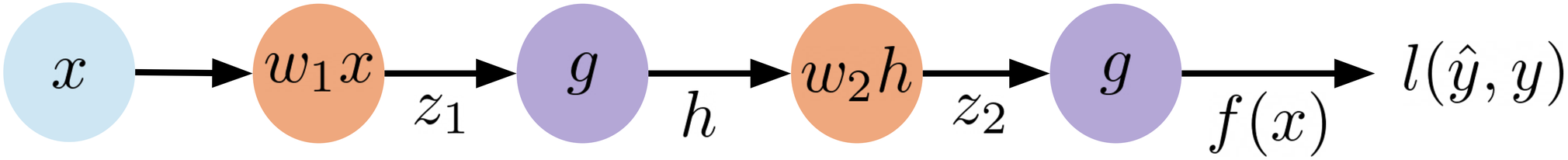
Backpropagation



Backpropagation



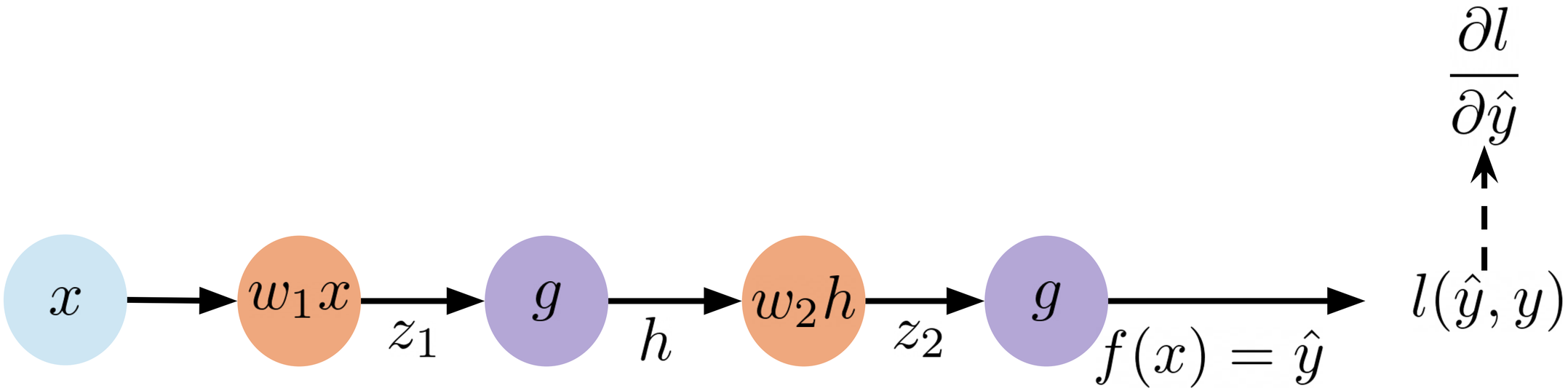
Backpropagation



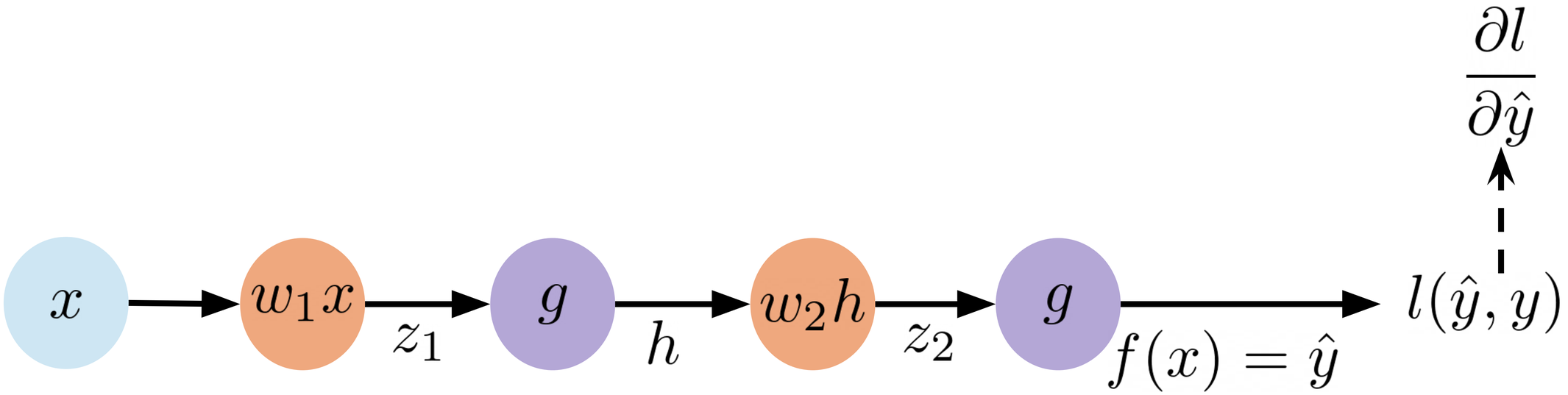
Question: Comment un petit changement des poids affecte la loss ?

$$\frac{\partial l}{\partial w_1} = ? \qquad \frac{\partial l}{\partial w_2} = ?$$

