

C5. Apprentissage des réseaux profonds

Advanced Machine Learning (MLA)

Kévin Bailly

kevin.bailly@sorbonne-universite.fr

<http://people.isir.upmc.fr/bailly/>

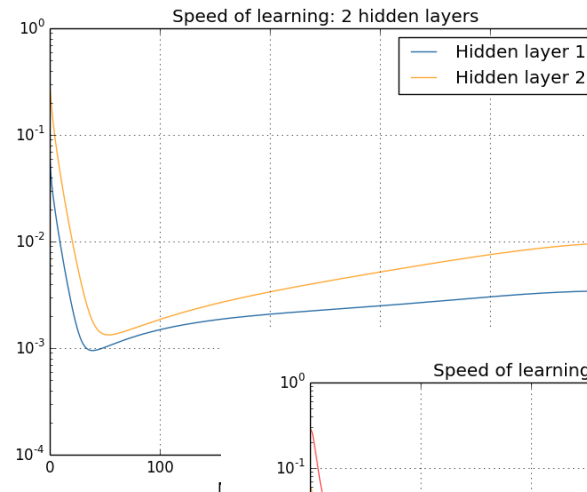
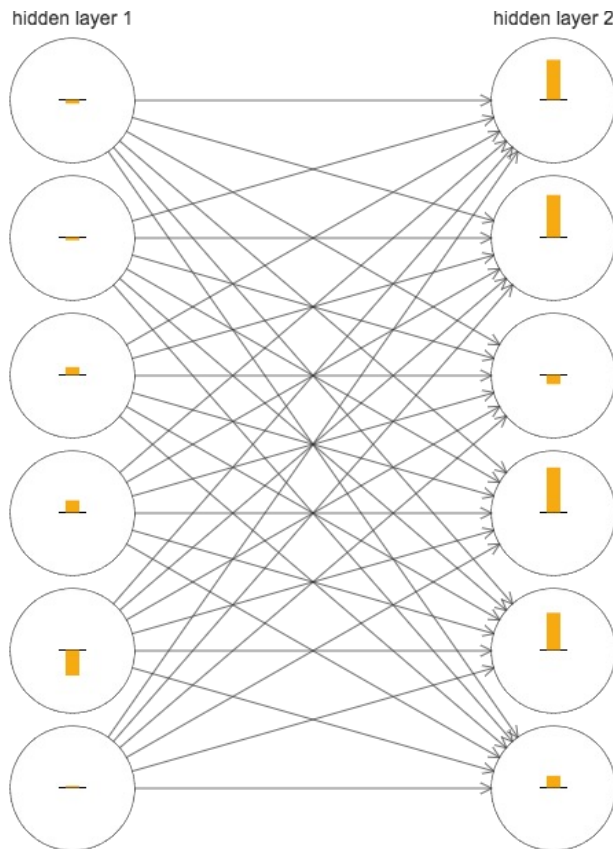
- Limites des modèles MLP classiques
- Définition de l'architecture
 - La fonction de coût entropie croisée
 - fonctions d'activation
 - Initialisation des paramètres (poids)
- L'optimisation des paramètres
 - Limites de la descente de gradient
 - Momentum
 - Adaptation du pas d'apprentissage
- Régularisation

Limites des modèles MLP classiques

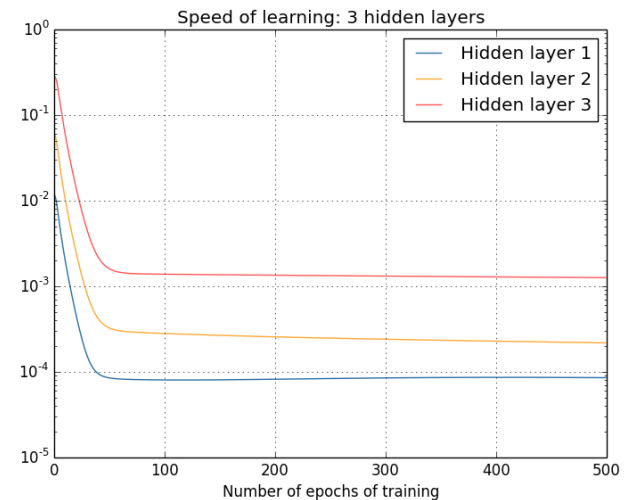
- Pour aborder des **problèmes plus complexes**, il faut des **réseaux plus profonds**
- 2 difficultés principales
 - Risque de sur-apprentissage : il faut de grandes bases de données
 - Risque de sous-apprentissage : dispersion du gradient & saturation des sigmoïdes

Dispersion du gradient

- Les dernières couches apprennent bien plus vite que les premières



Pourquoi ?

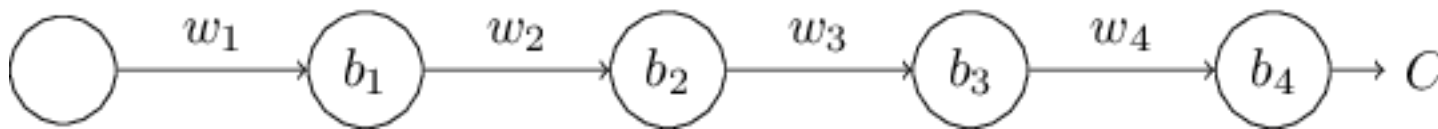


<http://neuralnetworksanddeeplearning.com/chap5.html>

Dispersion du gradient

- Dispersion du gradient :

Chaque passage par une sigmoïde **diminue l'amplitude du gradient** par un facteur 4 ou plus (max. à 0,25)

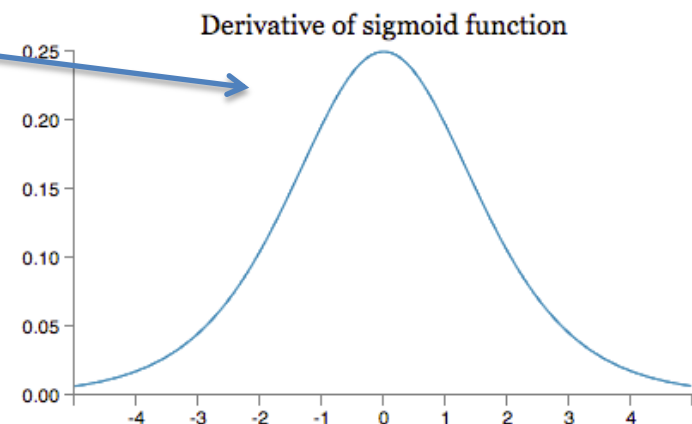


$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \boxed{w_2 \sigma'(z_2)} \boxed{w_3 \sigma'(z_3)} w_4 \sigma'(z_4) \frac{\partial C}{\partial a_4}$$

< 0,25 < 0,25 ...

