SCALIAN ACADEMY

Cultivons nos talents
**Scalian Academy**

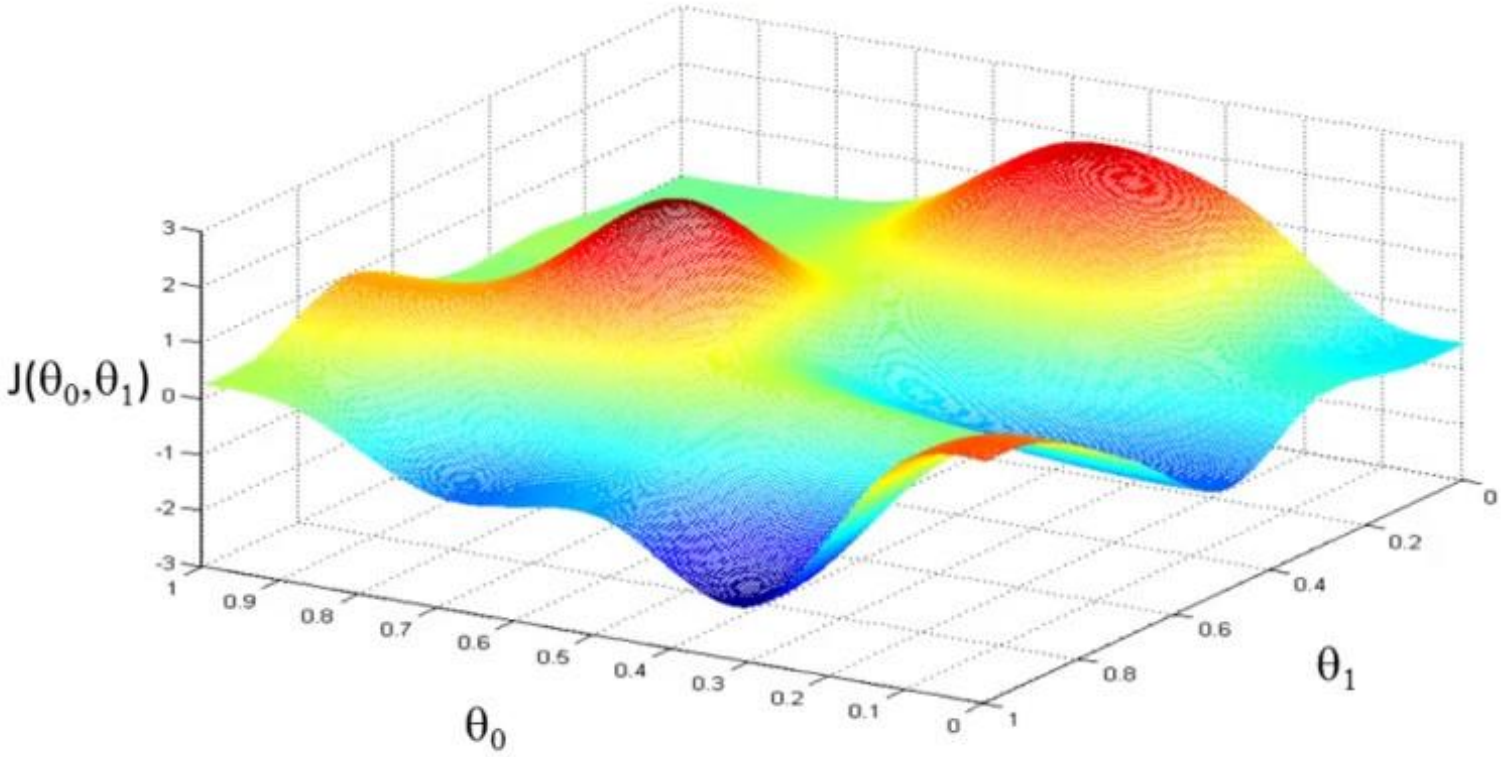
# FORMATION DEEP LEARNING

Entrainer son réseau de DL

# L’INITIALISATION

# COST FUNCTION

SCALIAN



# COST FUNCTION



$$J(\theta) = 1/N \sum_{n=1}^{N} (\hat{y}(x_n, \theta) - y(x_n))^2$$

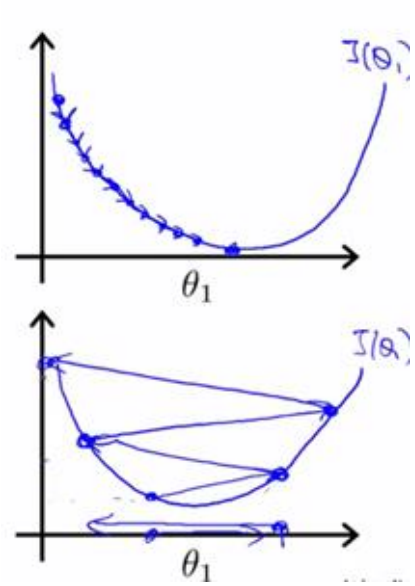
**=> Objectif : Minimiser la Cost function**

# DESCENTE DE GRADIENTS

SCALIAN

$$J(\theta) = 1/N \sum_{n=1}^{N} (\hat{y}(x_n, \theta) - y(x_n))^2$$

Dérivée partielle : $$\frac{\partial J(\theta)}{\partial \theta_1} = 1/N \sum_{n=1}^{N} (\hat{y}(x_n, \theta) - y(x_n)) x_n$$




Pour i allant de 1 à nombre_choisi :

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

Avec $\alpha > 0$, le taux d'apprentissage

# INITIALISATION DES POIDS

SCALIAN

- L'apprentissage des poids :
  - Nouveau_poids = ancien_poids + **petit changement** (dépendant du taux d'apprentissage et **de combien bouger le poids changera l'erreur**)
  - On **n'**initialise **pas** les poids à la même valeur (en particulier à zéro) !
    - S'ils sont tous identiques, et multipliés par la même valeur, ils vont rester identiques
    - Leur faire prendre des valeurs différentes les uns des autres à l'initialisation aide à leur faire apprendre des informations différentes

- Beaucoup d'époque * beaucoup de données = beaucoup de steps = beaucoup de modification des poids