Backpropagation

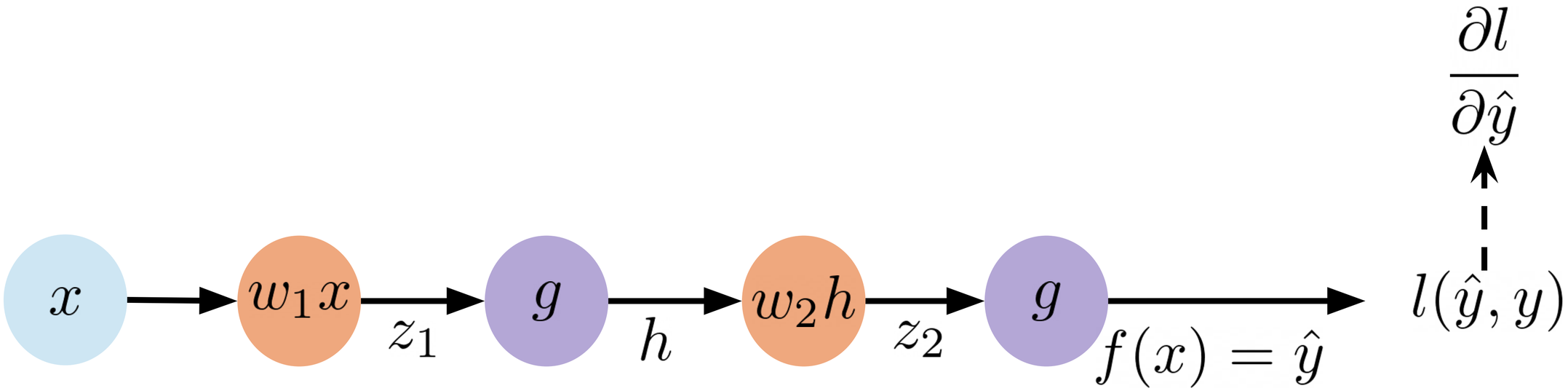


Backpropagation



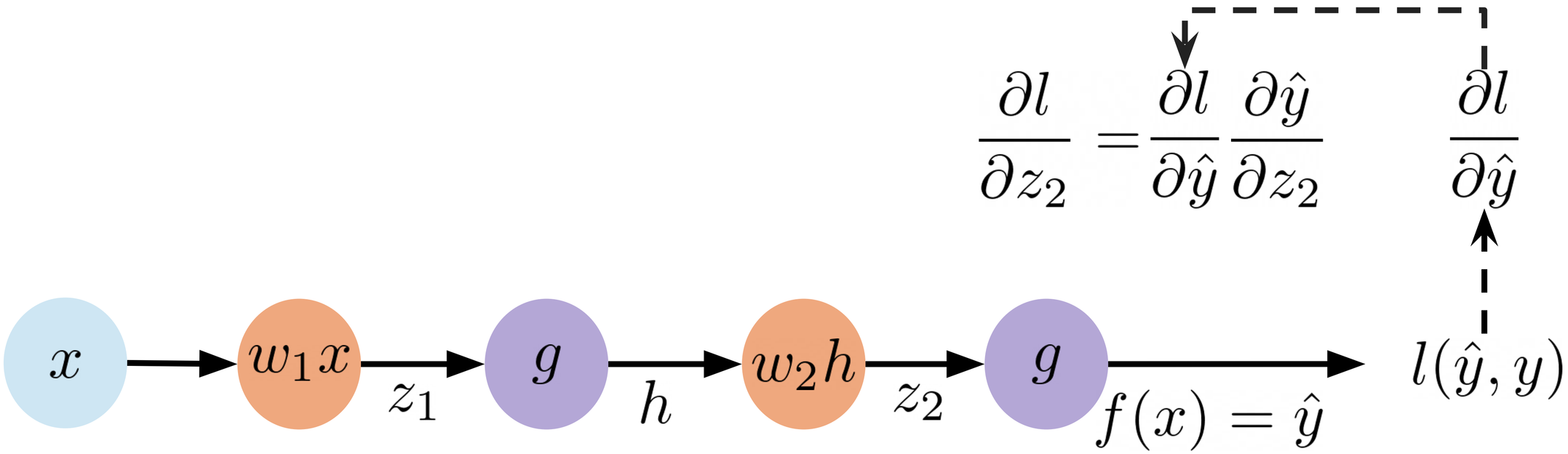
$$l(\hat{y}, y) = \|\hat{y} - y\|^2$$

Backpropagation



$$\hat{y} = g(z_2)$$

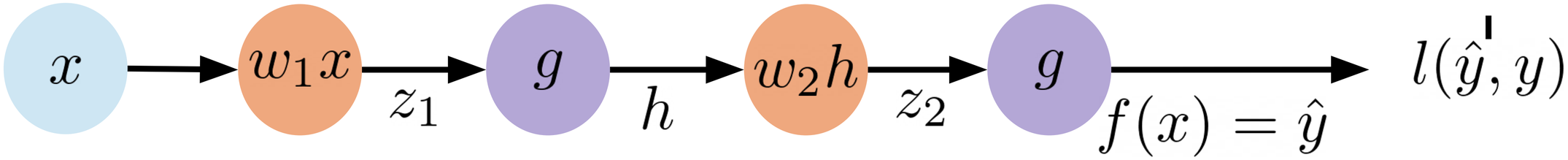
Backpropagation



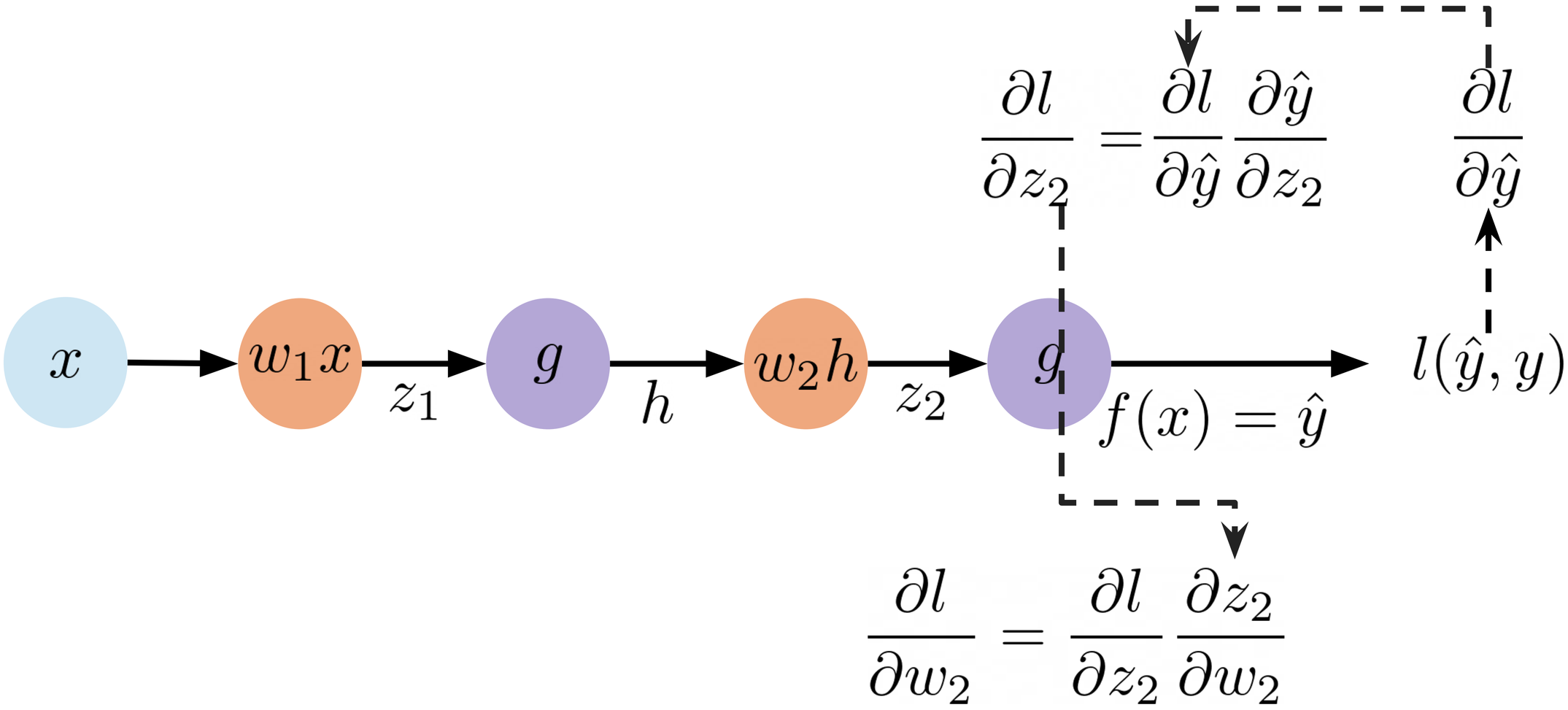
Backpropagation

Règle de la chaîne

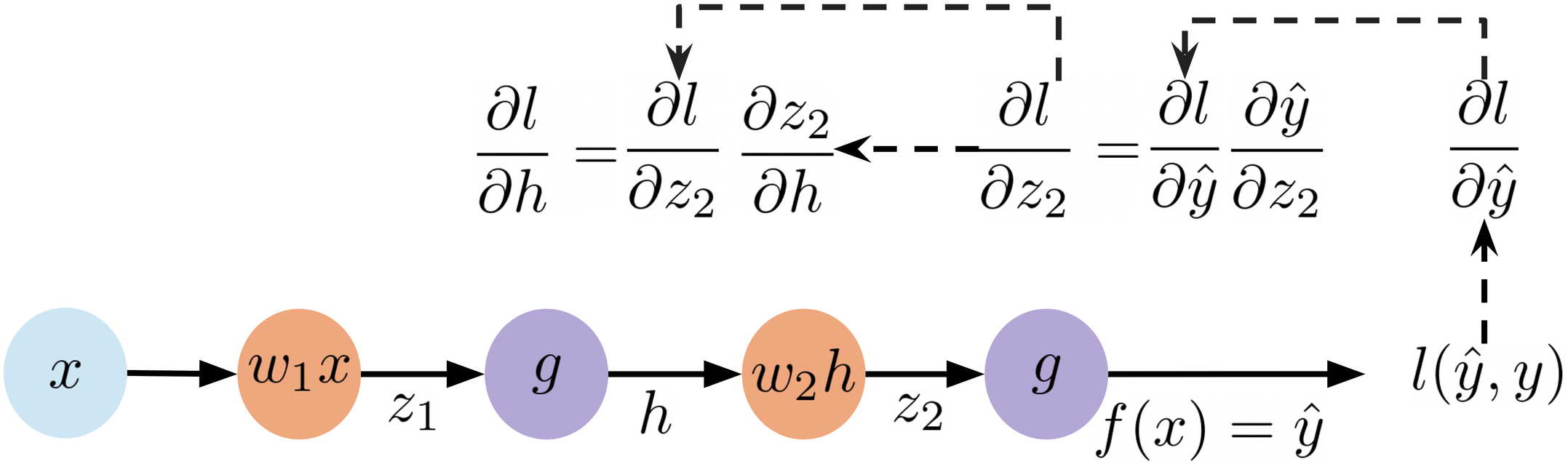
$$\frac{\partial l}{\partial z_2} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_2}$$



Backpropagation



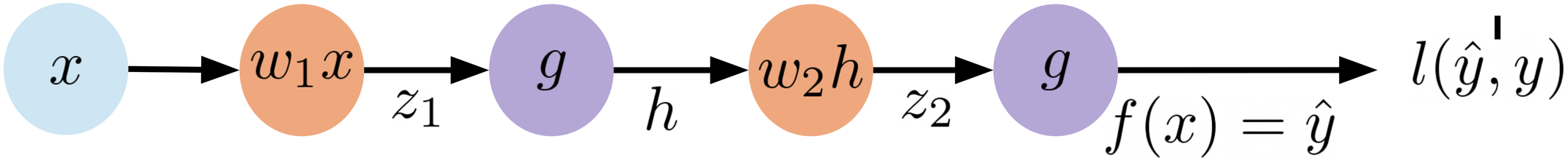
Backpropagation



$$\frac{\partial l}{\partial w_2} = \frac{\partial l}{\partial z_2}\frac{\partial z_2}{\partial w_2}$$

Backpropagation

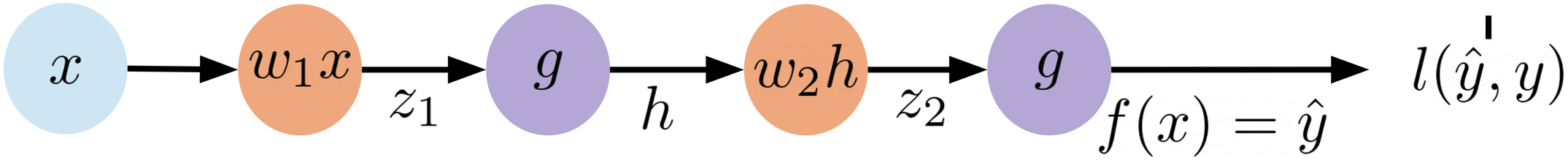
$$\frac{\partial l}{\partial z_1} = \frac{\partial l}{\partial h}\frac{\partial h}{\partial z_1} \qquad \frac{\partial l}{\partial h} = \frac{\partial l}{\partial z_2}\frac{\partial z_2}{\partial h} \qquad \frac{\partial l}{\partial z_2} = \frac{\partial l}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_2} \qquad \frac{\partial l}{\partial \hat{y}}$$



$$\frac{\partial l}{\partial w_2} = \frac{\partial l}{\partial z_2}\frac{\partial z_2}{\partial w_2}$$

