Majeure Machine Learning

Deep learning
Optimisation

Contenu

- Stochastic / Mini-batch Gradient Descent

- Régularisation

- Dropout

- Early stopping

- Learning rate decay / adaptatif

- Pre-training (Transfer Learning / Semi-Supervised)

Ce que vous devrez savoir faire

- Comprendre les différentes techniques de régularisation pour les RNNs

- Comprendre les différentes variantes liées à la descente de gradient

- Comprendre le principe et l'avantage du Transfer Learning

- Comprendre l'intuition de l'apprentissage semi-supervisé

Initialisation des poids

Initialisation des poids & biais

- Pour les biais :
  - Initialisation à 0

- Pour les poids :
  - Impossible de tout initialiser à 0 ou à la même valeur
    - Tous les neurones auraient le même comportement
    - Besoin de casser la symétrie

  - Une solution :

$W_{i,j}^{(k)}$ généré à partir de $U[-b, b]$ où $b = \frac{\sqrt{6}}{\sqrt{H_k + H_{k-1}}}$

  - Autre solution (Xavier) :

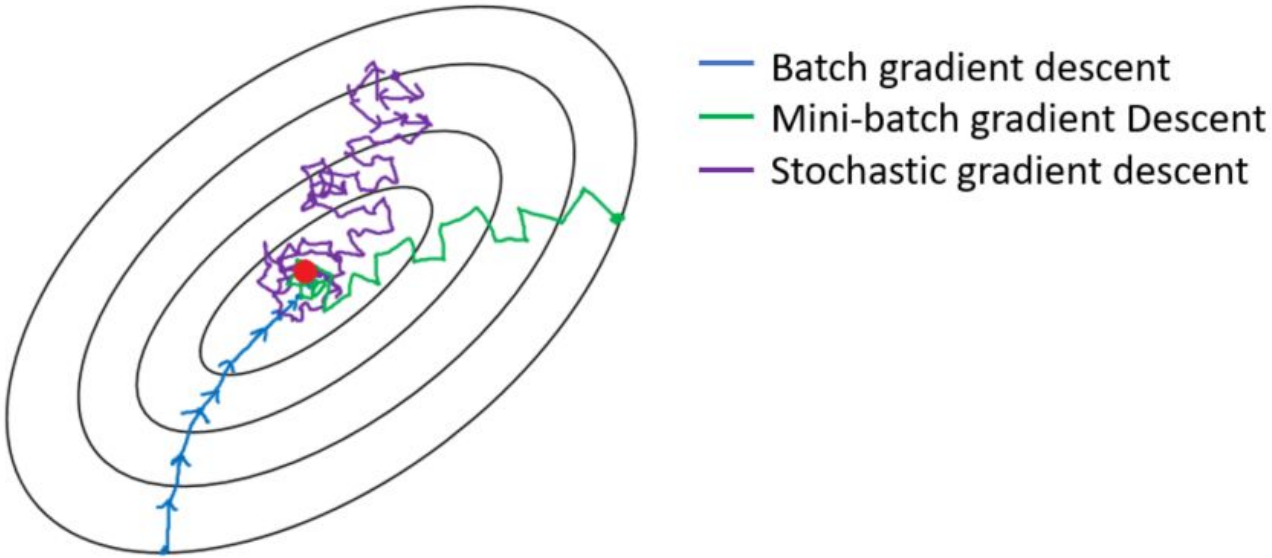
$W_{i,j}^{(k)}$ généré à partir de $\mathcal{N}(0, Var(W))$ où $Var(W) = \frac{2}{H_{k-1} + H_k}$

- k => couche
- H_k => nombre de neurones dans la couche k

Fonctions pour minimiser la fonction de coût

Descente de Gradient - Variations

Motivation : Converger vers un meilleur optimum local



Descente de Gradient - Vocabulaire

Epoch : Cycle complet où la totalité du jeu de données est passé dans la phase de “forward” et la “backward”

Mini-batch size : Nombre d’exemples utilisés pour un batch

Itération : Le nombre de batch nécessaires pour compléter un epoch

Ex : pour un jeu d'entraînement de 2016 exemples, une taille de batch de 32, il faudra 63 itérations pour effectuer un epoch, et plusieurs epoch pour converger.