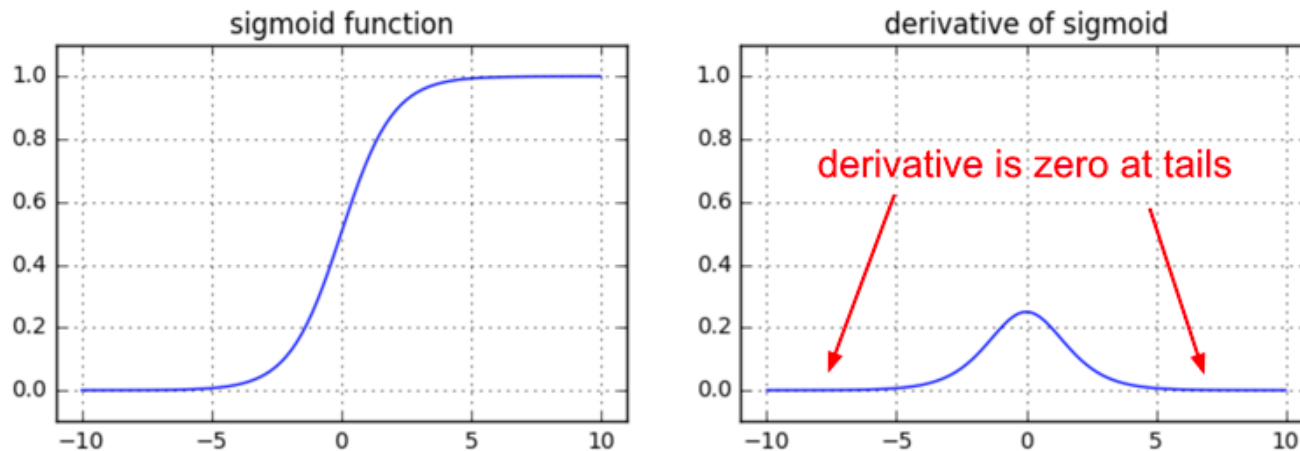
Est-ce qu'il suffit d'augmenter les valeurs des w ?

<http://neuralnetworksanddeeplearning.com/chap5.html>



Limites des modèles MLP classiques

- Que se passe-t-il si les poids du neurone sont très grands ?
 - Les activations des sigmoïdes sont presque binaires (0 ou 1)
 - le gradient devient nul (on ne rétro-propage plus rien)



Dispersion + explosion → Gradient instable

<https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b>

<https://www.youtube.com/watch?v=gYpoJMIgyXA&feature=youtu.be&t=14m14s>

Les méthodes modernes

- Quelles avancées depuis les années 80
 - Même algorithme de rétro-propagation
 - Même(s) méthode(s) de descente de gradient
- Mais :
 - Nouvelles architectures (DAG, fonctions de coût intermédiaires...)
 - Fonction de coût : entropie croisée
 - Combinée avec la fonction d'activation softmax
 - Fonction d'activation : ReLU
 - + performante pour les grandes bases de données
 - + proche du comportement des neurones réels (activation parcimonieuse et linéaire)
 - Régularisation / normalisations / stratégies de descente
 - Pour aider le réseau à apprendre

- Limites des modèles MLP classiques
- Définition de l'architecture
 - La fonction de coût entropie croisée
 - fonctions d'activation
 - Initialisation des paramètres (poids)
- L'optimisation des paramètres
 - Limites de la descente de gradient
 - Momentum
 - Adaptation du pas d'apprentissage
- Régularisation

- Limites des modèles MLP classiques
- Définition de l'architecture
 - La fonction de coût entropie croisée
 - fonctions d'activation
 - Initialisation des paramètres (poids)
- L'optimisation des paramètres
 - Limites de la descente de gradient
 - Momentum
 - Adaptation du pas d'apprentissage
- Régularisation

Entropie croisée

- Permet de mesurer la ressemblance entre deux distributions de probabilité
 - Utilisée pour mesurer la ressemblance en la **prédiction** et la **vérité terrain**

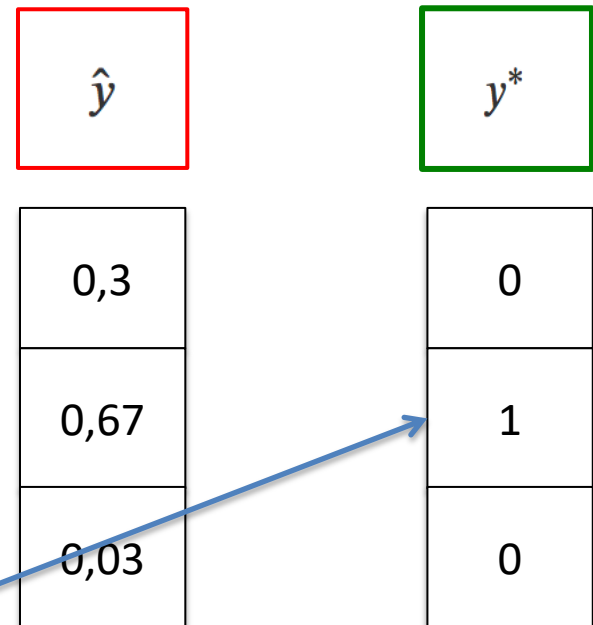
Non symétrique

$$l_{CE}(\hat{y}, y^*) = - \sum_{i=1}^{N_{\text{classes}}} \boxed{y_i^* \log \hat{y}_i}$$

Que se passe-t-il si les sorties du réseau ne sont pas des probabilités ?

Comment être sûr qu'elles le soient ?

« One hot encoding »