  - S'ils sont trop élevés et modifié trop fortement, **risque d'explosion**

Par conséquence :

- Initialisation à des valeurs **faibles, proches ou autours de zero, et aléatoires.**
- Premiers essais : tirer les poids suivant une distribution normale ou uniforme
- Distributions spécifiques (**xaviers, he for ReLu**,…) : basés sur distributions normales ou uniforme + post-traitement (normalisations par des formules en fonction du nombre de poids, ….)
- Les biais peuvent être initialisées à 0, ou à des valeurs faibles comme les poids.

# VOCABULAIRE

SCALIAN

**Epoque (epoch)** : Un cycle complet, où la totalité du jeu de données est présenté une fois pour l'entrainement du réseau

- Un réseau de neurone est en général entrainé sur un nombre important d'époques
- A chaque époque, l'ordre de présentation des échantillons du jeu de donnée est généralement différent

**Batch et mini-batch** : **Termes qui prêtent à confusion**

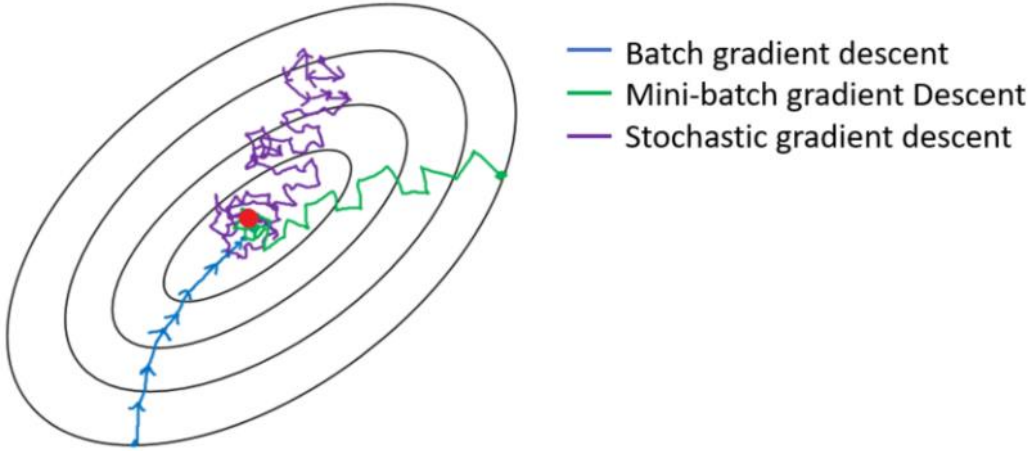
- Définition classique:
  - **Batch** = toutes les données
  - **Mini-batch** = sous-ensemble du jeu de donnée sur lequel on calcule la loss et update les poids (taille décidée par la RAM)
- Définition « DL récente » (ex : article de YoloV4)
  - **Mini-batch** : présentation d'un petit ensemble de données ( taille décidée par la RAM)
  - **Batch** : sous ensemble du jeu de donnée sur lesquel on calcule la loss et update les poids.

