Réseaux de Neurones

Luong Phat NGUYEN

École Polytechnique de l'université de Tours

2020

POLYTECH TOURS
Ecole d'ingénieurs polytechnique de l'université de Tours

LIFAT EA 6300
Laboratoire d'Informatique Fondamentale et Appliquée de Tours

Plan

1. Introduction
2. Descente de Gradient
3. Perceptron
4. MLP
5. Règles d'Apprentissage
6. Régularisation

Introduction

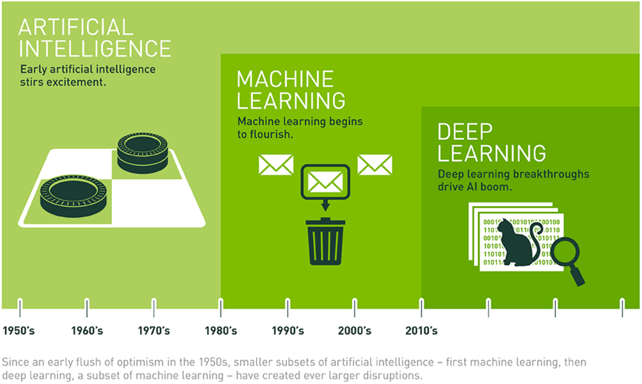


FIGURE – La relation entre Intélligence Artificielle, Machine Learning et Deep Learning. Source : What's the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?

Histoire

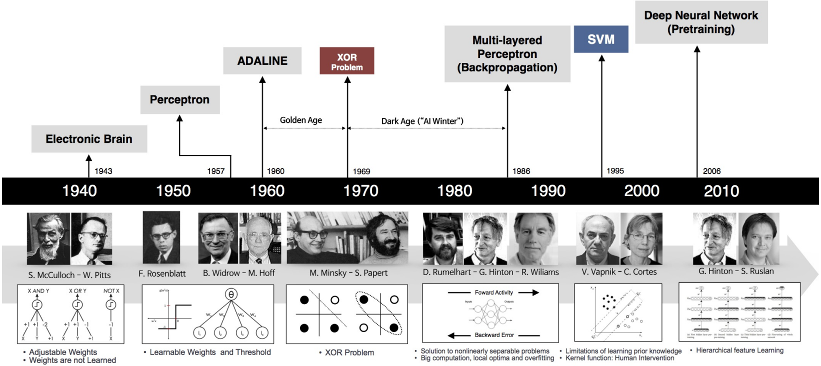


FIGURE – L'histoire générale de Deep Learning. Source : Deep Learning 101 - Part 1: History and Background

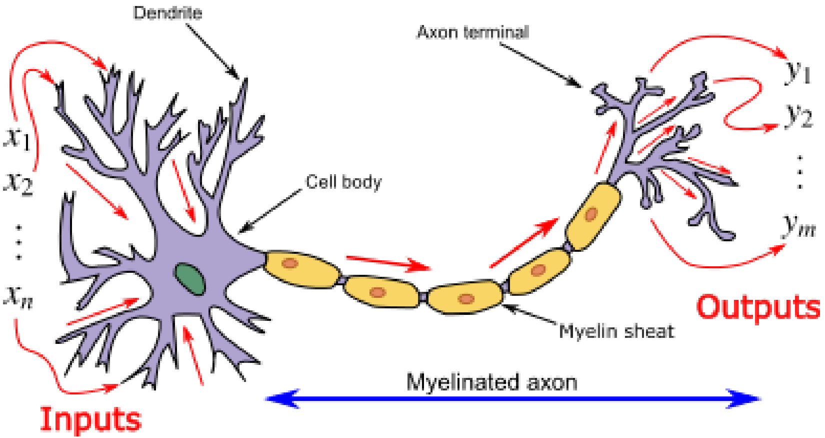


Figure – Réseau de neurones est équivalent à une conception des neurones artificiels du cerveau qui reçoivent des entrées (images, sons, etc.) et produisent des sorties correspondantes. Source : Artificial neural network

Apprentissage

En général, on présente des exemples et on modifie les poids en fonction des sorties obtenues.