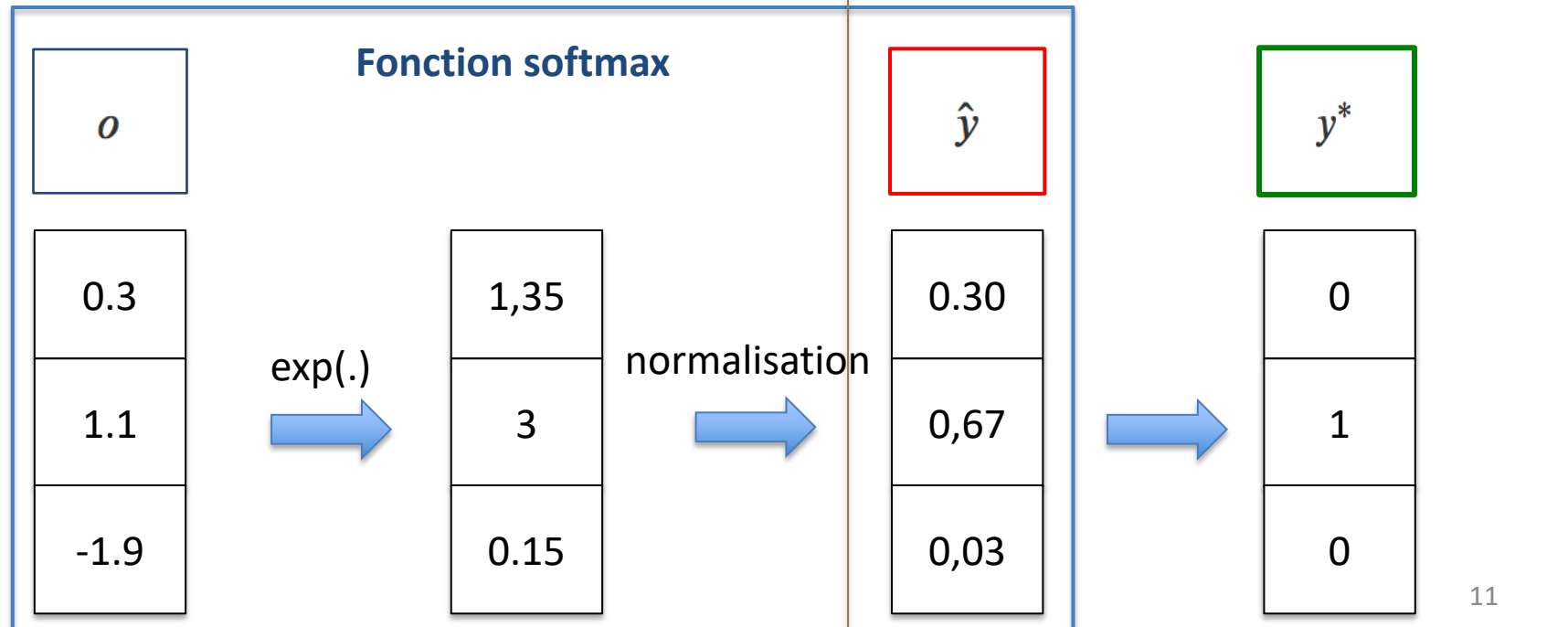


Entropie croisée

- La fonction d'activation **softmax**
- La combinaison [softmax + entropie croisée] donne généralement de meilleurs résultats que [sigmoïde + L2]

Plus d'infos : **Deep learning book**

Chapitre 6.2.2 : Lien entre fonction de coût et fonctions d'activation en sortie

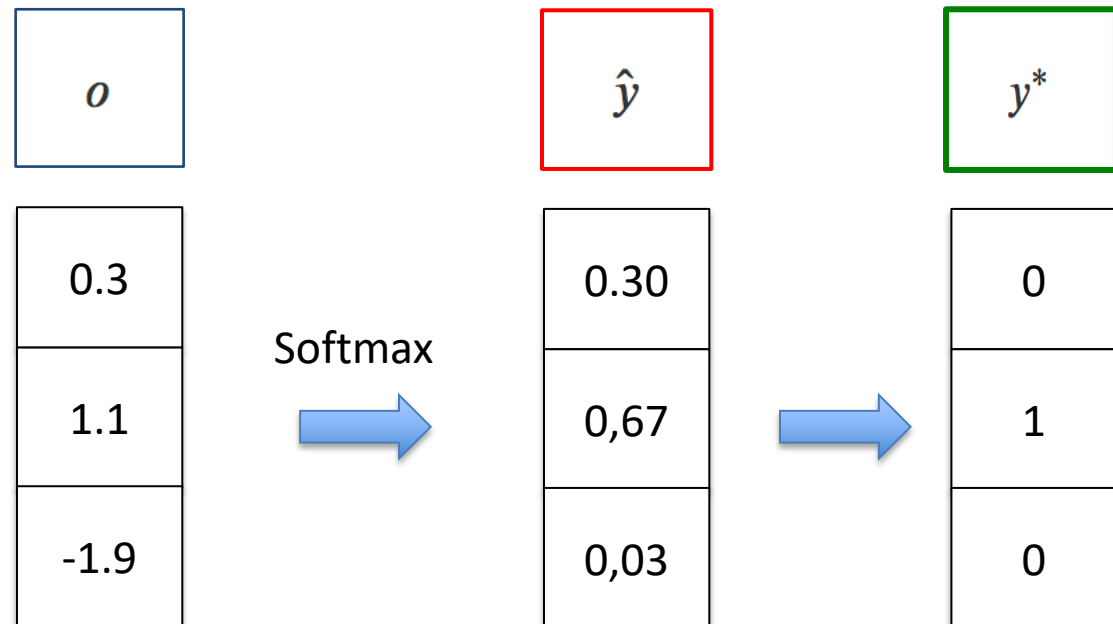


Entropie croisée

- Exercice : calculer le gradient de la fonction de coût entropie croisée par rapport à o

$$y_j = S(o_j) = \frac{e^{o_j}}{\sum_k e^{o_k}}$$

$$l_{CE}(\hat{y}, y^*) = - \sum_{i=1}^{N_{\text{classes}}} y_i^* \log \hat{y}_i$$



- **Question 1 :**

- En vous inspirant de l'exemple suivant :

- https://pytorch.org/tutorials/beginner/basics/optimization_tutorial.html

- Écrivez un code qui :

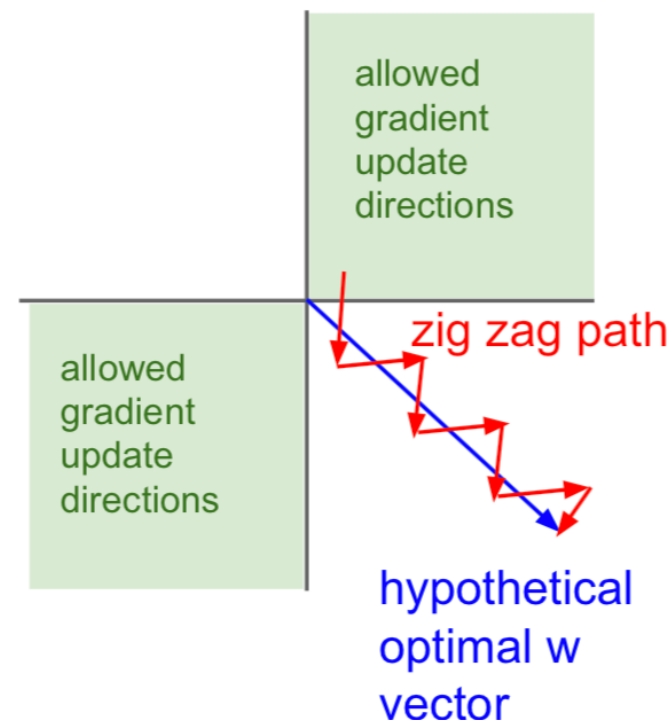
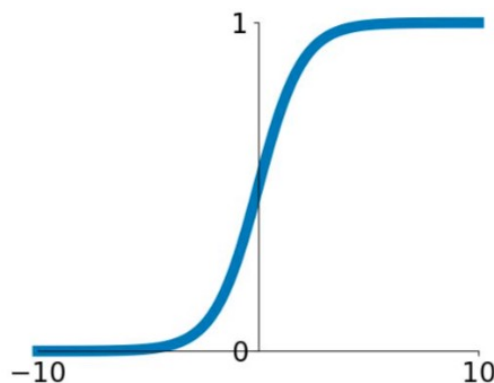
- charge la base de donnée MNIST
 - permet de visualiser les exemples contenus dans un batch
 - définit un réseau MLP à trois couches avec les caractéristiques suivantes :
 - tailles des couches cachées 120 et 80 neurones
 - fonction de coût : entropie croisée
 - fonction d'activation : sigmoïde
 - apprend ce réseau sur la partition d'apprentissage (2 epochs)
 - évalue les performances