**Itération ou step** : La présentation d'un mini-batch

# BATCHS ET APPRENTISSAGE

## « DÉFINITION CLASSIQUE »

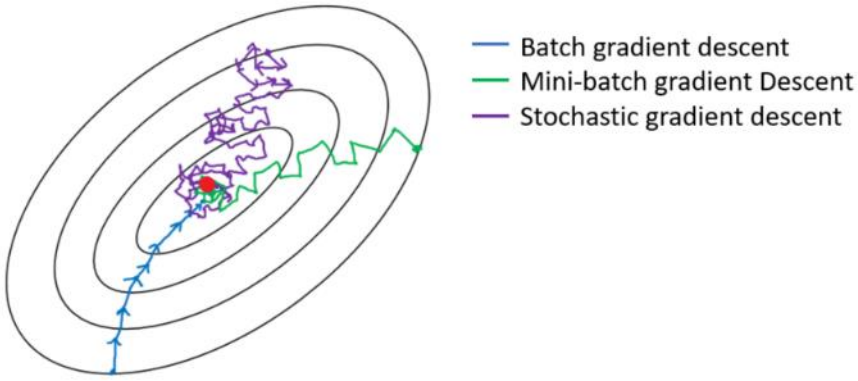
| | Stochastic (minibatch_size =1) | Batch | Mini-Batch |
|---|---|---|---|
| Calcul de l'erreur | Pour chaque exemple | Pour tous les exemples | Pour chaque mini-batch |
| Mise à jour du modèle | Pour chaque exemple | Après évaluation de l'ensemble des données | Pour chaque mini-batch |



# BATCHS ET APPRENTISSAGE

## « DÉFINITION CLASSIQUE »

| | Stochastic (minibatch_size =1) | 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 |




Batch : efficace en nombre de batch, mais présentation d'un batch très longue, et pas toujours possible en RAM

Stochastic : caractère très aléatoire, suivant l'ordre dans lequel les exemples sont présentés. Lente en fonction du nombre de mini-batch.

**Mini-Batch : Bon compromis**

# LES OPTIMIZERS

SCALIAN

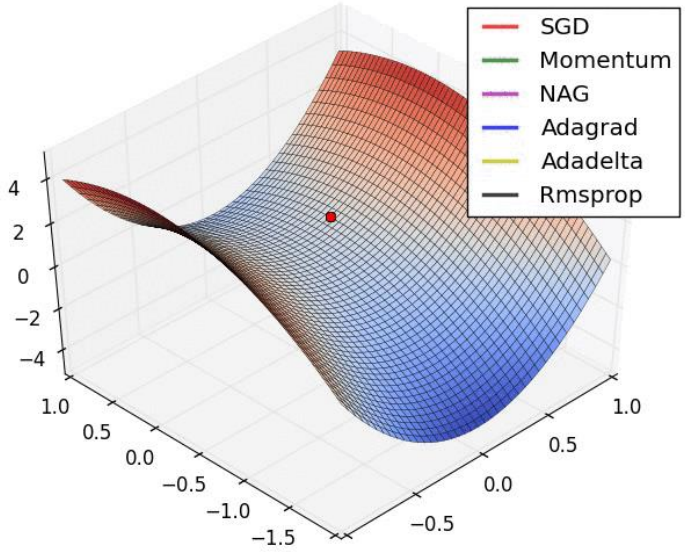
**Différents optimizers :**

- **SGD : basique**
- **Momentum, Nesterov accelerated gradient (Nag) : prennent en compte les dernières variations pour optimiser l'apprentissage**
- **Adagrad, Adadelta, RMSprop : prennent en compte la fréquence d'utilisation des poids dans leur variation**
- **Adam, AdaMax, Nadam : à la fois optimisation similaire à Momentum et à Adagrad**