- Supervisé : minimise écart entre sortie obtenue et sortie désirée
- Renforcé : pénalité/récompense
- Non supervisé : regroupement des exemples en fonction de ressemblance que le RN doit extraire

Notation

a	Un scalaire
$\boldsymbol{a}$	Un vecteur
$\boldsymbol{A}$	Une matrice
$\mathbf{A}$	Un tenseur
$\mathbb{A}$	Un ensemble
$\mathbb{R}$	Un ensemble de réels
$\{0, 1, ..., n\}$	Un ensemble d'intègre entre 0 et n
$[a, b]$	L'intervalle réel comprenant a et b
$[a, b)$	L'intervalle réel comprenant a et sauf b
a_i	i^e élément de vecteur $\boldsymbol{a}$, l'indexation commence à 1
$A_{i,j}$	Élément i, j de matrice $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	i^e ligne de matrice $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	i^e colonne de matrice $\boldsymbol{A}$

Notation

$\boldsymbol{A}^T$	Transposé de matrice $\boldsymbol{A}$
$\boldsymbol{A} \odot \boldsymbol{B}$	produit matriciel de Hadamard des 2 matrices
$\det(\boldsymbol{A})$	Déterminant de matrice $\boldsymbol{A}$
$\frac{dy}{dx}$	Dérivée de y par rapport à x
$\frac{\partial y}{\partial x}$	Dérivée partielle de y par rapport à x
$\nabla_{\boldsymbol{x}} y$	Gradient de y par rapport à $\boldsymbol{x}$
$\nabla_{\boldsymbol{X}} y$	Gradient matricielle de y par rapport à $\boldsymbol{X}$
$\nabla_{\mathbf{X}} y$	Tenseur comprenant gradient de y par rapport à $\mathbf{X}$

Algorithme de Descente de Gradient

- On a une fonction $f(\boldsymbol{\theta})$ où $\boldsymbol{\theta}$ est un vecteur de coordonnées.
- Objectif : Trouver un minimum global pour la fonction $f(\boldsymbol{\theta})$, idéalement :

$$\frac{\partial f(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = 0$$

- Règle de mise à jour pour trouver $\boldsymbol{\theta}^*$ le minimum global : l'algorithme commence avec une initialization θ_0 aléatoire et après t itérations, nous avons

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_t)$$

  Ou en forme courte : $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$

Algorithme de Descente de Gradient

Exemple :
$f(x) = x^2 + 5\sin(x)$
$x_0 = -5,\ \eta = 0.1$
Calcul l'extrème de $f(x)$ en 2 manières :

- $\frac{\partial f}{\partial x} = 0.$
- Utilier l'algorithme de descente de gradient.