- Limites des modèles MLP classiques
- Définition de l'architecture
 - La fonction de coût entropie croisée
 - fonctions d'activation
 - Initialisation des paramètres (poids)
- L'optimisation des paramètres
 - Limites de la descente de gradient
 - Momentum
 - Adaptation du pas d'apprentissage
- Régularisation

Les fonctions d'activations courantes

- La fonction sigmoïde
 - Valeurs dans l'intervalle [0,1]
 - Régime linéaire pour de faibles valeurs de x
 - Bio-inspirée (approximation de la fonction seuil)
 - Saturation et dispersion du gradient
 - Valeurs de sortie non centrées en 0
 - Coût de calcul de la fonction exponentielle

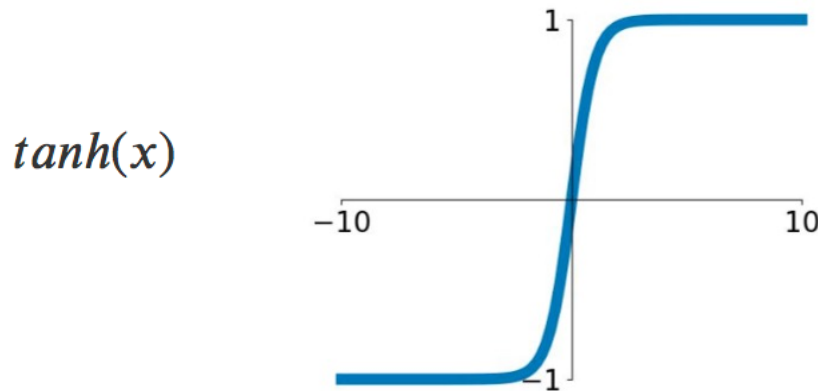
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



<http://cs231n.github.io/neural-networks-1/#actfun>

Les fonctions d'activations courantes

- **La fonction tangente hyperbolique**
 - Valeurs dans l'intervalle $[-1,1]$
 - Centrée en 0
 - Saturation et dispersion du gradient



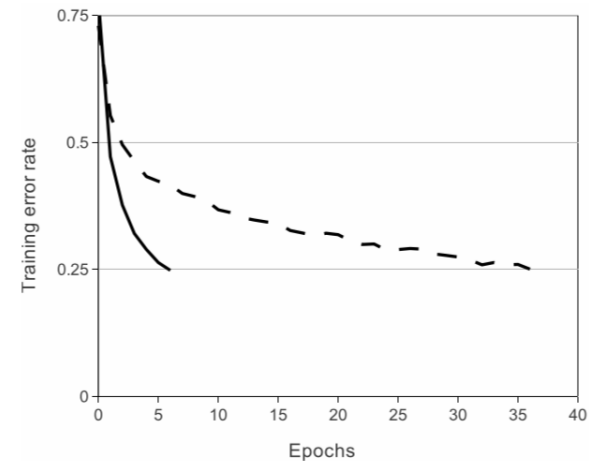
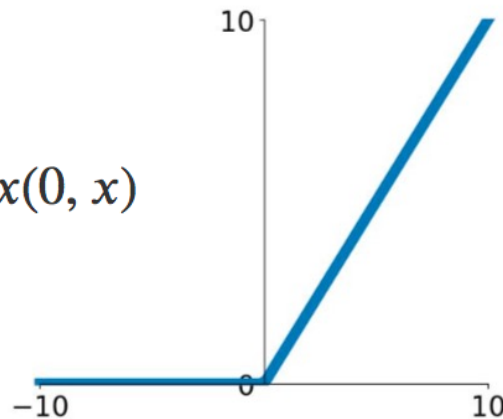
<http://cs231n.github.io/neural-networks-1/#actfun>

Les fonctions d'activations courantes

- **ReLU : Rectified Linear Unit**

- La plus utilisée actuellement
- Accélère la convergence : x6 d'après [Krizhevsky et al., 2012]
- Pas de dispersion du gradient
- Rapide à calculer
- Biologiquement plus plausible que la sigmoïde

$$\text{ReLU}(x) = \max(0, x)$$

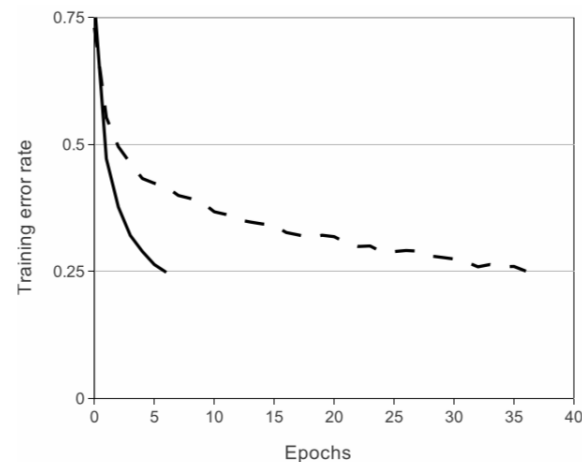
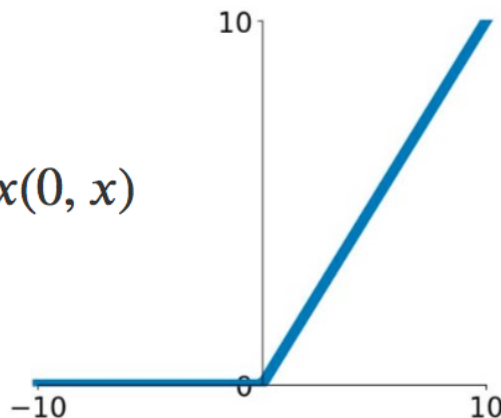