**Dans beaucoup de modèles state-of-the-art, utilisation d'Adam ou de SGD avec « annealing » (repli simulé : diminution progressive de l'exploration des poids).**

**Compromis entre coût et performance**

- **Majorité du temps d'entraînement passée dans la phase de calcul des nouveaux poids**




Source : http://ruder.io/optimizing-gradient-descent

# ADAM OPTIMIZER

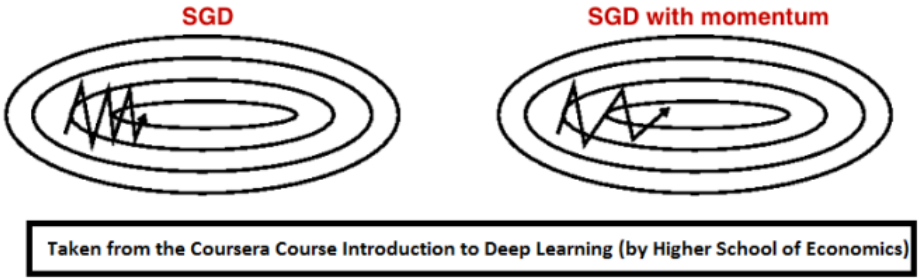
SCALIAN

**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 l’utilisation du poids (historique du gradient)

# ENTRAINER ET ÉVALUER SON RÉSEAU

# ENTRAÎNER SON RÉSEAU DE NEURONES

SCALIAN

**Comme un algorithme de ML, mais avec quelques particularités :**

- **Si peu de données, du feature engineering donnera de meilleurs résultats qu'un algorithme de type deep learning.**

- **Beaucoup de paramètres (architecture/apprentissage/…)**
  - **On ne peut généralement pas faire de méthode automatique sur tous ces paramètres**
  - **On se limite souvent au taux d'apprentissage (voir pas du tout).**
  - **Ajouter un « decay » (décroissance) au taux d'apprentissage améliore presque toujours les performances**

- **L'apprentissage peux être très long**
  - **Certains réseaux mettent des semaines à apprendre sur plusieurs GPU**
  - **Souvent trop long pour faire de la validation croisée.**

- **Il est difficile de « comprendre » ce qui se passe à l'intérieur du modèle**
  - **Intérêt de la visualisation**
  - **Intérêt d'afficher de nombreuses informations sur l'état du réseau en temps réel au cours de l'entraînement**

- **Mettre en concurrence différentes architectures.**

# LES PROBLEMES D'ENTRAINEMENTS

# LES PROBLÈMES « CLASSIQUES » D'UN RÉSEAU DE NEURONE

SCALIAN

**1.(bis) Mon réseau ne converge pas !**

**Problème : La sortie du réseau explose (Nan)**

**Causes potentielles :**

- **Les poids ou les biais initiaux sont trop élevés**
- **Le taux d'apprentissage est trop fort**
  - **Le baisser ou augmenter son decay**
- **Changer de fonction d'activation si le réseau le permet, ajouter du gradient cliping**

# LES PROBLÈMES « CLASSIQUES » D'UN RÉSEAU DE NEURONE

SCALIAN

1. **Mon réseau ne converge pas !**

**Problème : l'accuracy reste aux alentours du niveau de chance sur les jeux d'entraînement et de validation**

**Causes potentielles :**

- **Les données et les labels sont mal alignés et ne correspondent pas.**
- **Les données sont mal fournies au réseau**
- **L'apprentissage ne fonctionne pas**
  - **Les couches de sorties ne sont pas touchées par l'optimizer**
  - **Les poids des couches ont été initialisés à 0**
  - **Temps d'apprentissage trop court**
  - **Taux d'apprentissage mal réglé (trop fort ou trop faible)**
  - **Optimizer pas assez performant**
  - **Disparition du gradient de l'erreur (gradient vanishing)**
  - **« Explosion » du gradient de l'erreur (gradient exploding)**