Backpropagation

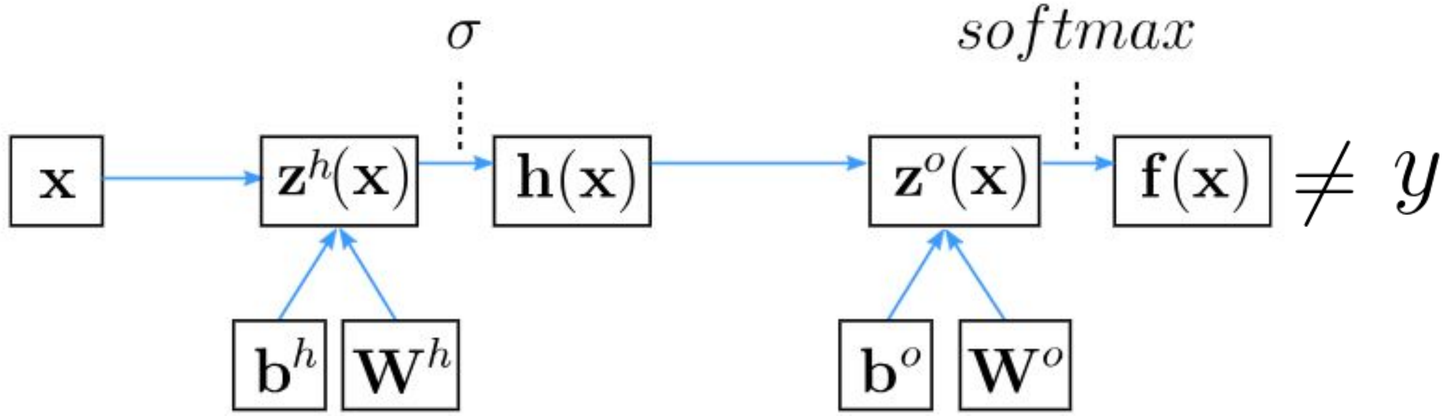
$$\frac{\partial l}{\partial z_1} = \frac{\partial l}{\partial h}\frac{\partial h}{\partial z_1} \qquad \frac{\partial l}{\partial h} = \frac{\partial l}{\partial z_2}\frac{\partial z_2}{\partial h} \qquad \frac{\partial l}{\partial z_2} = \frac{\partial l}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z_2} \qquad \frac{\partial l}{\partial \hat{y}}$$



$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial z_1}\frac{\partial z_1}{\partial w_1}$$

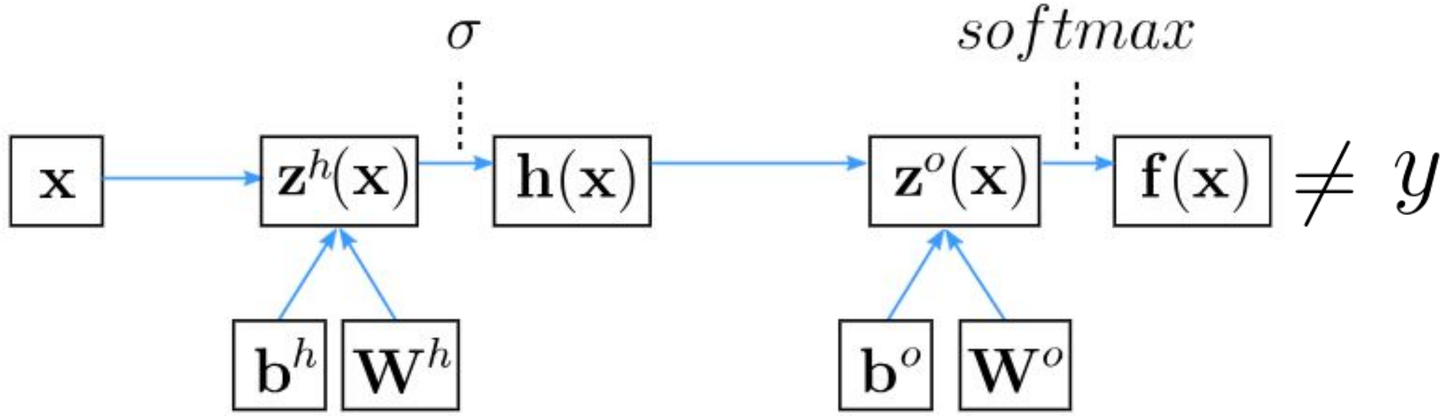
$$\frac{\partial l}{\partial w_2} = \frac{\partial l}{\partial z_2}\frac{\partial z_2}{\partial w_2}$$

Backpropagation



$$\nabla l = \left[\frac{\partial l}{\partial W^o}, \frac{\partial l}{\partial b^o}, \frac{\partial l}{\partial W^h}, \frac{\partial l}{\partial b^h} \right]$$

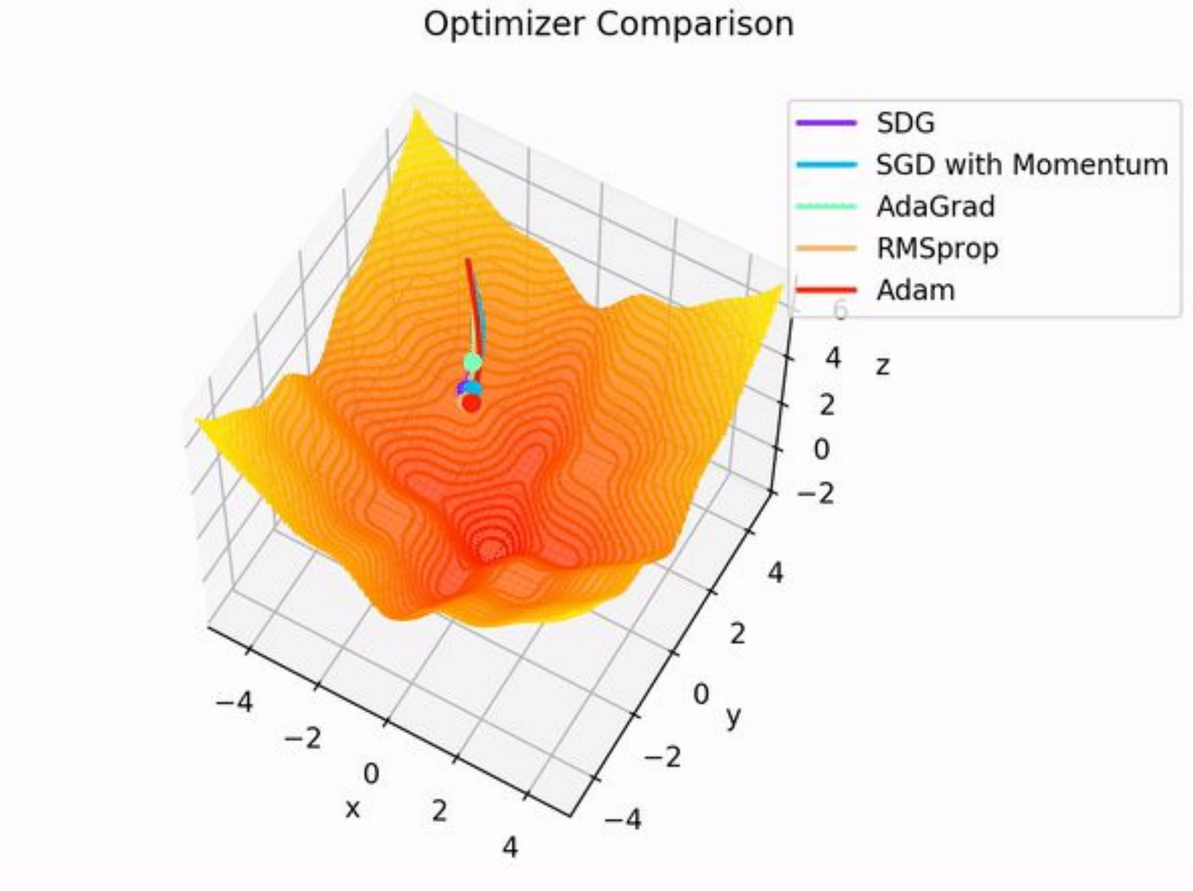
Backpropagation



$$\theta \leftarrow \theta - \eta\delta \qquad \delta = \nabla_\theta l(f(x), y) \qquad \nabla l = \left[\frac{\partial l}{\partial W^o}, \frac{\partial l}{\partial b^o}, \frac{\partial l}{\partial W^h}, \frac{\partial l}{\partial b^h}\right]$$