Algorithme de Descente de Gradient

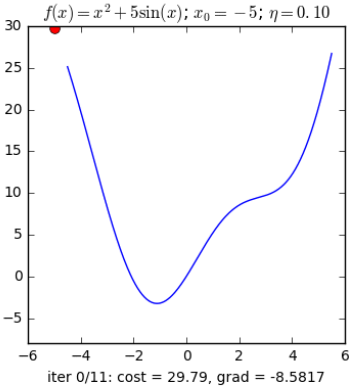


Figure – Exemple de l'algorithme de gradient en $1D$

Algorithme de Descente de Gradient

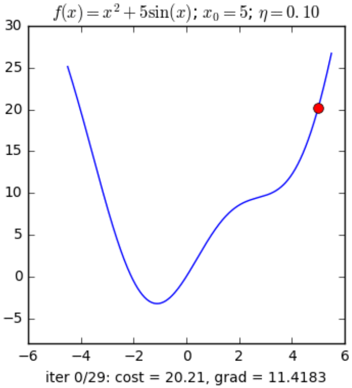


FIGURE – Exemple de l'algorithme de gradient en $1D$

Algorithme de Descente de Gradient

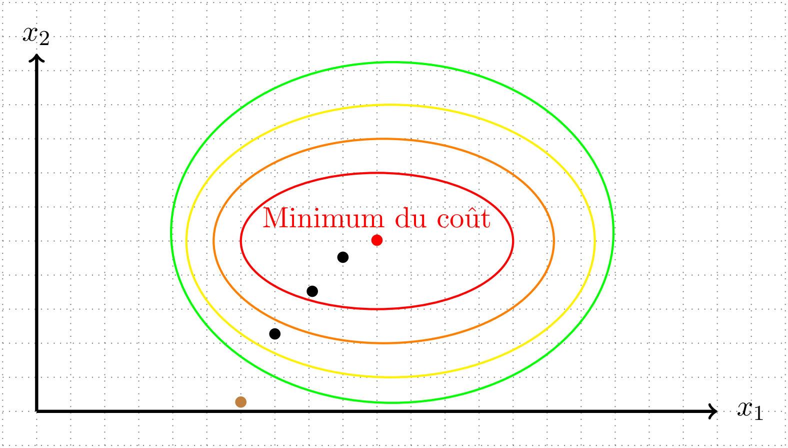


Figure – Exemple de l'algorithme de gradient en $2D$

Algorithme de Perceptron

Réseau de neurones à une seule couche de poids (pas de couche cachée)

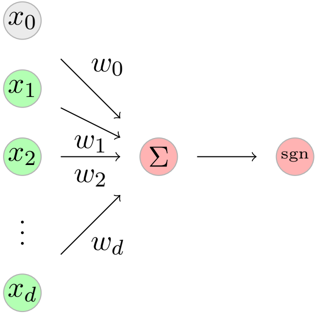
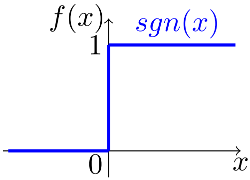


FIGURE – Illustration de l'algorithme de perceptron



$$y_i = sgn(\mathbf{w}^T \boldsymbol{x_i})$$

Avec $\boldsymbol{x} \in \mathbb{R}^d$ et $\mathbf{w} \in \mathbb{R}^d$.

Fonction de coût :

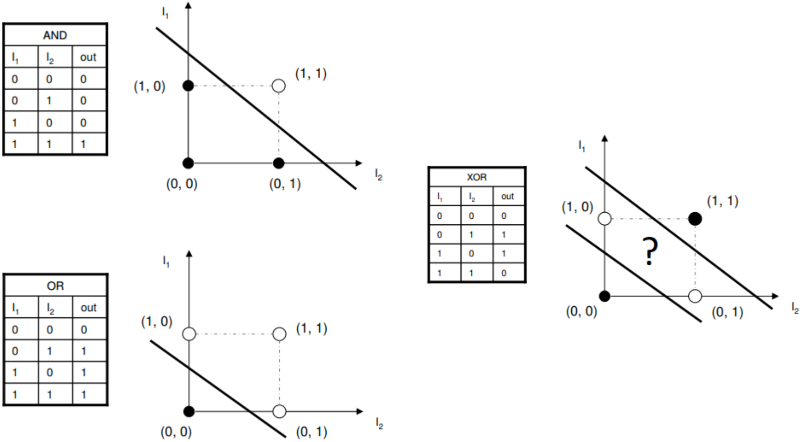
$J = -y_i \mathbf{w} \boldsymbol{x_i}$.