Les fonctions d'activations courantes

- **ReLU : Rectified Linear Unit**

- La plus utilisée actuellement
- Accélère la convergence : x6 d'après [Krizhevsky et al., 2012]
- Pas de dispersion du gradient
- Rapide à calculer
- Biologiquement plus plausible que la sigmoïde

$$\text{ReLU}(x) = \max(0, x)$$



Certains neurones peuvent devenir **inactifs** :

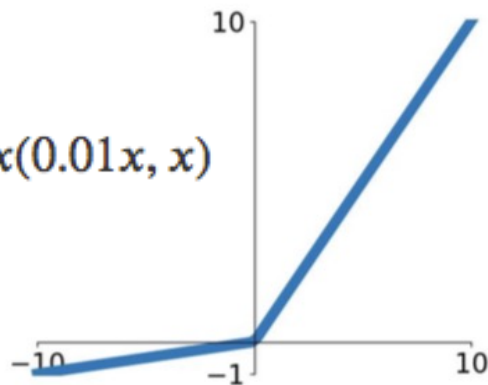
- Attention à l'**initialisation** des poids
- Attention au pas de la descente

Les fonctions d'activations courantes

- **Leaky ReLU**

- Accélère la convergence : x6 d'après [Krizhevsky et al., 2012]
- Pas de dispersion du gradient
- Rapide à calculer
- **Pas de neurones inactifs**

$$LReLU(x) = \max(0.01x, x)$$

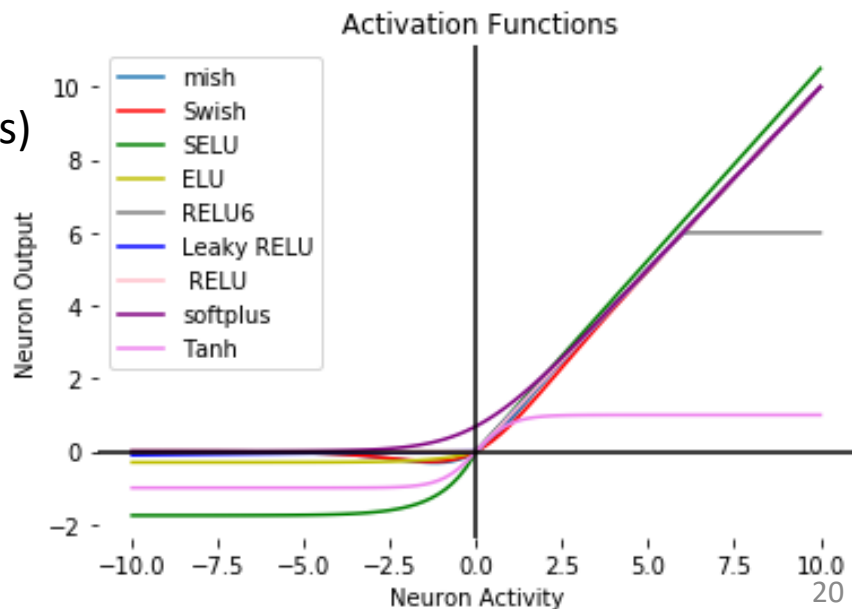


Parametric ReLU : la pente α est un paramètre appris

$$PReLU(x) = \max(\alpha x, x)$$

Les fonctions d'activations courantes

- Beaucoup d'autres fonctions d'activations :
 - ELU, SELU, GELU, Swish...
- En pratique :
 - Utiliser ReLU (attention au pas d'apprentissage)
 - Utiliser certaines fonctions d'activation modernes :
 - Compromis coût / performance (calcul de l'exponentielle)
 - Par défaut dans certaines architectures modernes (eg. GELU dans les Transformers)



- Reprendre le réseau de reconnaissance des chiffres manuscrits (base MNIST) et tester différentes fonctions d'activation.
- Reporter les résultats obtenus

- Limites des modèles MLP classiques
- Définition de l'architecture
 - La fonction de coût entropie croisée
 - fonctions d'activation
 - Initialisation des paramètres (poids)
- L'optimisation des paramètres
 - Limites de la descente de gradient
 - Momentum
 - Adaptation du pas d'apprentissage
- Régularisation

Initialisation des poids

- Le choix de la méthode d'initialisation est très important
 - Point de départ de l'initialisation
- Ce choix dépend :
 - Du type des fonctions d'activation
 - De l'architecture du réseau
 - De la méthode d'optimisation et de régularisation
- Trouver la bonne initialisation est difficile et empirique
 - Mais il existe des bonnes pratiques !

Initialisation des poids

- Que se passe-t-il si les poids de deux neurones connectés à la même entrée sont initialisés avec les mêmes valeurs ?

Initialisation des poids

- Que se passe-t-il si tous les poids sont initialisés avec la même valeur ?
 - Tous les poids évoluent de la même manière
- Comment casser la symétrie ?
 - En les initialisant aléatoirement
 - La plupart du temps, loi normale ou uniforme
- Quelle variance choisir ?

Initialisation des poids

- Que se passe-t-il si on les initialise avec une variance trop faible ?
 - Les activations dans les couches suivantes tendent vers 0
- Que se passe-t-il si on les initialise avec une variance trop grande ?
 - Risque de saturation des sigmoïdes
 - Risque d'instabilité
- Que se passe-t-il si un neurone est connecté à un grand nombre de neurones de la couche précédente ?
 - Risque de saturation
 - Il **faut adapter les poids des connexions** en fonction du nombre de neurones de la couche précédente

Initialisation des poids

- 1^{ère} solution : en fonction du nombre d'entrées n du neurone

$$\mathbf{W}_{i,j} \rightsquigarrow U \left(-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}} \right)$$

- Objectif : même variance d'activation, quelque soit la couche précédente

Initialisation des poids

- 2^{ème} solution : en fonction du nombre d'entrées n et de sortie m du neurone
- Méthode de Xavier, très souvent utilisée [Glorot et Bengio 2010]

$$\mathbf{W}_{i,j} \rightsquigarrow U \left(-\sqrt{\frac{6}{n+m}}, \sqrt{\frac{6}{n+m}} \right)$$