Moyenne sur les données d'entraînement

Optimizers


Optimizer Comparison
SDG
SGD with Momentum
AdaGrad
RMSprop
Adam
z
x
y

Optimizers

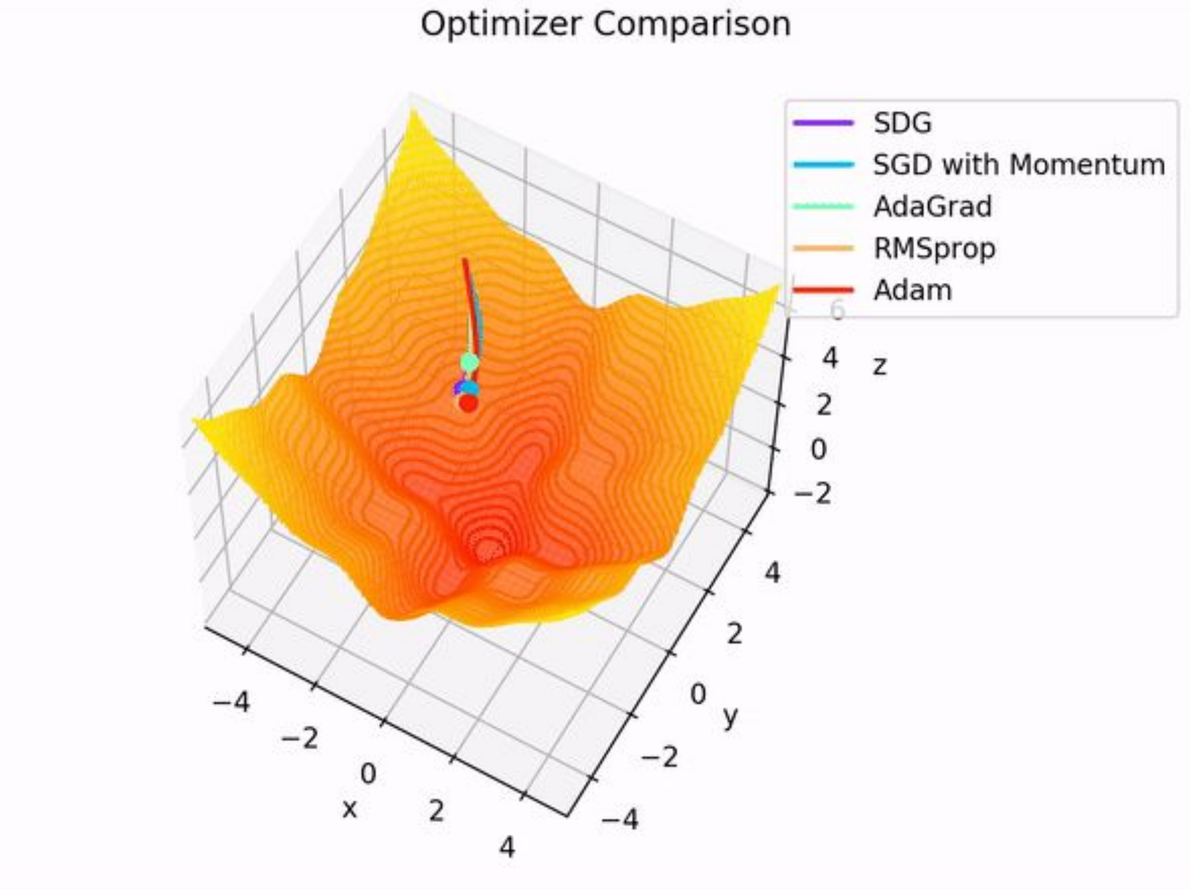
Stochastic Gradient Descent (SGD)

SGD + Moment

AdaGrad

RMSprop

Adam



Optimizers

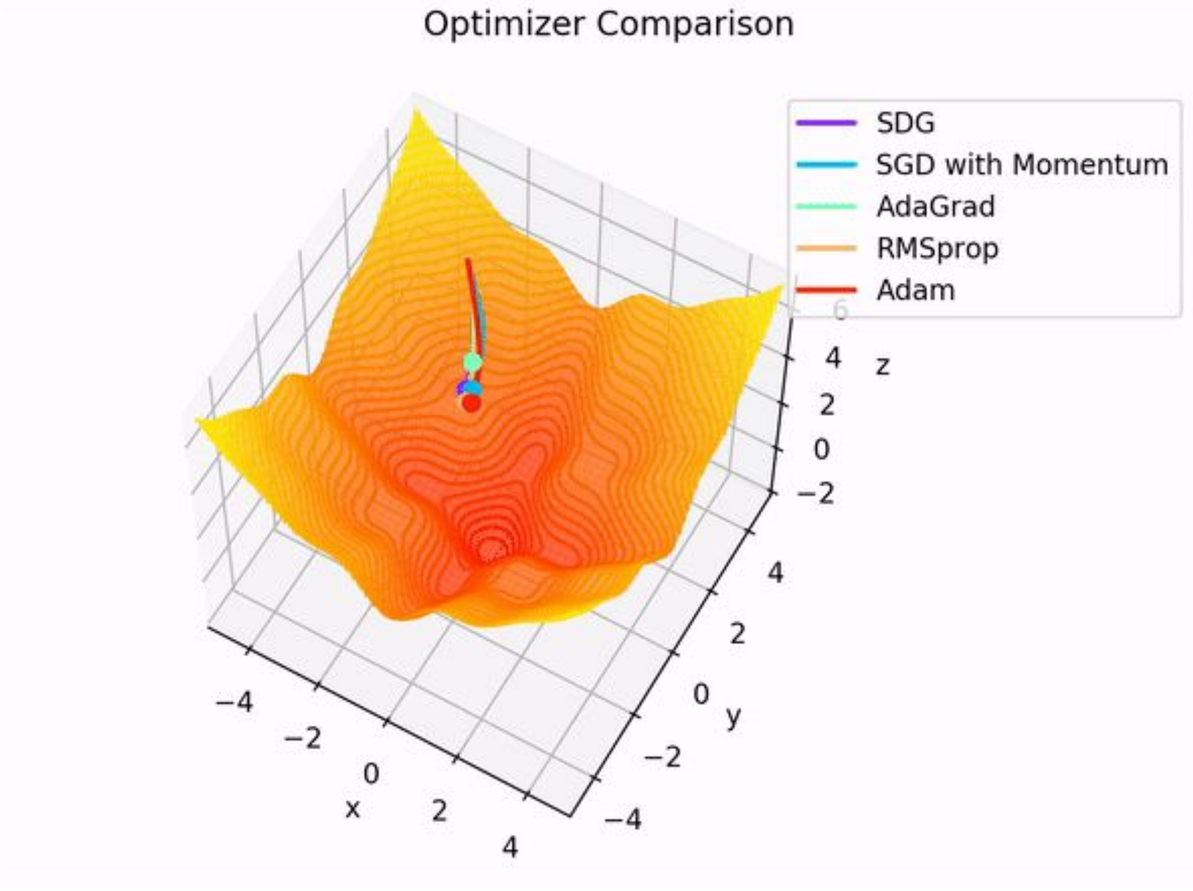
Stochastic Gradient Descent (SGD)

SGD + Moment

AdaGrad

RMSprop

Adam



Récap

Recette de l’entraînement d’un modèle

1. Préparer la donnée → Dataloader
1. Définir / construire le modèle → ??
1. Définir un objectif → ??
1. Optimiser les paramètres → ??
1. Evaluer

Récap

Recette de l’entraînement d’un modèle

1. Préparer la donnée → Dataloader
1. Définir / construire le modèle → Réseaux de neurones
1. Définir un objectif → ??
1. Optimiser les paramètres → ??
1. Evaluer

Récap

Recette de l’entraînement d’un modèle

1. Préparer la donnée → Dataloader
1. Définir / construire le modèle → Réseaux de neurones
1. Définir un objectif → Fonction coût
1. Optimiser les paramètres → ??
1. Evaluer

Récap

Recette de l’entraînement d’un modèle

1. Préparer la donnée → Dataloader
1. Définir / construire le modèle → Réseaux de neurones
1. Définir un objectif → Fonction coût
1. Optimiser les paramètres → Boucle d’entraînement
1. Evaluer

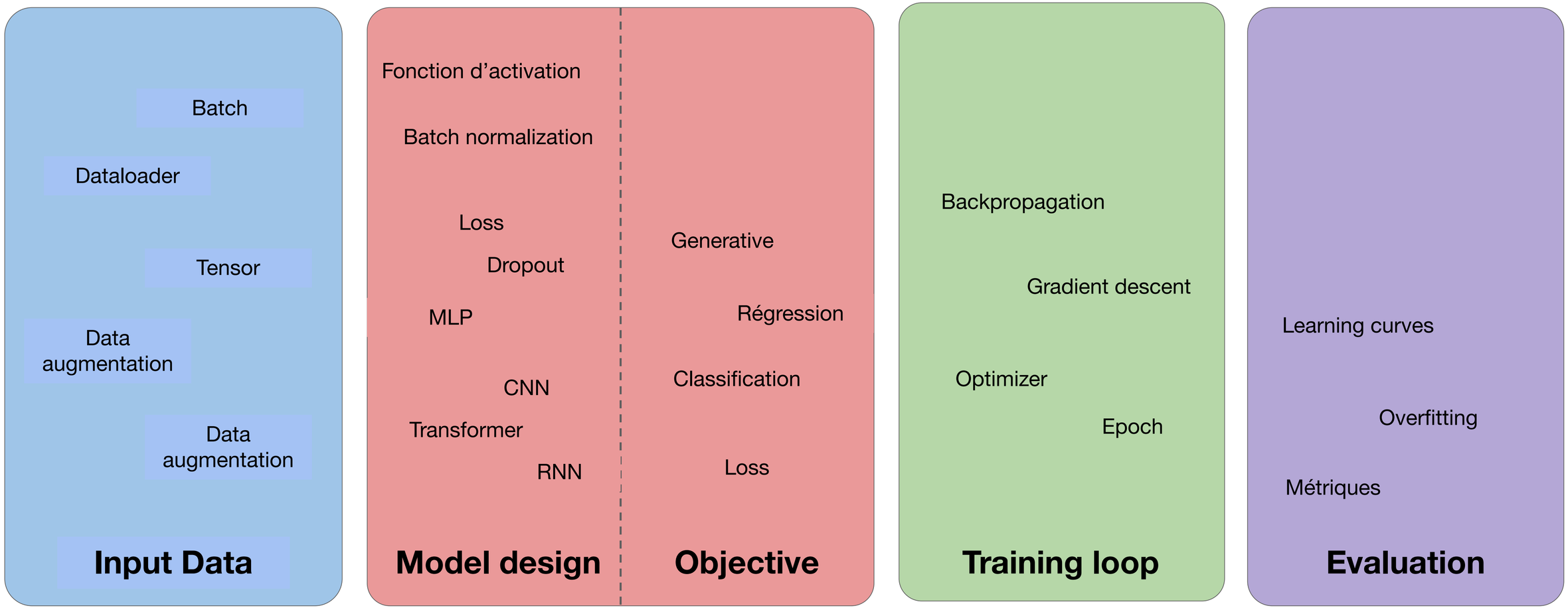
Récap

Recette de l’entraînement d’un modèle

1. Préparer la donnée → Dataloader
2. Définir / construire le modèle → Réseaux de neurones
3. Définir un objectif → Fonction coût
4. Optimiser les paramètres → Boucle d’entraînement
5. Evaluer

https://pytorch.org/docs/stable/

Amélioration continue



Récap

Recette de l’entraînement d’un modèle

1. Préparer la donnée → Dataloader
2. Définir / construire le modèle → Réseaux de neurones
3. Définir un objectif → Fonction coût
4. Optimiser les paramètres → Boucle d’entraînement → Hyperparamètres !
5. Evaluer

https://pytorch.org/docs/stable/

Nombre d’Epoch

Nombre d’Epoch

- Une **epoch** correspond à un passage complet du jeu de données d’entraînement dans le réseau de neurones.