# 1: L'ENTRAINEMENT NE CONVERGE PAS !

SCALIAN

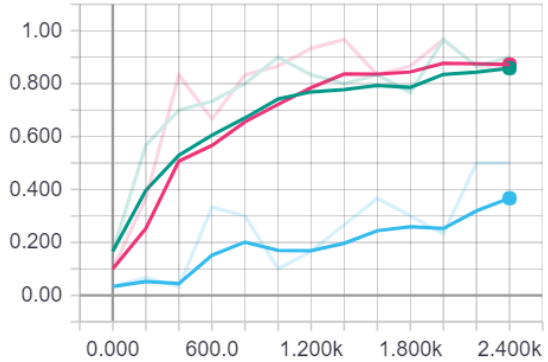
## Gradient Vanishing

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

Effet constaté : ma performance reste faible sur l'entrainement et la validation

Solution : ne pas utiliser la Sigmoid ou la Tanh comme fonction d'activation mais privilégier la ReLu, réduire le nombre de couche, augmenter le taux d'apprentissage, ...



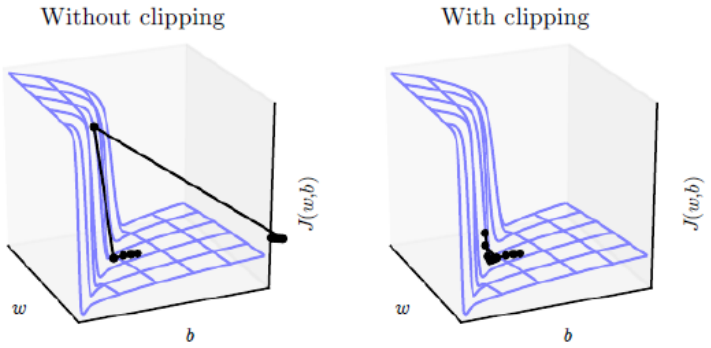

## Gradient Exploding

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

Effet constaté : la sortie du réseau est NAN ou Inf ou 1*e(exposant très grand)

Solutions : utiliser le Gradient Clipping, faible taux d'apprentissage et valeurs initiales des poids faibles

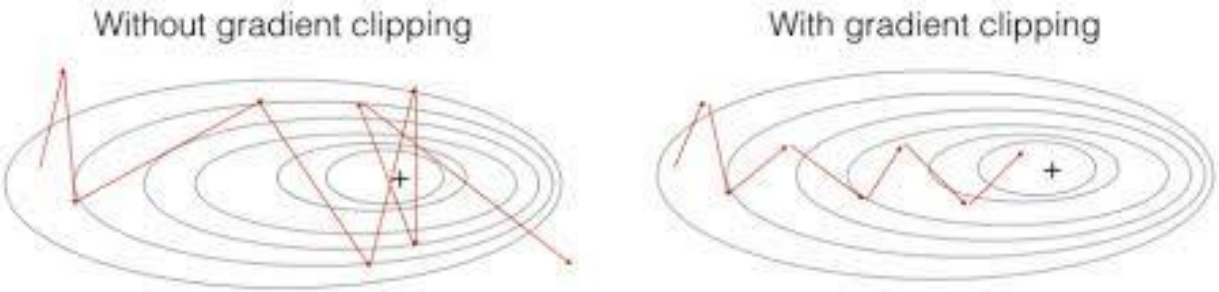


# GRADIENT CLIPPING

SCALIAN

Motivation : Eviter ‘’ l’explosion ’’ des gradients de l’erreur

Principe : borner les valeurs possibles : on définit un intervalle (min,max) tel que la valeur des gadients ne peux pas dépasser cet intervalle.



# 2 : LA VALIDATION NE SUIT PAS

**Mon réseau a de bonnes performances sur le jeu d'entraînement mais pas sur le jeu de validation**

**Causes potentielles :**

- **Différences entre le jeu de validation et le jeu d'entrainement**
  - **Jeu de validation différent/plus complexe/mauvaise séparation**
  - **Le ratio d'exemples de chaque type est différent dans les jeux d'entraînement et validation**
    - **Modifier les proportions du jeu de d'entraînement (ajout de donnée ou sampling)**

- **Le réseau fait du sur-apprentissage (Overfitting) du jeu d'entraînement**
  - **Ajouter de la variation aléatoire dans les données à chaque présentation au réseau (« data augmentation »).**
  - **Arrêter l'entraînement plus tôt quand les performances divergent (« early stopping »)**