Plus d'informations

Descente de Gradient - Variations

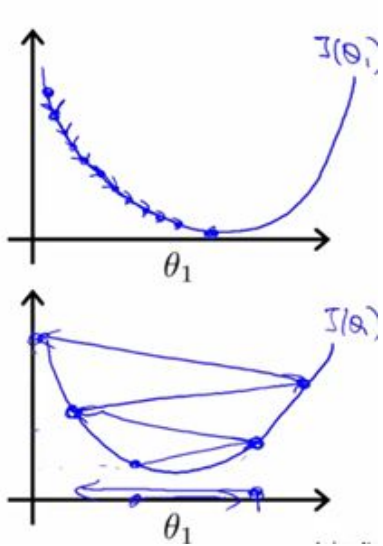
	Stochastic	Batch	Mini-Batch
Calcul de l'erreur	Pour chaque exemple	Pour tous les exemples	Pour chaque mini-batch
Mise à jour du modèle	Pour chaque exemple	Après évaluation de l'ensemble des données	Pour chaque mini-batch
Vitesse de convergence	rapide	lente	intermédiaire
Utilisation	Robuste	Efficace	Robuste et efficace

Learning rate decay

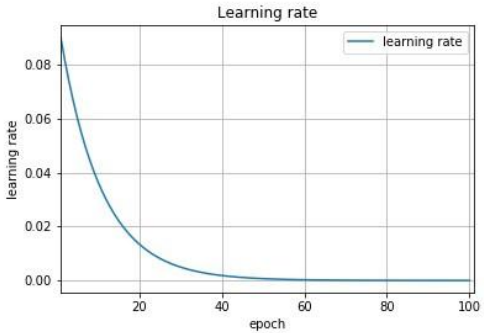
Rappel sur le learning rate :

$$\theta_1 = \theta_1 - \alpha \frac{\partial J(\theta)}{\partial \theta_1}$$

=> 0 < α < 1, le pas d'avancement



Exponential learning rate decay :



Motivation : Converge plus rapidement et trouve un meilleur local optimum

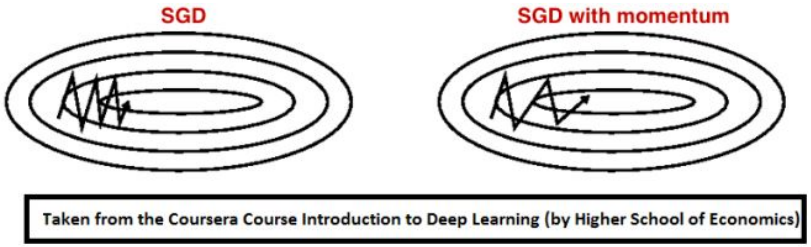
Adam - Adaptive Moment Estimation

ADAM = SGD + Momentum + RMSProp

Momentum :

-Accumule les gradients des étapes précédent

Intuition : permet de bénéficier de la “pente”



Taken from the Coursera Course Introduction to Deep Learning (by Higher School of Economics)

RMSProp (Adaptative learning rate):

- Calcule un learning rate différent pour chaque paramètre

Intuition : Les learnings rates s’adaptent en fonction de la pente de la courbe (historique du gradient)

Point d’attention : Les learnings rates (alpha) s’adaptent ici à chaque itération.

Méthode à privilégier !

Motivation : Converge plus rapidement et trouve un meilleur local optimum que SGD

Réduction de la capacité

La Régularisation

=> Objectif : Réduire la capacité pour prévenir l'Overfitting

Régression linéaire :

$$J(\theta) = \frac{1}{N} \sum_{n=1}^{N} (\hat{y}(x_n, \theta) - y(x_n))^2 + \lambda \sum_j \theta_j{}^2$$