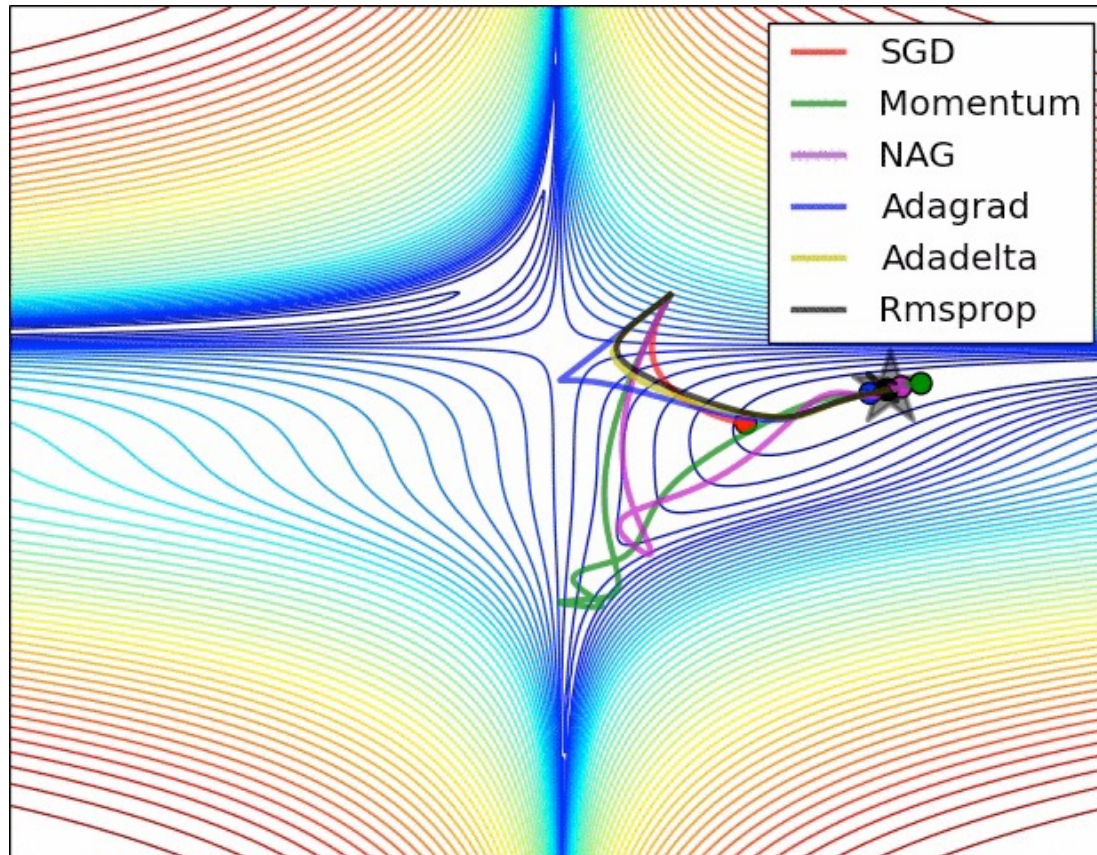
- **Objectif** : même variance d'activation et même variance du gradient, quelque soit la couche (la formule dépend du type de la fonction d'activation)
- Ne prend pas en compte les non-linéarités (mais reste tout de même efficace)

Initialisation des poids

- Il existe de nombreuses autres méthodes d'initialisation des poids :
 - Pas de méthode universelle
 - Stratégies actuelles simples et heuristiques
 - Phénomène encore mal compris (évolution au cours de l'apprentissage, impact sur la généralisation...)
- Seule certitude : il faut casser la symétrie
 - A considérer comme un hyper-paramètre (si les ressources sont disponibles)
- Il faut trouver un compromis entre :
 - Des poids forts pour casser la symétrie et limiter la dispersion du gradient
 - Des poids faibles pour limiter les risques d'explosion du gradient
- Possibilité d'utiliser un réseau pré-appris comme initialisation
 - Apprentissage non supervisé ou pour une autre tâche

- Limites des modèles MLP classiques
- Définition de l'architecture
 - La fonction de coût entropie croisée
 - fonctions d'activation
 - Initialisation des paramètres (poids)
- **L'optimisation des paramètres**
 - Limites de la descente de gradient
 - Momentum
 - Adaptation du pas d'apprentissage
- Régularisation

Méthodes d'optimisation



Credit : Alec Radford

Apprentissage du réseau

- Objectif : optimiser les paramètres du réseau qui minimisent un certain coût (ex: entropie croisée, erreur quadratique)

$$\theta^* = \operatorname{argmin}_{\theta} \mathcal{L}(\hat{y}, y^*) = \sum_{i=1}^{N_{\text{exemples}}} \ell(\hat{y}^{(i)}, y^{*(i)})$$

- Quelle méthode utiliser ?
 - Grid search ?
 - Random search ?
 - Solution analytique ?
 - La descente de gradient

SGD : Stochastic gradient descent

– Quels exemples utilisent-on pour estimer le gradient ?

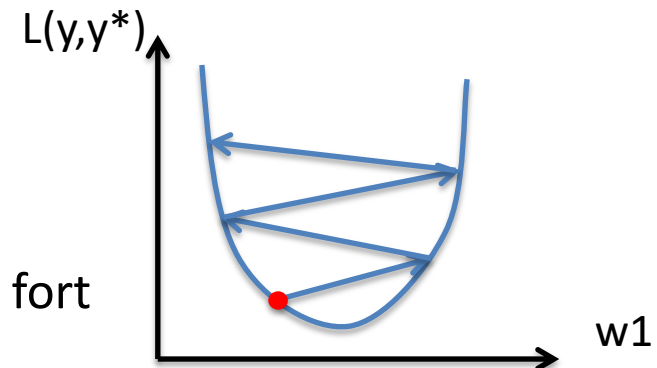
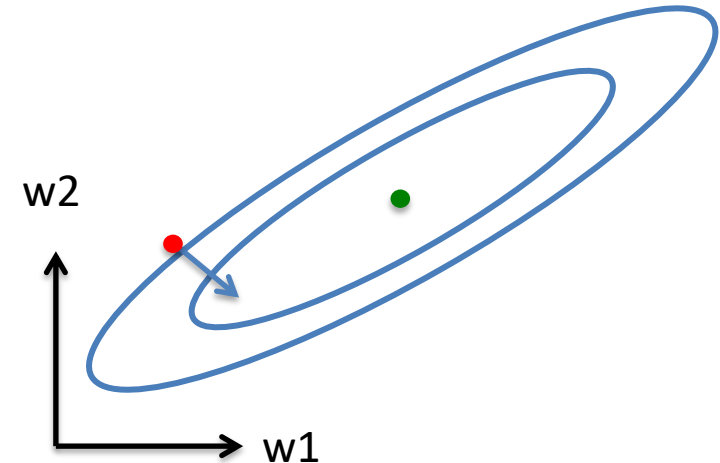
- **Tous les exemples d'apprentissage** : descente de gradient classique (ou *batch gradient descent*)
- **Un petit sous ensemble** (généralement un puissance de 2) : descente de gradient stochastique par mini-batch (SGD)
- **Un seul exemple** : descente de gradient stochastique online.