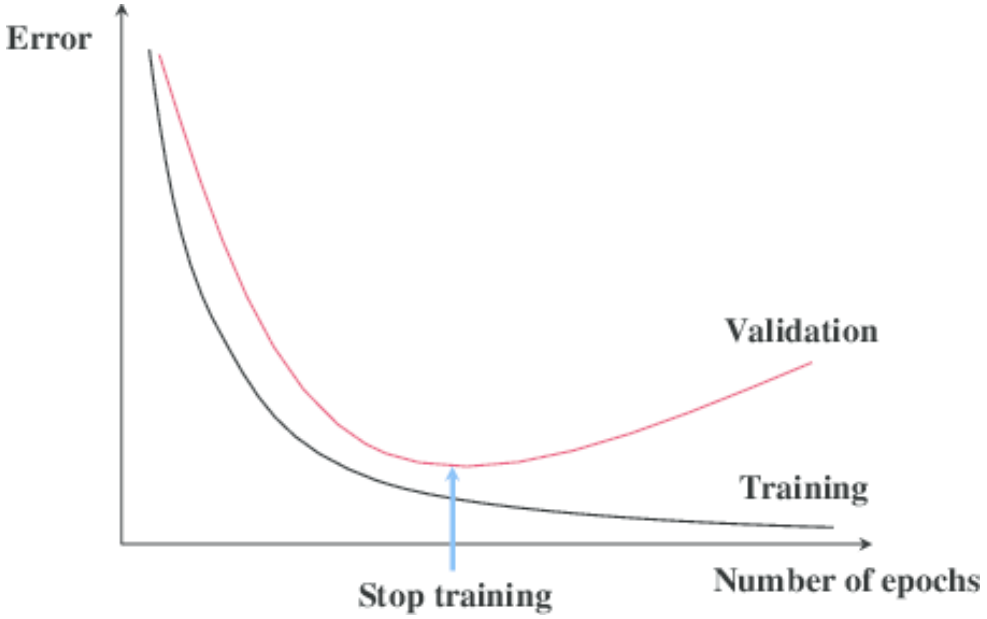
# EARLY STOPPING

SCALIAN

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



# 2 : LA VALIDATION NE SUIT PAS

Mon réseau a de bonnes performances sur le jeu d'entraînement mais pas sur le jeu de validation

Causes potentielles :

- Différences entre le jeu de validation et le jeu d'entrainement
  - Jeu de validation différent/plus complexe/mauvaise séparation
  - Le ratio d'exemples de chaque type est différent dans les jeux d'entraînement et validation
    - Modifier les proportions du jeu de d'entraînement (ajout de donnée ou sampling)

- Le réseau fait du sur-apprentissage (Overfitting) du jeu d'entraînement
  - Ajouter de la variation aléatoire dans les données à chaque présentation au réseau (« data augmentation »).
  - Arrêter l'entraînement plus tôt quand les performances divergent (« early stopping »)
  - **Modifier l'architecture :**
    - **Diminuer la capacité**
    - **Régularisation, batch normalisation, dropout**

# BATCH NORMALISATION

SCALIAN

Batch normalisation interne au réseau

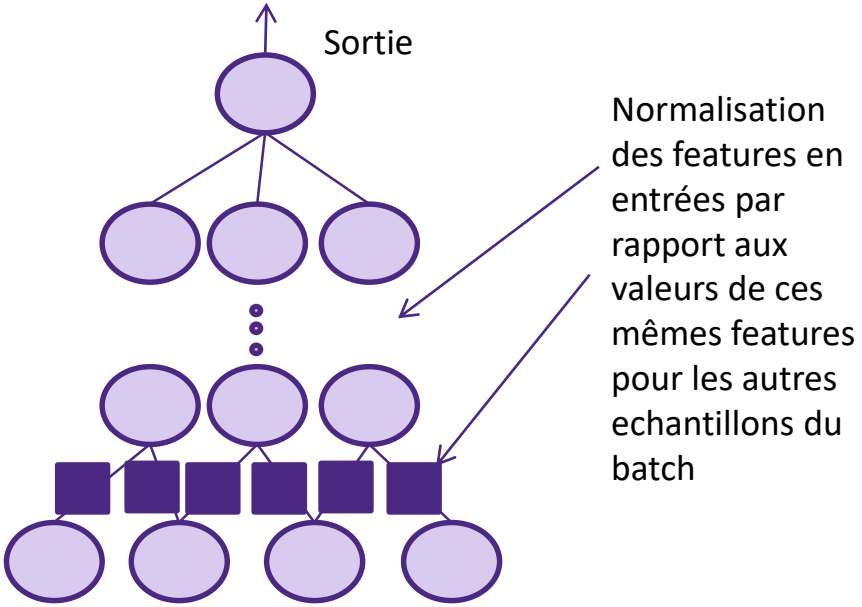
Peut être placée à de multiples endroits dans le réseau

Principe : on normalise chaque feature aux entrées de chaque couche pour chaque mini-batch

**Empiriquement** fonctionne bien, la raison exacte du pourquoi reste discutée...

Si la taille de mini-batch est trop petite, peut ne pas aider.