Nombre d’Epoch

- Une **epoch** correspond à un passage complet du jeu de données d’entraînement dans le réseau de neurones.
- Le nombre d’epoch est un hyperparamètre important qui détermine le nombre de fois que le réseau de neurone sera entraîné à partir du jeu de données.

Nombre d’Epoch

- Une **epoch** correspond à un passage complet du jeu de données d’entraînement dans le réseau de neurones.
- Le nombre d’epoch est un hyperparamètre important qui détermine le nombre de fois que le réseau de neurone sera entraîné à partir du jeu de données.
- Un entraînement avec un nombre d’epoch plus élevé peut conduire à de meilleur performance, mais le risque de sur-apprentissage (overfitting) augmente aussi.

Nombre d’Epoch

- Une **epoch** correspond à un passage complet du jeu de données d’entraînement dans le réseau de neurones.
- Le nombre d’epoch est un hyperparamètre important qui détermine le nombre de fois que le réseau de neurone sera entraîné à partir du jeu de données.
- Un entraînement avec un nombre d’epoch plus élevé peut conduire à de meilleur performance, mais le risque de sur-apprentissage (overfitting) augmente aussi.
- Cet hyperparamètre est généralement fixé en fonction de l’expérimentation et des performances sur le jeu de données de validation. L’idée est de continuer à entraîner le réseau jusqu'à que les performances sur les données de validation commencent à se dégrader.

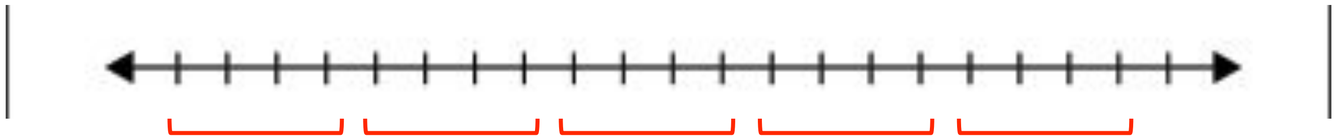
Batch size

Batch size

- En deep learning, les jeux de données sont souvent assez volumineux et il n’est pas efficace d’introduire tout l’ensemble de données au réseau de neurones en une seule fois. Il est plus approprié d’utiliser des lots de données plus petit.

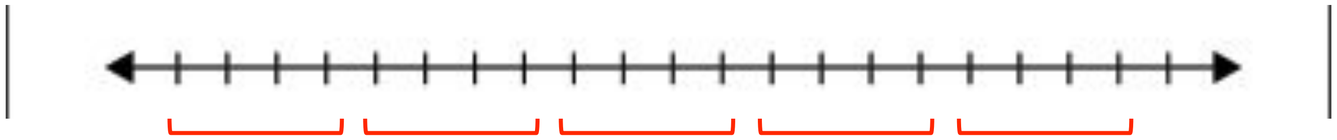
Batch size

- En deep learning, les jeux de données sont souvent assez volumineux et il n’est pas efficace d’introduire tout l’ensemble de données au réseau de neurones en une seule fois. Il est plus approprié d’utiliser des lots de données plus petit.



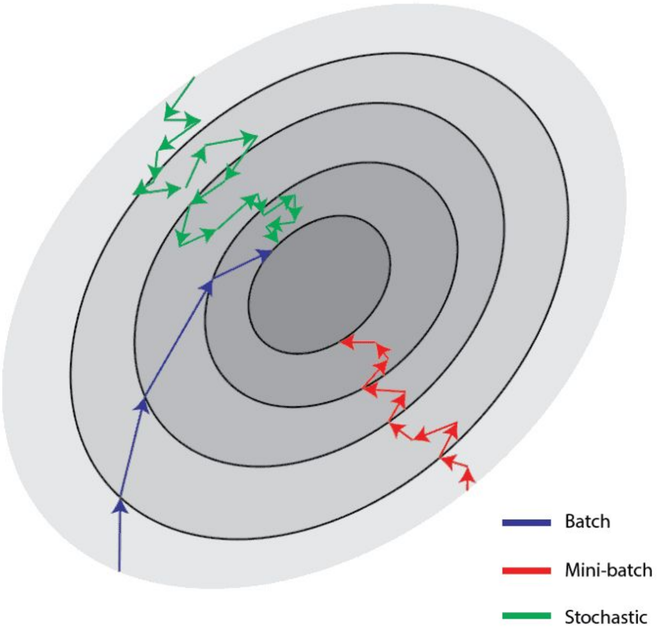
Batch size

- En deep learning, les jeux de données sont souvent assez volumineux et il n'est pas efficace d'introduire tout l'ensemble de données au réseau de neurones en une seule fois. Il est plus approprié d'utiliser des lots de données plus petit.



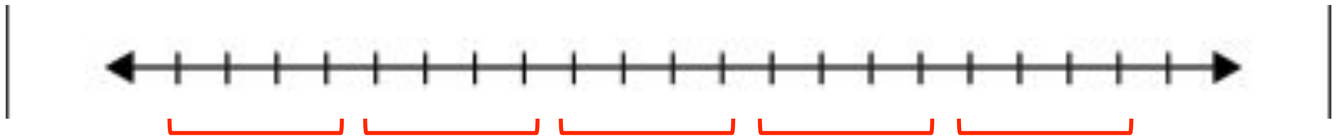
Batch size ↗ → Précision ↗ Vitesse ↘

Batch size ↘ → Précision ↘ Vitesse ↗



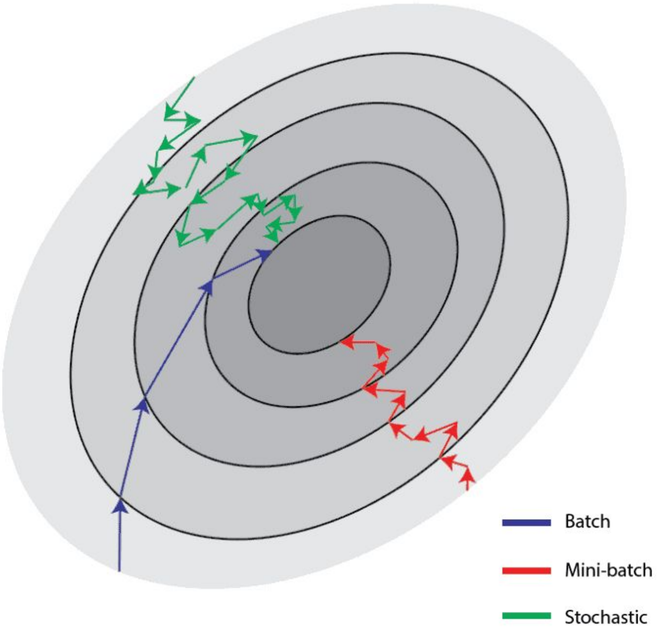
Batch size

- En deep learning, les jeux de données sont souvent assez volumineux et il n'est pas efficace d'introduire tout l'ensemble de données au réseau de neurones en une seule fois. Il est plus approprié d'utiliser des lots de données plus petit.



Batch size ↗ Précision ↗ Vitesse ↘

Batch size ↘ Précision ↘ Vitesse ↗



- Le choix de la taille de batch est un hyperparamètre. Il est recommandé d'utiliser des valeurs qui sont des puissances de 2.

Récap

Recette de l’entraînement d’un modèle

1. **Préparer la donnée** → Dataloader
2. Définir / construire le modèle → Réseaux de neurones
3. Définir un objectif → Fonction coût
4. Optimiser les paramètres → Boucle d’entraînement
5. Evaluer

https://pytorch.org/docs/stable/

Deep learning avec Pytorch

1. Préparer la donnée

Format *Tensor* de Pytorch → tout est “tensorisé”

Deep learning avec Pytorch

1. Préparer la donnée

Format *Tensor* de Pytorch → tout est “tensorisé”

- Données = tenseurs (généralise les matrices à 3D, 4D, …)

Deep learning avec Pytorch

1. Préparer la donnée

Format *Tensor* de Pytorch → tout est “tensorisé”

- Données = tenseurs (généralise les matrices à 3D, 4D, …)
- tenseurs = “faciles” pour paralléliser les calculs (CUDA)

Deep learning avec Pytorch

1. Préparer la donnée

Format *Tensor* de Pytorch → tout est “tensorisé”

- Données = tenseurs (généralise les matrices à 3D, 4D, …)
- tenseurs = “faciles” pour paralléliser les calculs (CUDA)
- Autograd pour tracker le gradient de la loss

Deep learning avec Pytorch

1. Préparer la donnée

```
from torch.utils.data import DataLoader, TensorDataset

X_train_tensor, y_train_tensor = ...
X_test_tensor, y_test_tensor = ...

# Create TensorDataset and DataLoader
train_dataset = TensorDataset( *tensors: X_train_tensor, y_train_tensor)
test_dataset = TensorDataset( *tensors: X_test_tensor, y_test_tensor)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

Deep learning avec Pytorch

1. Préparer la donnée

```python
from torch.utils.data import DataLoader, TensorDataset

X_train_tensor, y_train_tensor = ...
X_test_tensor, y_test_tensor = ...