$\longrightarrow \nabla_{\mathbf{w}} J = -y_i \boldsymbol{x_i}$

$\longrightarrow$ La mise à jour des paramètres :

$\mathbf{w} = \mathbf{w} + \eta y_i \boldsymbol{x_i}$

Algorithme de Perceptron - Problème de XOR



AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1

OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1

XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0

FIGURE – Problème avec XOR. Source : Solving XOR with a single Perceptron

Perceptron Multicouche (MLP)

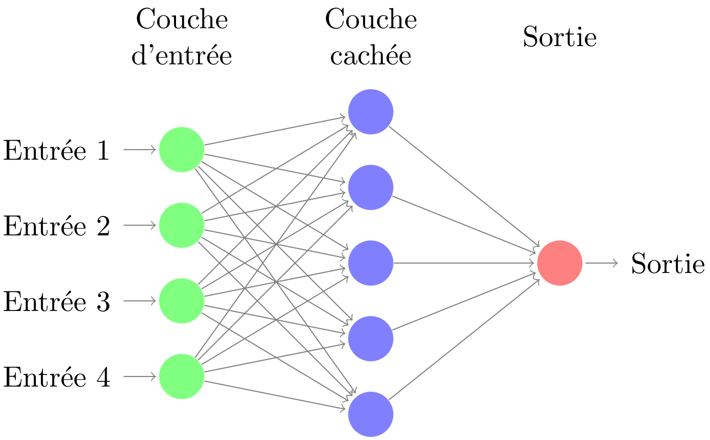
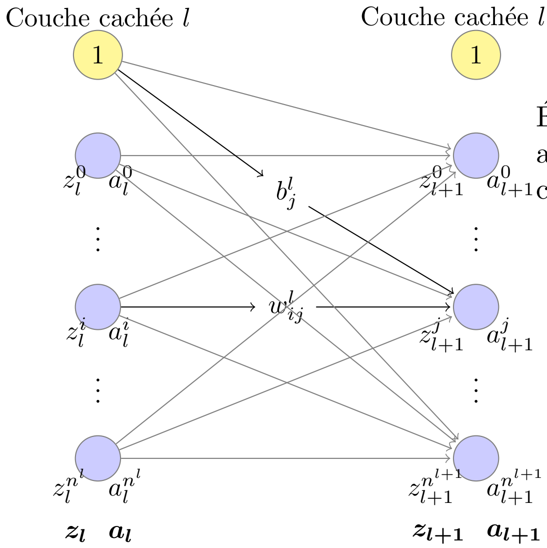


Figure – Exemple de réseaux de neurones de 2 couches.

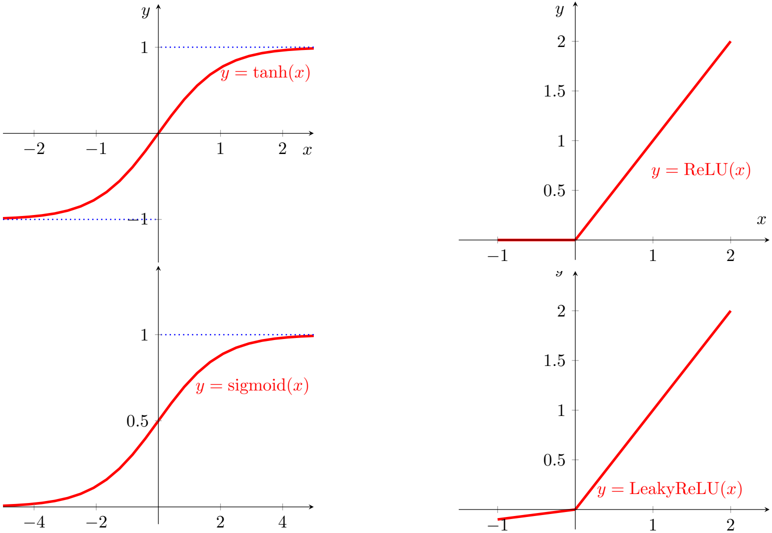
Propagation avant



Étapes de propagation avant entre 2 couches cachées l et $l+1$:

1. $\mathbf{W}_l \in \mathbb{R}^{d^l \times d^{l+1}}$
2. $\boldsymbol{b}_l \in \mathbb{R}^{d^{l+1} \times 1}$
3. $z_{l+1}^j = (\boldsymbol{w}_l^j)^T \boldsymbol{a}_l + b_j^l$
   $\boldsymbol{z}_{l+1} = \mathbf{W}_l^T \boldsymbol{a}_l + \boldsymbol{b}_l$
4. $\boldsymbol{a}_{l+1} = f(\boldsymbol{z}_{l+1})$ avec f la fonction d'activation.