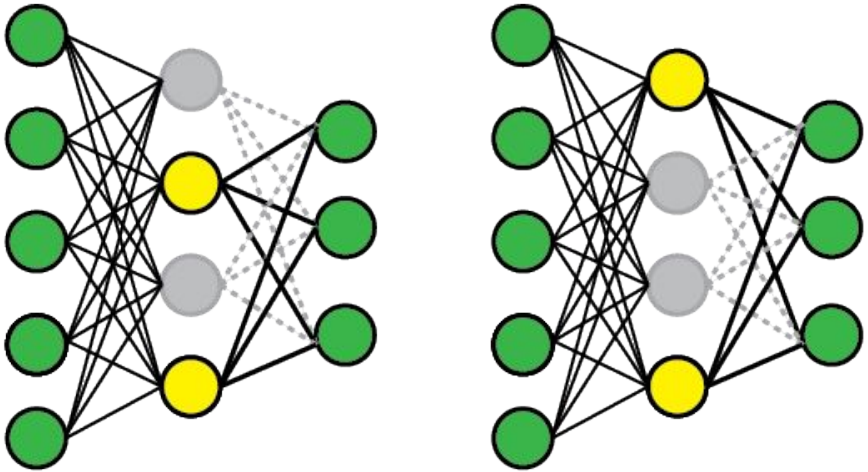
Avec λ > 0, la "force" de régularisation

Réseau de neurones :

$$J(w) = \frac{1}{2N} \sum_{n=1}^{N} (\hat{y}(x_n, w) - y(x_n))^2 + \frac{\lambda}{2N} \sum_w w^2$$

Dropout

=> Objectifs : Réduire la capacité pour prévenir l'Overfitting + augmenter la robustesse du réseau



Procédé :

Eteindre* aléatoirement une partie des neurones des couches cachées durant l'apprentissage

Intuition => Les neurones sont obligés d'apprendre des concepts plus généraux

* on fixe l'output à 0

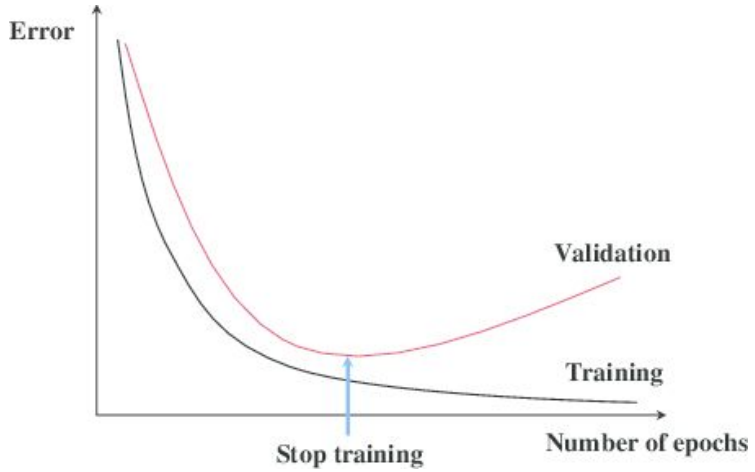
Early Stopping

=> Objectif : Arrêter l'entraînement au début de l'overfitting

Train : 80%
Validation : 10%
Test : 10%

Algorithme :

Calculer l'erreur sur le jeu de validation tous les n epochs
=> Si l'erreur n'a pas diminué depuis m epochs, arrêter l'entraînement



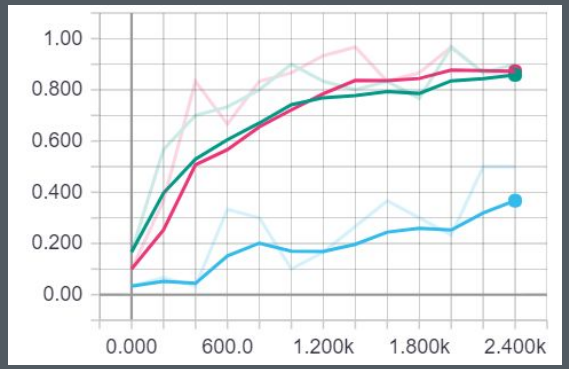
Problèmes des réseaux de neurones

Gradient Vanishing

Énoncé : Au fur à mesure que l'on rajoute des couches cachés, les gradients des couches les plus à gauches deviennent de plus en plus petit (et ont du mal à apprendre)

Conséquence : L'apprentissage est long et les premières couches apprennent mal

Solution : ne pas utiliser la Sigmoid ou la Tanh comme fonction d'activation mais privilégier la ReLu