– Quel impact sur la descente ?

- Avec peu d'exemples : estimation bruitée mais beaucoup plus rapide à calculer
- Trouver un compromis
- Dépend des autres traitement effectués (par exemple, BatchNorm très sensible à la taille des batches)

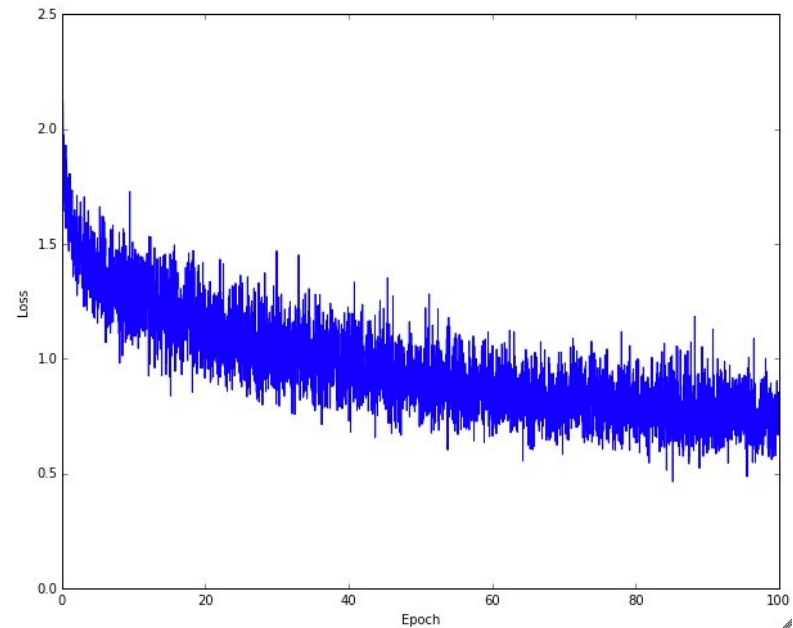
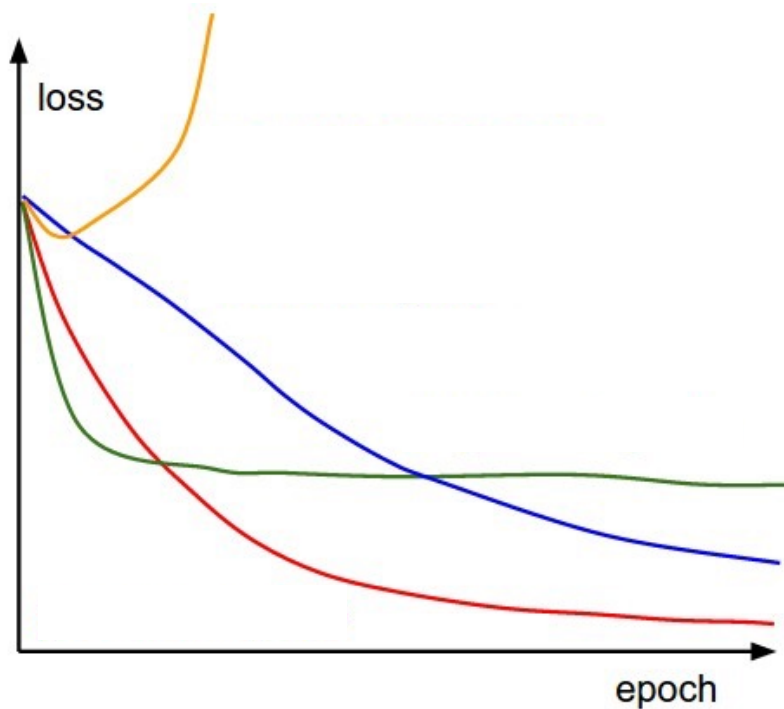
Limites de la descente de gradient

- Les minima locaux et les points selles
 - Gradients nuls
 - **Très fréquent en grande dimension**
- La direction du gradient n'est pas optimale, sauf si l'ellipse est un cercle
- La fonction de coût est fortement non linéaire
 - Le choix du pas est important
 - Il faut avancer rapidement lorsque le gradient est faible et stable
 - Il faut avancer lentement lorsque il est fort et instable



Limites de la descente de gradient

- Observer la courbe d'erreur pour régler le pas d'apprentissage



Source : cs231

Limites de la descente de gradient

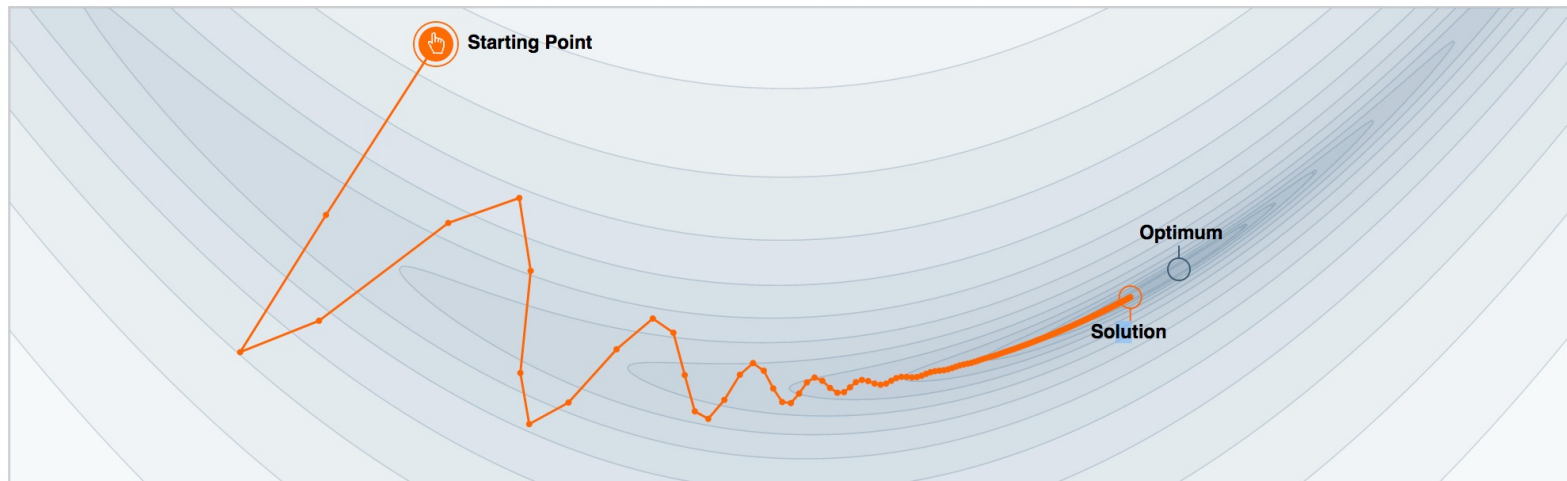
- La plupart du temps, une simple descente de gradient ne suffit pas. Quelle(s) solution(s) ?
 - Momentum
 - Adaptation du pas du gradient
 - Normaliser les données (batch norm)
 - Régularisation (L2, L1, dropout)