Fonctions d'activation



Fonctions d'activation

Pour la classification multi-classes, la fonction d'activation de softmax est souvent utilisé *one-hot encoding* pour la dernière couche du réseaux de neurones.
softmax : la sortie est un vecteur d'une somme des éléments égale à 1

$$\text{softmax}(\boldsymbol{x}) = \frac{exp(\boldsymbol{x})}{\sum_{i=1}^{N} \boldsymbol{x}} \tag{1}$$

Avec N le nombre d'éléments.

Base de Données

3 bases pour l'apprentissage supervisé :

1. **Base d'apprentissage** :
   - Utilisée pour ajuster des paramètres du modèle.
   - Éviter sur-apprentissage.
2. **Base de validation** :
   - Pas toujours utilisée.
   - Utilisée pour ajuster des hyperparamètres dans le modèle (poids ou biais).
   - Utilisée pour la régularisation.
3. **Base de test** :
   - Utilisée pour fournir une évaluation non biaisée du modèle final adapté à la base d'apprentissage.
   - Suit la même distribution de probabilité que la base d'apprentissage
   - Évaluer la capacité de généralisation du modèle entraîné.

Fonctions de coût

$J(\hat{\boldsymbol{y}}, \boldsymbol{y})$: une fonction de coût ou une fonction de perte est une fonction qui mesure la différence entre la sortie d'un modèle $\hat{\boldsymbol{y}} \in \mathbb{R}^M$ et la vérité terrain $\boldsymbol{y} \in \mathbb{R}^M$

Quelques fonctions de coût pratiques :

- Erreur quadratique moyenne :

$$J(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{M} \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|^2$$

- Erreur absolue moyenne :

$$J(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{M} \sum_{i=1}^{M} |\hat{y}_i - y_i|$$

- Entropie croisée (cross-entropy) :

$$J(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{i=1}^{M} y_i \log \hat{y}_i \quad avec\ y_i, \hat{y}_i \in [0, 1]$$

Propagation arrière

- *One-hot encoding* : $\hat{\boldsymbol{y}} \in \mathbb{R}^{M\times 1} \to \hat{\mathbf{Y}} \in \mathbb{R}^{M\times C}$, $\boldsymbol{y} \in \mathbb{R}^{M\times 1} \to \mathbf{Y} \in \mathbb{R}^{M\times C}$.
- La fonction de coût :

$$J(\hat{\mathbf{Y}}, \mathbf{Y}) = -\frac{1}{M}\sum_{i=1}^{M}\sum_{j}^{C} y_{ji}\log \hat{y}_{ji}$$

  Où M est le nombre d'instances ; C est le nombre de classes (labels) ; $\mathbf{X}, \mathbf{Y}$ sont les données et les labels de la base d'apprentissage ; $\mathbf{W}, \boldsymbol{b}$ sont les ensembles de poids et de biais du réseau.
- Pour appliquer les méthodes basées des algorithmes de gradients, il faut calculer : $\frac{\partial J}{\partial \mathbf{W}^l}, \frac{\partial J}{\partial \boldsymbol{b}^l}$, $l = 1, 2, ..., L$

Propagation arrière (1)