- Des réseaux récents (ex:Yolov4) prennent dans ce cas les valeurs de plusieurs mini-batchs pour s'assurer d'avoir assez d'éléments pour qu'elle fonctionne.



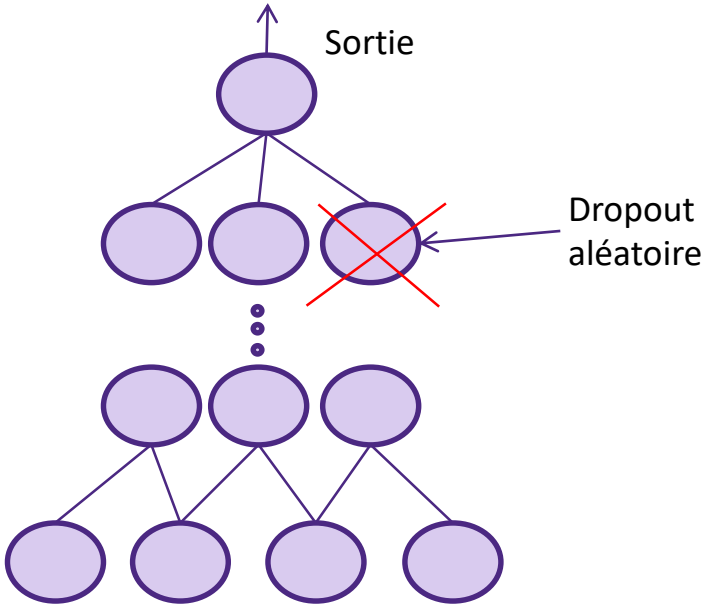
# LE DROPOUT

SCALIAN

**Permet d'éviter l'overfitting et d'améliorer la performance du réseau**

**Idée : ignorer des neurones aléatoirement à chaque step pour forcer le modèle à prêter attention à tous les neurones**

- **Rajoute de l'aléatoire et permet d'éviter l'overfitting**
- **Permet de prêter attention à des features qui seraient autrement ignorées**
- **Appliquer le dropout de préférence dans les couches précédent la couche de sortie**
- **Désactiver le dropout pour la validation**

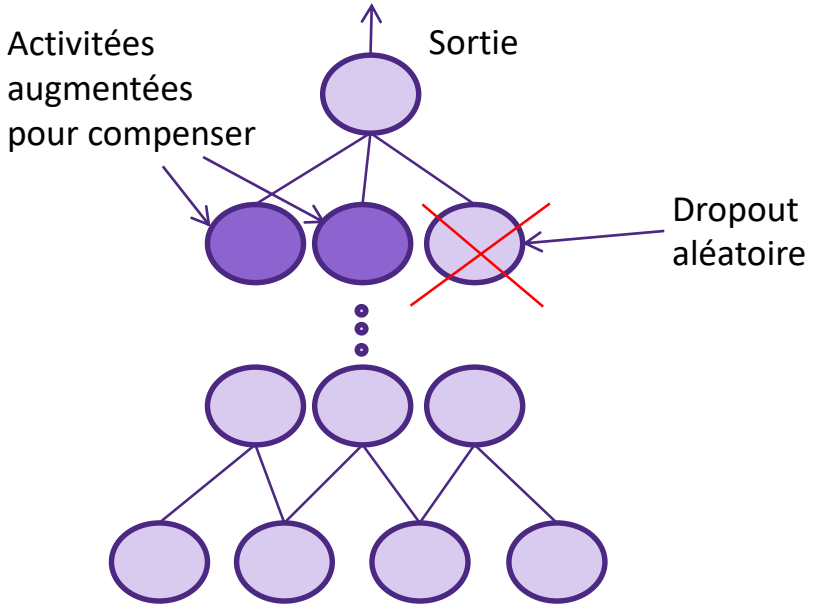


# LE DROPOUT

**Permet d'éviter l'overfitting et d'améliorer la performance du réseau**

**Idée : ignorer des neurones aléatoirement à chaque step pour forcer le modèle à prêter attention à tous les neurones**

- **Rajoute de l'aléatoire et permet d'éviter l'overfitting**
- **Permet de prêter attention à des features qui seraient autrement ignorées**
- **Appliquer le dropout de préférence dans les couches précédent la couche de sortie**
- **Désactiver le dropout pour la validation**