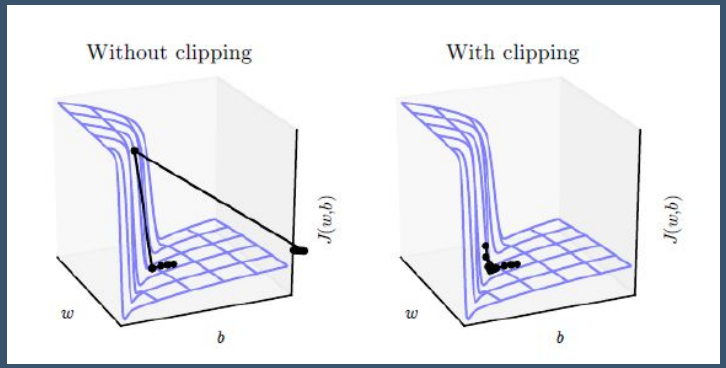


Gradient Exploding

Énoncé : Les valeurs des gradients et donc des poids des premières couches peuvent être très grandes et peuvent tendre vers l'infini

Conséquence : Les valeurs des poids et des gradients peuvent être égales à NaN

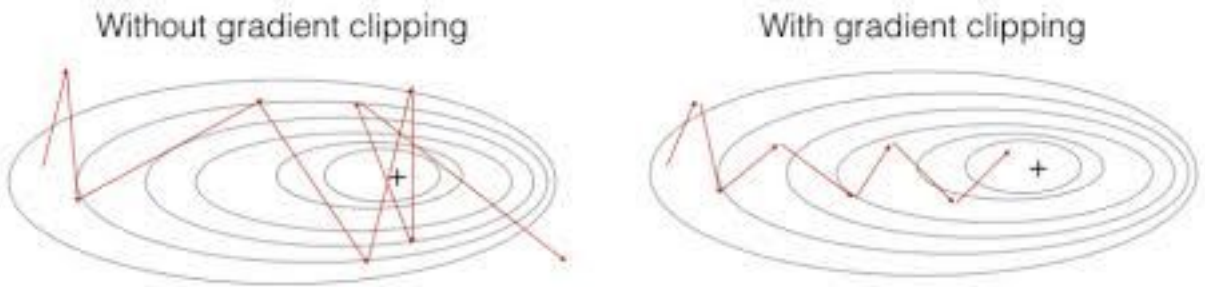
Solution : utiliser le Gradient Clipping



Gradient clipping

Motivation : Eviter le gradient exploding

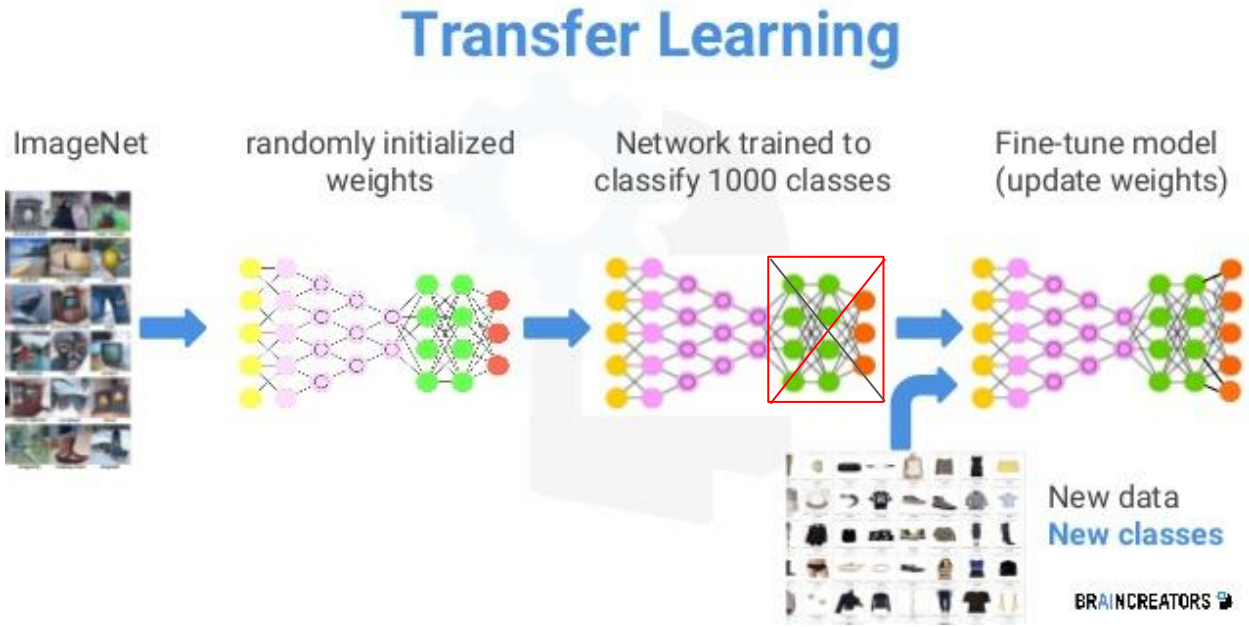
Principe : définir un intervalle (min,max) tel que la valeur des poids ne pourra jamais être en dehors de cet intervalle