1. Calcul d'erreur de la couche sortie ($\boldsymbol{e}^L$) :

$$\boldsymbol{e}^L = \frac{\partial J}{\partial \boldsymbol{z}^L}$$

2. Calcul le gradient de l'erreur selon $\mathbf{W}^L$ et $\boldsymbol{b}^L$:

$$\frac{\partial J}{\partial \mathbf{W}^L} = \frac{\partial J}{\partial \boldsymbol{z}^L} \frac{\partial \boldsymbol{z}^L}{\partial \mathbf{W}^L} = \boldsymbol{e}^L (\boldsymbol{a}^{L-1})^T$$

$$\frac{\partial J}{\partial \boldsymbol{b}^L} = \frac{\partial J}{\partial \boldsymbol{z}^L} \frac{\partial \boldsymbol{z}^L}{\partial \boldsymbol{b}^L} = \boldsymbol{e}^L$$

Propagation arrière (2)

3. Pour chaque $l = L-1, L-2, ...2, 1$, calcul d'erreur de sortie de la couche l :

$$\boldsymbol{e}^l = (\mathbf{W}^{l+1}\boldsymbol{e}^{l+1}) \odot f'(\boldsymbol{z}^l)$$

Où $\odot$ est la multiplication par élément entre 2 tenseurs de dimension égale.

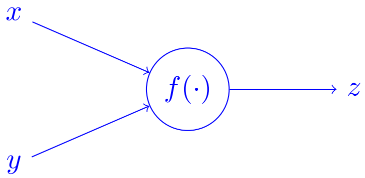
4. Répéter la procédure de la couche l à $l-1$:

$$\frac{\partial J}{\partial \mathbf{W}^l} = \frac{\partial J}{\partial \boldsymbol{z}^l}\frac{\partial \boldsymbol{z}^l}{\partial \mathbf{W}^l} = \boldsymbol{e}^l(\boldsymbol{a}^{l-1})^T$$
$$\frac{\partial J}{\partial \boldsymbol{b}^l} = \frac{\partial J}{\partial \boldsymbol{z}^l}\frac{\partial \boldsymbol{z}^l}{\partial \boldsymbol{b}^l} = \boldsymbol{e}^l$$

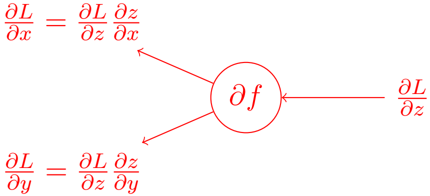
Perceptron Multicouche

Propagation avant : fournir le réseau un échantillon des entrées, calculer l'activation et la sortie pour chaque neurone dans l'ordre croissant.
Propagation arrière : évaluer la dérivée de la fonction de pertes pour les paramètres et les mettre à jour pour chaque neurones dans l'ordre décroissant.

Propagation avant



Propagation arrière



Perceptron Multicouche

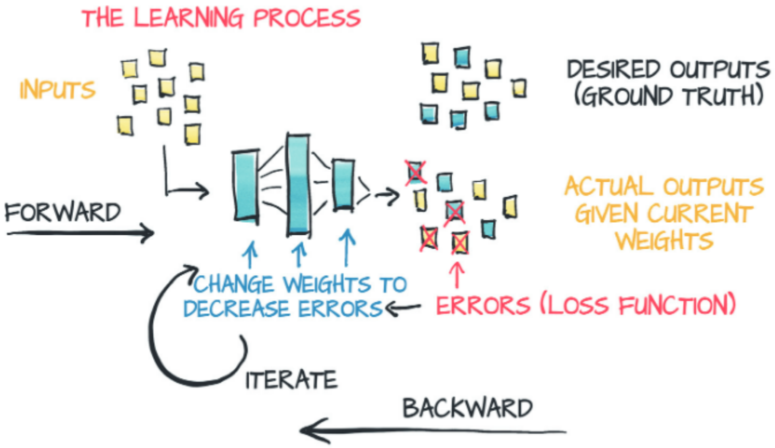


FIGURE – Processus d'apprentissage général. Source : Deep Learning with Pytorch

Batch – Descente de Gradient Stochastique

- Apprentissage par batch : toutes les données dans la base d'apprentissage $\mathbf{X}$ sont fournies dans le réseau pour l'entrainement. Le moyenne du gradient de la "loss" $J(\mathbf{X}, \mathbf{Y})$ est utilisé pour update les paramètres θ_j.

$$\theta_j^{t+1} = \theta_j^t - \frac{\eta}{M} \frac{\partial J(\mathbf{X}, \mathbf{Y})}{\partial \theta_j}$$

Risque d'exploser la mémoire si les données sont lourdes !