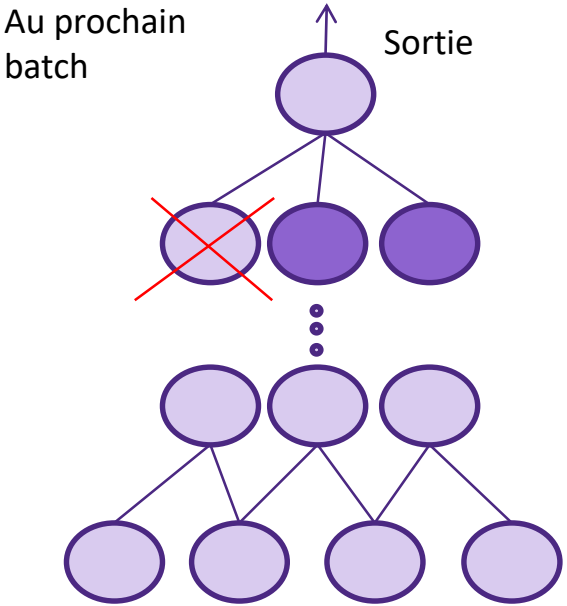
# LE DROPOUT



**Permet d'éviter l'overfitting et d'améliorer la performance du réseau**

**Idée : ignorer des neurones aléatoirement à chaque step pour forcer le modèle à prêter attention à tous les neurones**

- **Rajoute de l'aléatoire et permet d'éviter l'overfitting**
- **Permet de prêter attention à des features qui seraient autrement ignorées**
- **Appliquer le dropout de préférence dans les couches précédent la couche de sortie**
- **Désactiver le dropout pour la validation**

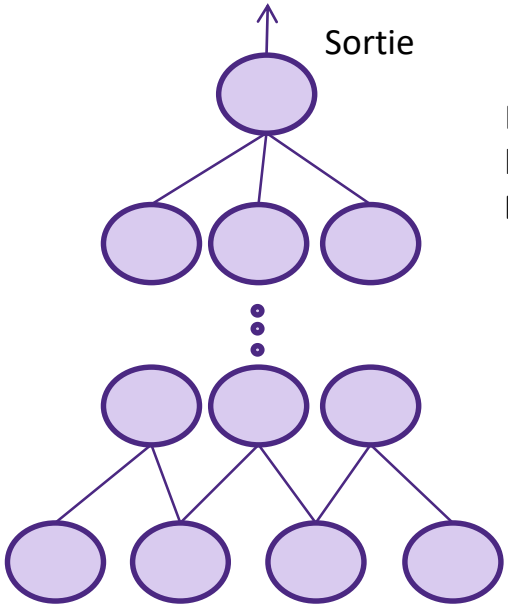


# LE DROPOUT

**Permet d'éviter l'overfitting et d'améliorer la performance du réseau**

**Idée : ignorer des neurones aléatoirement à chaque step pour forcer le modèle à prêter attention à tous les neurones**

- **Rajoute de l'aléatoire et permet d'éviter l'overfitting**
- **Permet de prêter attention à des features qui seraient autrement ignorées**
- **Appliquer le dropout de préférence dans les couches précédent la couche de sortie**
- **Désactiver le dropout pour la validation**




En dehors de l'entrainement : **Dropout désactivé**

# RÉGULARISATION

SCALIAN

⇒ **Objectif :** Prévenir l'overfitting **en diminuant l'utilisation de la capacité disponible**

- Utilisée dans beaucoup d'algorithme de ML de façon différente
- **En deep learning : Sous forme de contrainte supplémentaire ajoutée à la fonction de loss**
  - **L'apprentissage est encouragé à garder la valeur d'un maximum de poids proche de 0.**

**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 :**

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