# Create TensorDataset and DataLoader
train_dataset = TensorDataset( *tensors: X_train_tensor, y_train_tensor)
test_dataset = TensorDataset( *tensors: X_test_tensor, y_test_tensor)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

import module

données au format *tensor*

Définir l'objet Dataset

Dataloader

Récap

Recette de l’entraînement d’un modèle

1. Préparer la donnée → Dataloader
2. **Définir / construire le modèle** → Réseaux de neurones
3. Définir un objectif → Fonction coût
4. Optimiser les paramètres → Boucle d’entraînement
5. Evaluer

https://pytorch.org/docs/stable/

Deep learning avec Pytorch

2. Construire le modèle

Deep learning avec Pytorch

2. Construire le modèle

```python
import torch
import torch.nn as nn

# Define a simple fully connected neural network
class FCNN(nn.Module):  2 usages  new *
    def __init__(self):  new *
        super(FCNN, self).__init__()
        self.fc1 = nn.Linear( in_features: 10,  out_features: 64)  # 10 input features, 64 hidden neurons
        self.fc2 = nn.Linear( in_features: 64,  out_features: 32)  # 64 hidden neurons, 32 hidden neurons
        self.fc3 = nn.Linear( in_features: 32,  out_features: 1)   # 32 hidden neurons, 1 output neuron

    def forward(self, x):  new *
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)  # No activation on the output for regression
        return x

# Instantiate the model, loss function, and optimizer
model = FCNN()
```

import module

Initialise classe (nn.Module)

Initialise les couches

Définir le *forward*

Déclarer le modèle

Récap

Recette de l’entraînement d’un modèle

1. Préparer la donnée → Dataloader
2. Définir / construire le modèle → Réseaux de neurones
3. **Définir un objectif** → Fonction coût
4. Optimiser les paramètres → Boucle d’entraînement
5. Evaluer

https://pytorch.org/docs/stable/

Deep learning avec Pytorch

3. Définir une fonction loss

Deep learning avec Pytorch

3. Définir une fonction loss

```
criterion_regression = nn.MSELoss()  # Mean Squared Error loss for regression

criterion_classification = nn.CrossEntropyLoss() # Cross-Entropy Error loss for classification
```

→ Définir la loss

```
import torch.optim as optim
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

→ Définir l’optimizer

Récap

Recette de l’entraînement d’un modèle

1. Préparer la donnée → Dataloader
2. Définir / construire le modèle → Réseaux de neurones
3. Définir un objectif → Fonction coût
4. **Optimiser les paramètres** → Boucle d’entraînement
5. Evaluer

https://pytorch.org/docs/stable/

Deep learning avec Pytorch

4. Optimiser les paramètres (boucle d’entraînement)

Deep learning avec Pytorch

4. Optimiser les paramètres (boucle d’entraînement)

```python
# Training loop
epochs = 100
for epoch in range(epochs):
    model.train()
    epoch_loss = 0.0
    for X_batch, y_batch in train_loader:
        optimizer.zero_grad()
        output = model(X_batch)
        loss = criterion(output, y_batch)
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()

    # Print average loss for the epoch
    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch + 1}/{epochs}], Loss: {epoch_loss / len(train_loader):.4f}')
```

Deep learning avec Pytorch

4. Optimiser les paramètres (boucle d'entraînement)

```python
# Training loop
epochs = 100
for epoch in range(epochs):
    model.train()
    epoch_loss = 0.0
    for X_batch, y_batch in train_loader:
        optimizer.zero_grad()
        output = model(X_batch)
        loss = criterion(output, y_batch)
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()

    # Print average loss for the epoch
    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch + 1}/{epochs}], Loss: {epoch_loss / len(train_loader):.4f}')
```

Nb d'epochs

change comportement de nn.Module()

Loop sur le dataloader

Initialise les gradients à 0

Forward = calcule une prédiction

Calcul de l'erreur

Backprop l'erreur (calcul gradient)

Descente de gradient

Accumule l'erreur

Monitoring

Récap

Recette de l’entraînement d’un modèle

1. Préparer la donnée → Dataloader
2. Définir / construire le modèle → Réseaux de neurones
3. Définir un objectif → Fonction coût
4. Optimiser les paramètres → Boucle d’entraînement
5. **Evaluer**

https://pytorch.org/docs/stable/

Deep learning avec Pytorch

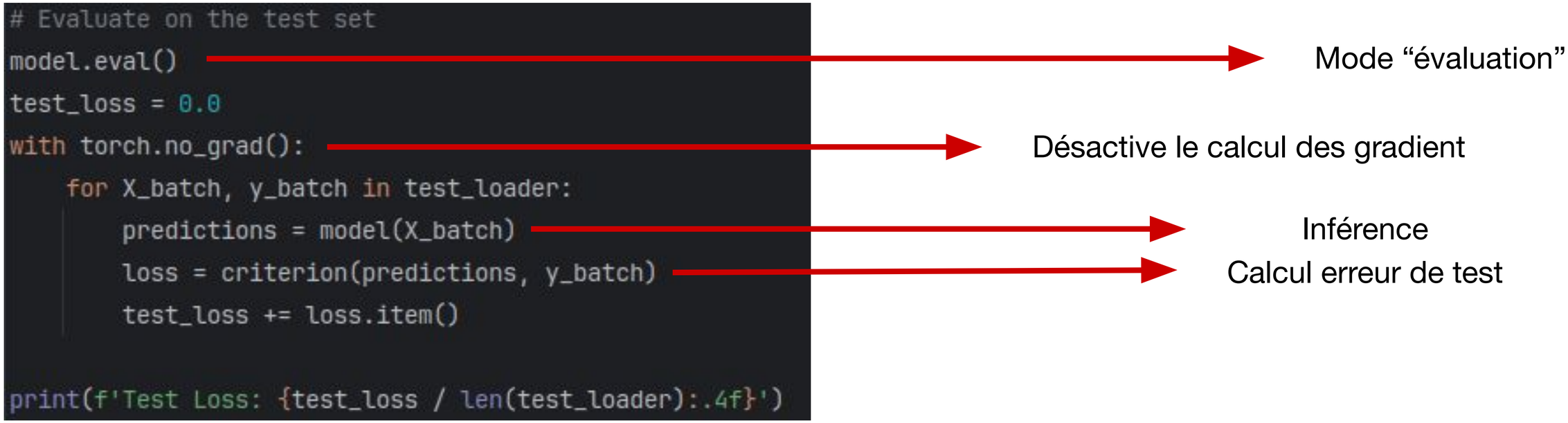
5. Evaluer

```
# Evaluate on the test set
model.eval()
test_loss = 0.0
with torch.no_grad():
    for X_batch, y_batch in test_loader:
        predictions = model(X_batch)
        loss = criterion(predictions, y_batch)
        test_loss += loss.item()

print(f'Test Loss: {test_loss / len(test_loader):.4f}')
```

Deep learning avec Pytorch

5. Evaluer

```
# Evaluate on the test set
model.eval()
test_loss = 0.0
with torch.no_grad():
    for X_batch, y_batch in test_loader:
        predictions = model(X_batch)
        loss = criterion(predictions, y_batch)
        test_loss += loss.item()