- Apprentissage par descente gradient stochastique : un exemple quelconque de la base d'apprentissage $\boldsymbol{x}_i$ est utilisé et update les paramètres θ_j.

$$\theta_j^{t+1} = \theta_j^t - \eta \frac{\partial J(\boldsymbol{x}_i, \boldsymbol{y}_i)}{\partial \theta_j}$$

Mini-batch Descente de Gradient Stochastique

- Apprentissage par mini-batch : des sous-ensemble quelconques $\mathbf{X}_k$ de la base d'apprentissage $\mathbf{X}$ est utilisé et update les paramètres θ_j.

$$\theta_j^{t+1} = \theta_j^t - \frac{\eta}{K} \frac{\partial J(\mathbf{X}_k, \mathbf{Y}_k)}{\partial \theta_j}$$

K : nombre de sous-ensembles dans la base d'apprentissage

Remarques

- La fonction de coût est non-convexe $\longrightarrow$ beaucoup de locaux minimums.
- Descente de gradient converge vers l'un des locaux minimums.

Régularisation

- Régularisation est une méthode qui aide le modèle à éviter le sur-apprentissage en gardant la généralisation du modèle.
- Des méthodes de régularisation pratiques :
  - Ajouter un terme de régularisation à la fonction de coût
  - Augmentation de nombre de données
  - Dropout
  - Early stopping

Régularisation

Ajouter un terme de régularisation à la fonction de coût

$$J_{reg}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda R(\boldsymbol{\theta})$$

Où λ est paramètre de régularisation et $\lambda R(\boldsymbol{\theta})$ s'appelle terme de régularisation.
Les termes de régularisation souvant utilisés :

- Régularisation de L2 :

$$R(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2 = \sum_{i=0}^{M} \theta_i^2$$

- Régularisation de L1 :

$$R(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2 = \sum_{i=0}^{M} \theta_i$$

Régularisation

Dropout : quelques neurones quelconques dans le réseau sont mis à 0. La probabilité d'enlever des neurones est un hyperparamètre à régler.

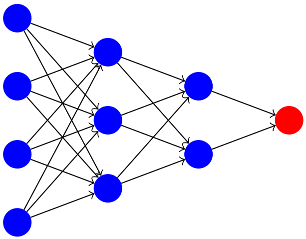


FIGURE – Sans Dropout.

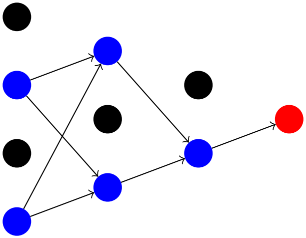


FIGURE – Avec Dropout de probabilité de $0,44$.

Regularisation

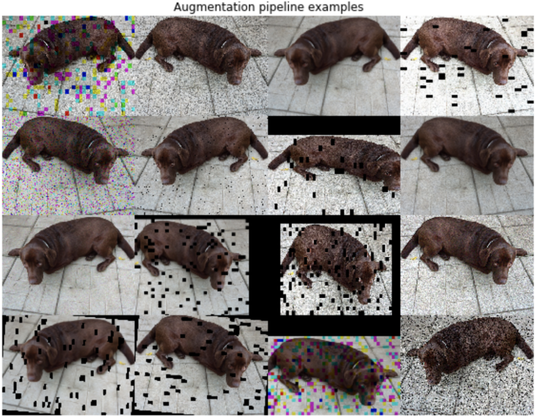


FIGURE – L'augmentation des données par : ajouter de bruit, ajuster de contraste, rotation, etc. Source : Data Augmentation for Deep Learning.