Pre-training

Transfer Learning

Motivation : Les relations apprises dans un cas peuvent être utilisées dans un autre



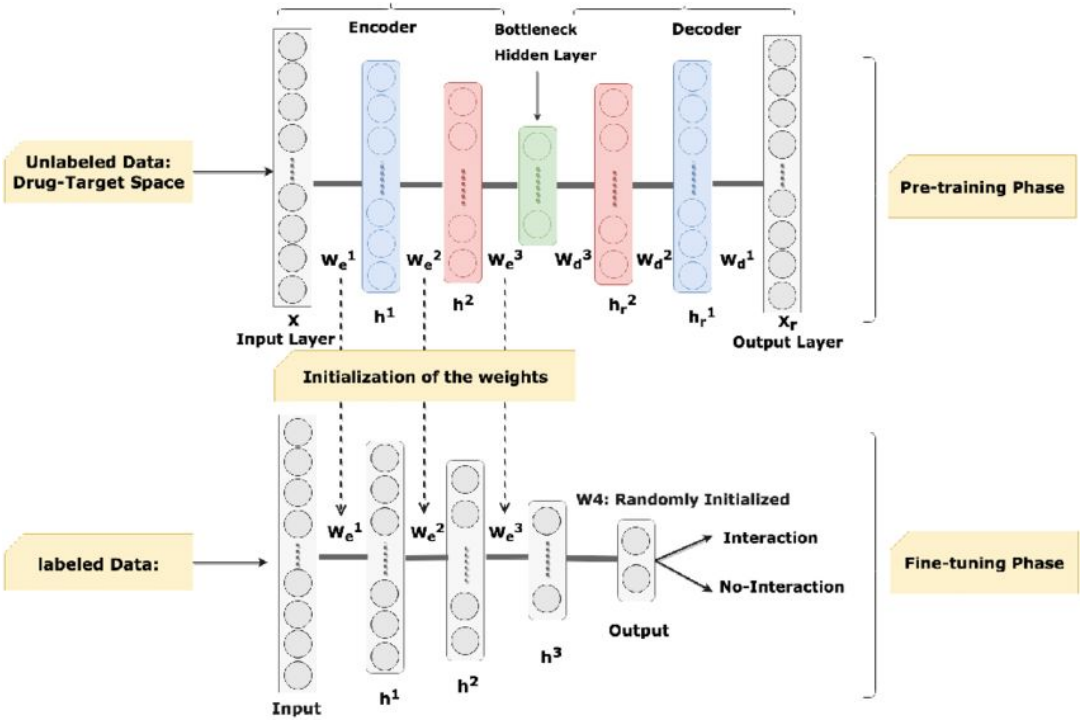
- On réutilise les couches basses d'un réseau déjà entraîné sur un gros jeu de données
- On peut ensuite entraîner les nouvelles couches hautes
- On peut également mettre à jour (fine-tuning) les poids des couches basses

=> Objectif : Entraîner plus rapidement, avec moins de données, et de manière plus efficace

Semi-Supervised pre-training

	X	Y
Labeled	x_1	y_1
	x_2	y_2
	x_3	y_3
	...	...
Unlabeled	x_{5000}	
	x_{5001}	
	x_{5002}	
	x_{5003}	
	...	
	x_{500000}	

Motivation : Utiliser tout le jeu de données pour apprendre les couches cachées



Fin du chapitre 5.2