print(f'Test Loss: {test_loss / len(test_loader):.4f}')
```



Retour sur l’Optimizer

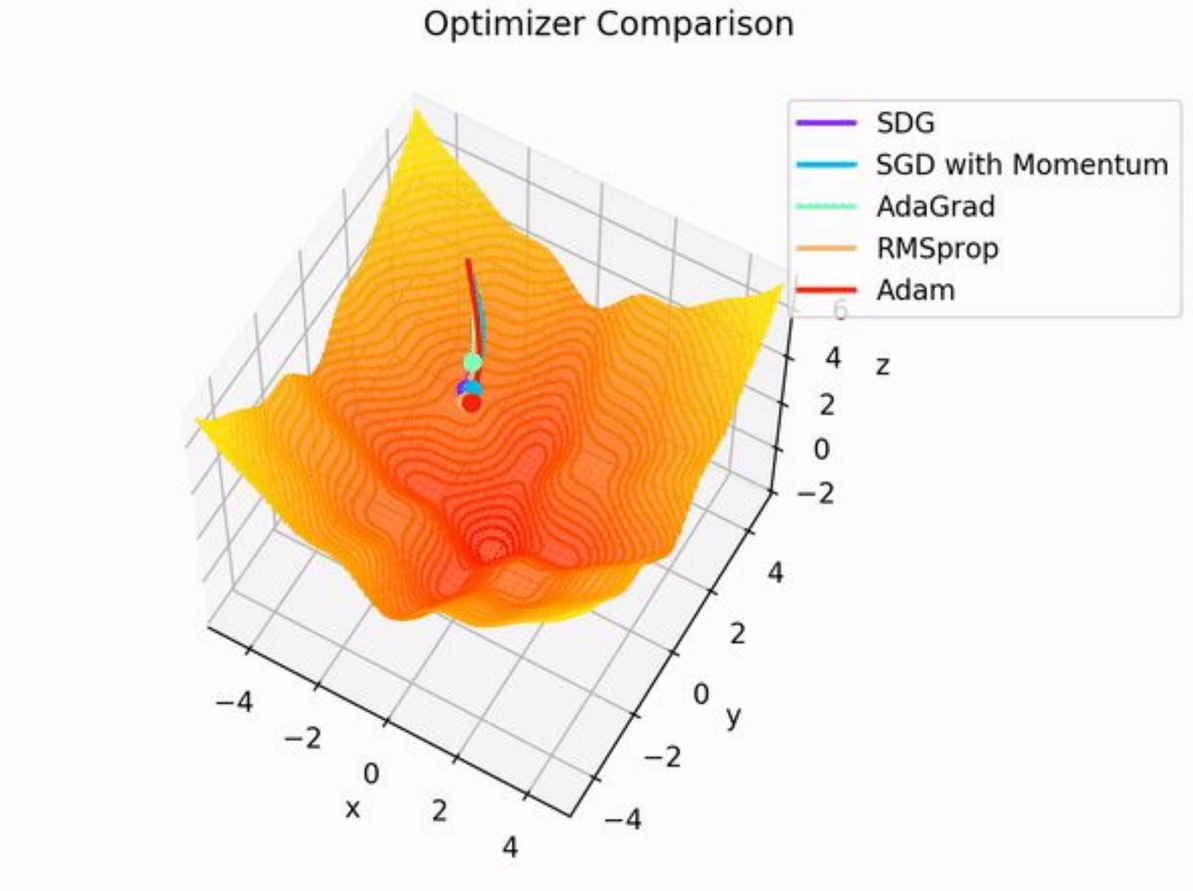
Stochastic Gradient Descent (SGD)

SGD + Moment

AdaGrad

RMSprop

Adam



Retour sur l’Optimizer

GD (Gradient Descent)

Initialisation aléatoire des paramètres

Calcul du gradient de la fonction coût $\delta = \nabla_\theta l(f(x), y)$ **(Moyenne sur l’ensemble des données d’entraînement)**

Faire un petit pas dans la direction opposée au gradient $\theta \leftarrow \theta - \eta\delta$

Continuer jusqu'à convergence

Retour sur l’Optimizer

SGD (Stochastic Gradient Descent)

Initialisation aléatoire des paramètres

Choisir aléatoirement une donnée d’entraînement

Calcul du gradient de la fonction coût $\delta = \nabla_\theta l(f(x), y)$

Faire un petit pas dans la direction opposée au gradient $\theta \leftarrow \theta - \eta\delta$

Continuer jusqu'à convergence

Retour sur l’Optimizer

SGD + Momentum

Initialisation aléatoire des paramètres

Choisir aléatoirement une donnée d’entraînement

Calcul du gradient de la fonction coût $\delta = \nabla_\theta l(f(x), y)$

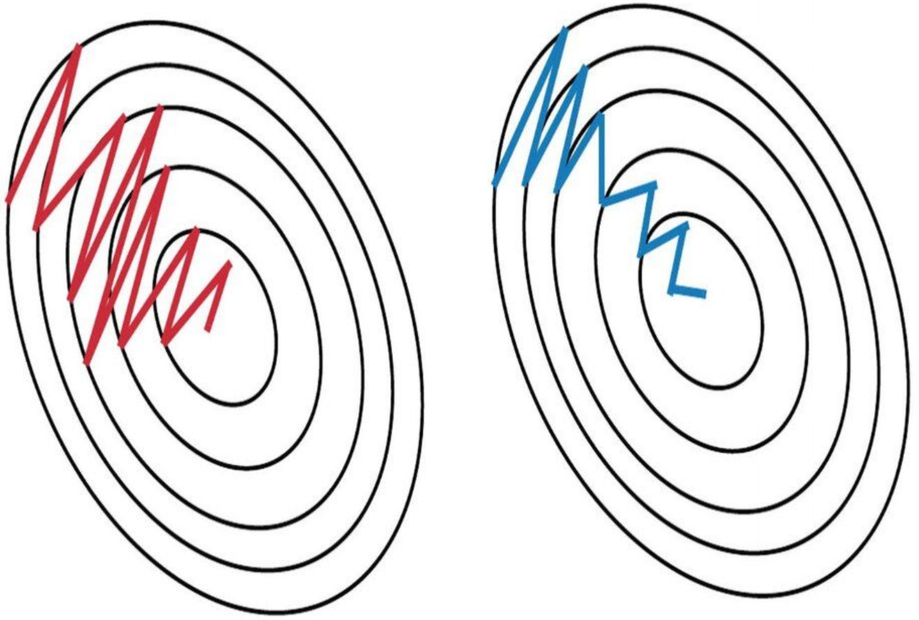
Calcul de la **vélocité** (accumulation du gradient) $v_t = \gamma v_{t-1} + \eta\delta$

Faire un petit pas dans la direction opposée au gradient $\theta \leftarrow \theta - v_t$

Continuer jusqu'à convergence

Retour sur l’Optimizer

SGD + Momentum



Initialisation aléatoire des paramètres

Choisir aléatoirement une donnée d’entraînement

Calcul du gradient de la fonction coût $\delta = \nabla_\theta l(f(x), y)$

Calcul de la **vélocité** (accumulation du gradient) $v_t = \gamma v_{t-1} + \eta\delta$

Faire un petit pas dans la direction opposée au gradient $\theta \leftarrow \theta - v_t$

Continuer jusqu'à convergence

Retour sur l’Optimizer

SGD + Momentum

Initialisation aléatoire des paramètres

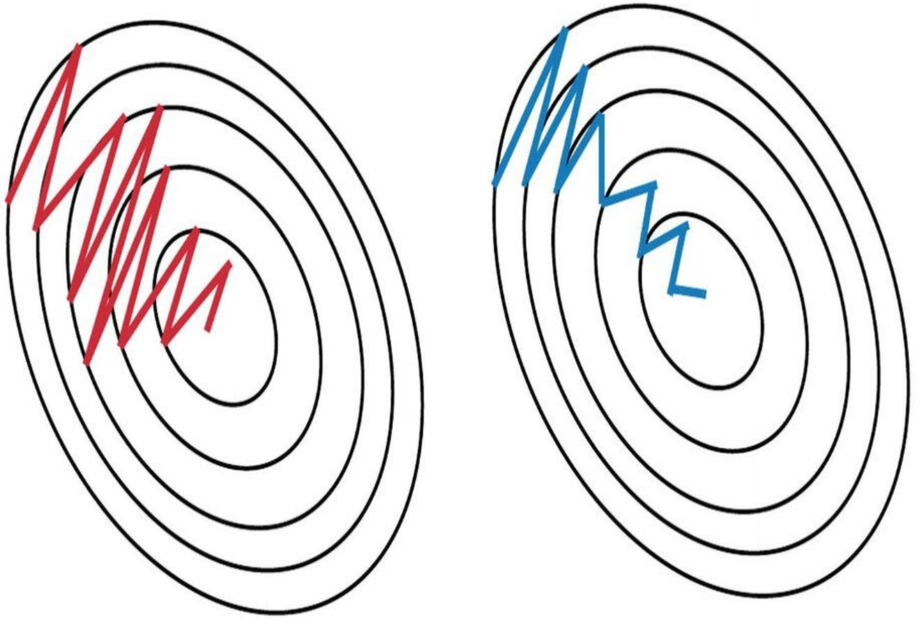
Choisir aléatoirement une donnée d’entraînement

Calcul du gradient de la fonction coût $\delta = \nabla_\theta l(f(x), y)$

Calcul de la **vélocité** (accumulation du gradient) $v_t = \gamma v_{t-1} + \eta\delta$

Faire un petit pas dans la direction opposée au gradient $\theta \leftarrow \theta - v_t$

Continuer jusqu'à convergence



SANS Momentum **AVEC Momentum**

Retour sur l’Optimizer

Adagrad (Adaptive gradient)

Initialisation aléatoire des paramètres

Choisir aléatoirement une donnée (ou un batch) d’entraînement

Calcul du gradient de la fonction coût $g_{t,i} = \nabla_{\theta_i} l(f(x), y)$

Accumulation du gradient $G_{t,i} = G_{t-1,i} + g_{t,i}^2$

Faire un petit pas dans la direction opposée au gradient $\theta_{i,t+1} = \theta_{i,t} - \frac{\eta}{\sqrt{G_{t,i}} + \epsilon} g_{t,i}$

Continuer jusqu'à convergence

Retour sur l'Optimizer

ADAM

Initialisation aléatoire des paramètres

Choisir aléatoirement une donnée (ou un batch) d'entraînement

Calcul du gradient de la fonction coût $g_{t,i} = \nabla_{\theta_i} l(f(x), y)$

Calcul du 1er moment $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_{t,i}$

Calcul du 2ème moment $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_{i,t}^2$

Faire un petit pas dans la direction opposée au gradient $\theta_{i,t+1} = \theta_{i,t} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$

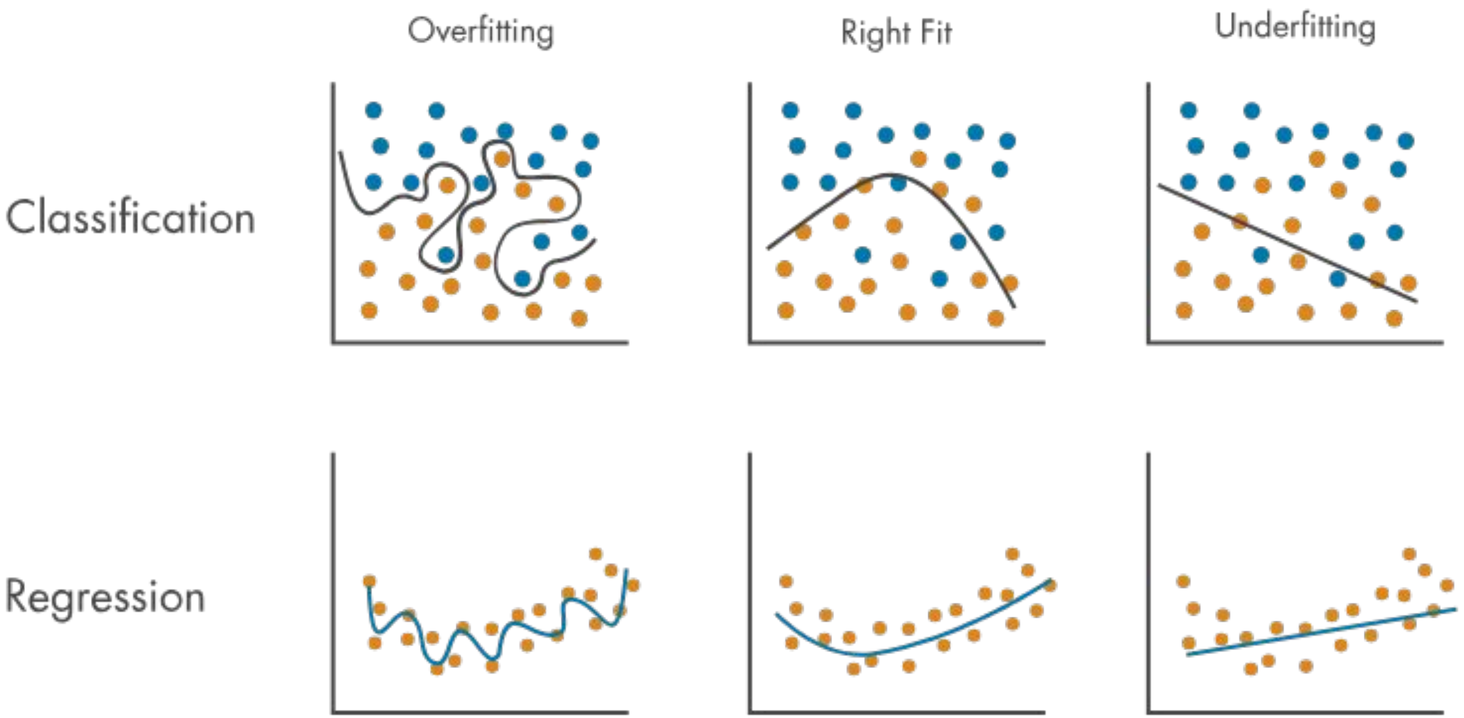
Continuer jusqu'à convergence

Correction du biais vers 0:

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

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Overfitting



Source : https://www.mathworks.com/discovery/overfitting.html

Comment vaincre l’Overfitting ?

La lutte contre l'overfitting est un aspect critique en apprentissage profond pour s'assurer que les modèles généralisent bien sur de nouvelles données. Il existe plusieurs méthodes et techniques pour lutter contre l'overfitting dans l'apprentissage profond :

Comment vaincre l’Overfitting ?

La lutte contre l'overfitting est un aspect critique en apprentissage profond pour s'assurer que les modèles généralisent bien sur de nouvelles données. Il existe plusieurs méthodes et techniques pour lutter contre l'overfitting dans l'apprentissage profond :

- **Plus de données**

Comment vaincre l’Overfitting ?

La lutte contre l'overfitting est un aspect critique en apprentissage profond pour s'assurer que les modèles généralisent bien sur de nouvelles données. Il existe plusieurs méthodes et techniques pour lutter contre l'overfitting dans l'apprentissage profond :

- **Plus de données**

- **Augmentation des données:** Application de transformations aléatoires qui font sens sur l’ensemble d’apprentissage.

Source : https://www.baeldung.com/cs/ml-data-augmentation

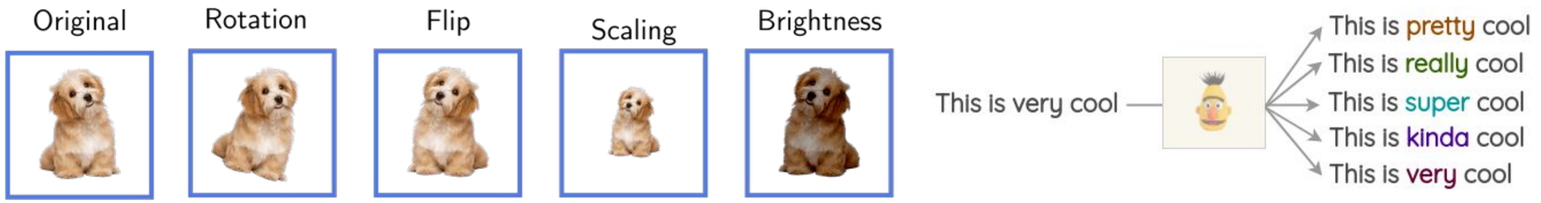
Source : Amit Chaudhary “A Visual Survey of Data Augmentation in NLP”

Comment vaincre l’Overfitting ?

La lutte contre l'overfitting est un aspect critique en apprentissage profond pour s'assurer que les modèles généralisent bien sur de nouvelles données. Il existe plusieurs méthodes et techniques pour lutter contre l'overfitting dans l'apprentissage profond :

- **Plus de données**

- **Augmentation des données:** Application de transformations aléatoires qui font sens sur l’ensemble d’apprentissage.



- **Régularisation de la fonction coût:** Ajouter un pénalité pour les valeurs extrême

Source : https://www.baeldung.com/cs/ml-data-augmentation

Source : Amit Chaudhary “A Visual Survey of Data Augmentation in NLP”

Tricks: Régularisation

Régularisation de la fonction coût: Ajouter un pénalité pour les valeurs extrême

Tricks: Régularisation

Régularisation de la fonction coût: Ajouter un pénalité pour les valeurs extrême

$$\tilde{l}(\hat{y}, y) = l(\hat{y}, y) + \lambda \sum_i |w_i| \longrightarrow \text{Lasso}$$

$$\tilde{l}(\hat{y}, y) = l(\hat{y}, y) + \lambda \sum_i \|w_i\|_2^2 \longrightarrow \text{Ridge}$$

Tricks: Régularisation

Régularisation de la fonction coût: Ajouter un pénalité pour les valeurs extrême

$$\tilde{l}(\hat{y}, y) = l(\hat{y}, y) + \lambda \sum_i |w_i| \longrightarrow \text{Lasso}$$

$$\tilde{l}(\hat{y}, y) = l(\hat{y}, y) + \lambda \sum_i \|w_i\|_2^2 \longrightarrow \text{Ridge}$$

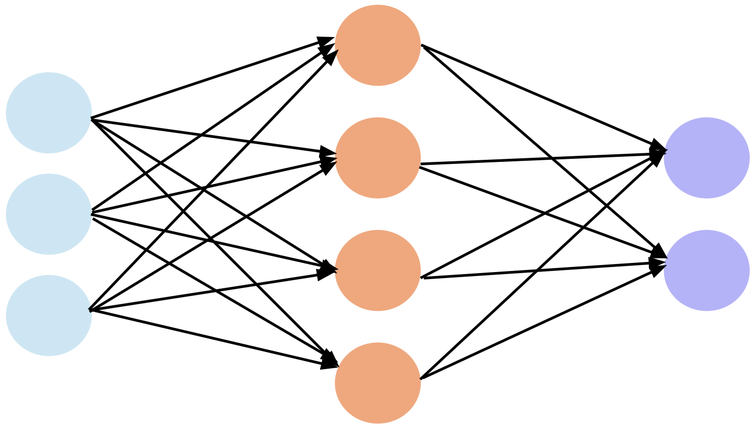
https://www.deeplearning.ai/ai-notes/regularization/index.html

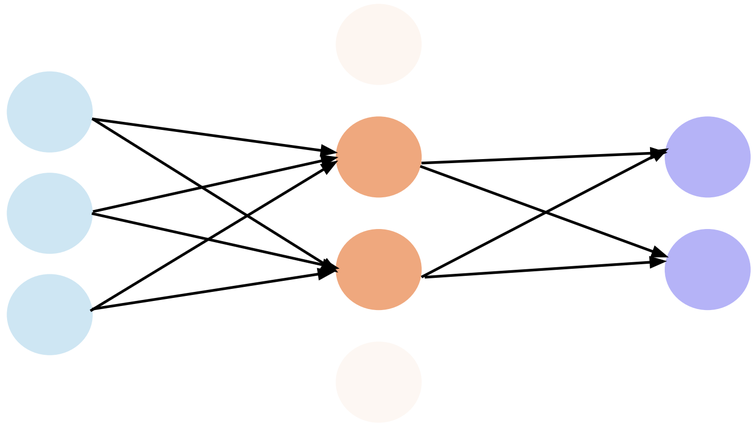
Tricks: Dropout

- Mettre aléatoirement un pourcentage des neurones à zéros pendant l’entraînement.
- Cela permet d’entraîner un modèle plus robuste et qui généralise mieux en évitant qu'un seul neurone ou groupe de neurones ne devienne trop spécialisé et trop dépendant des données d'apprentissage.

Tricks: Dropout

- Mettre aléatoirement un pourcentage des neurones à zéros pendant l’entraînement.
- Cela permet d’entraîner un modèle plus robuste et qui généralise mieux en évitant qu'un seul neurone ou groupe de neurones ne devienne trop spécialisé et trop dépendant des données d'apprentissage.





Mise à 0 de 50% des neurones
dans la couche cachée

Tricks: Batch Normalization

- « Batch normalization » (BatchNorm) est une technique d'apprentissage profond utilisée pour améliorer l’entraînement des réseaux neuronaux.
- Elle fonctionne en normalisant l'entrée de chaque couche dans un mini batch de données.
- Permet une convergence plus rapide pendant l’apprentissage, ce qui implique un entraînement de réseau de neurones plus profonds de manière plus efficace.

Tricks: Early Stopping

- Le « early stopping » ou en français l'arrêt anticipé est une stratégie visant à interrompre le processus d’apprentissage d'un réseau de neurones avant qu'il n'ait terminé toutes les epochs.
- Il s'agit d'évaluer périodiquement les performances du modèle sur un ensemble de données de validation distinct des données d'apprentissage.
- La décision d'arrêter l’entraînement est basée sur une mesure de performance choisie (par exemple, les valeurs de la fonction de coût sur les données de validation ou à partir de mesure tel que l’accuracy).
- L'objectif est de détecter le moment où les performances du modèle sur l'ensemble de validation commencent à se détériorer ou à atteindre un plateau.

TP 1: MLP et FC
pour classification MNIST