- Limites des modèles MLP classiques
- Définition de l'architecture
 - La fonction de coût entropie croisée
 - fonctions d'activation
 - Initialisation des paramètres (poids)
- **L'optimisation des paramètres**
 - Limites de la descente de gradient
 - **Momentum**
 - Adaptation du pas d'apprentissage
- Régularisation



Momentum

- Idée: ne pas tenir compte uniquement du gradient
- Prendre en compte les (quelques) déplacements précédents
- Analogie avec une balle se déplaçant dans une cuvette



Step-size $\alpha = 0.0019$



Momentum $\beta = 0.78$



We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

Momentum

- On introduit une variable v (la vitesse)
- Cette vitesse est une moyenne mobile exponentielle

Pas d'apprentissage (learning rate)

$$v \leftarrow \boxed{\alpha} v - \boxed{\rho} \nabla_{\theta} \mathcal{L}(f(x; \theta), y^*)$$

Pondère la contribution des anciens gradients

- Puis on met à jour les paramètres du réseaux

$$\theta \leftarrow \theta + v$$

- Nesterov momentum [Nesterov, 1983]

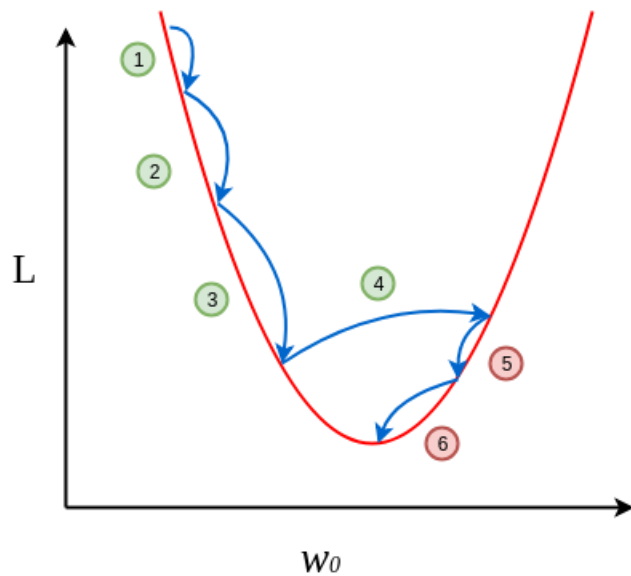
$$v \leftarrow \alpha v - \rho \nabla_{\theta} \mathcal{L}(f(x; \boxed{\theta + \alpha v}), y^*)$$

Donne
généralement de
meilleurs
résultats

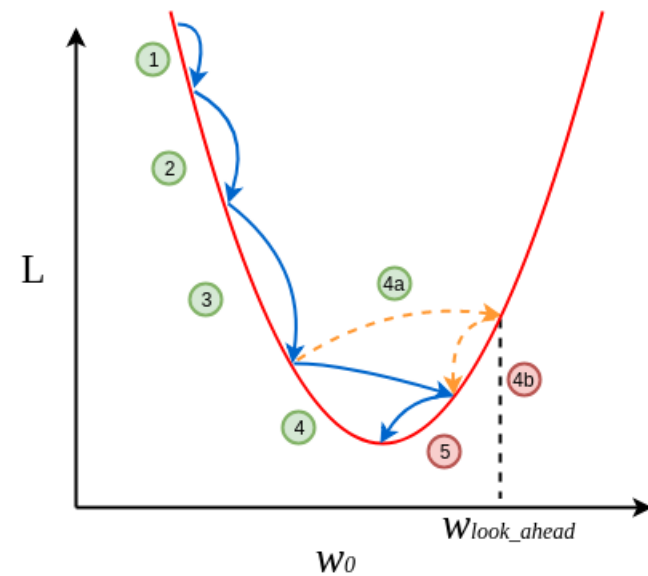
Le gradient n'est pas évalué pour la même position :

- On se déplace d'abord dans la direction du gradient cumulé
- Puis on cherche à corriger la prédiction par rapport au gradient à cette nouvelle position

Nesterov momentum



(a) Momentum-Based Gradient Descent



(b) Nesterov Accelerated Gradient Descent

- Limites des modèles MLP classiques
- Définition de l'architecture
 - La fonction de coût entropie croisée
 - fonctions d'activation
 - Initialisation des paramètres (poids)
- **L'optimisation des paramètres**
 - Limites de la descente de gradient
 - Momentum
 - Adaptation du pas d'apprentissage
- Régularisation

Adaptation du pas d'apprentissage

- AdaGrad [Duchi et al. JMLR 2011]: adapter le pas d'apprentissage pour chaque paramètre du réseau

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ

Require: Initial parameter θ

Require: Small constant δ , perhaps 10^{-7} , for numerical stability

Initialize gradient accumulation variable $\mathbf{r} = \mathbf{0}$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

Accumulate squared gradient: $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g}$

Historique de l'intensité des gradients (par paramètre)

Compute update: $\Delta \theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g}$. (Division and square root applied element-wise)

Normalisation du pas d'apprentissage

Apply update: $\theta \leftarrow \theta + \Delta \theta$

end while

Intérêt ?

Effet à long terme ?

[Deep Learning book]

Adaptation du pas d'apprentissage

- **RMSProp** [Hinton 2012 (non publié)] : amélioration d'AdaGrad pour des fonctions de coût non-convexes

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ , decay rate ρ .

Require: Initial parameter θ

Require: Small constant δ , usually 10^{-6} , used to stabilize division by small numbers.

Initialize accumulation variables $r = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

Accumulate squared gradient: $r \leftarrow \rho r + (1 - \rho) \mathbf{g} \odot \mathbf{g}$

moyenne mobile
exponentielle

Compute parameter update: $\Delta\theta = -\frac{\epsilon}{\sqrt{\delta + r}} \odot \mathbf{g}$. ($\frac{1}{\sqrt{\delta + r}}$ applied element-wise)

Apply update: $\theta \leftarrow \theta + \Delta\theta$

end while

Nouvel hyper-paramètre à déterminer
Généralement: 0.9 ou 0.99

Adaptation du pas d'apprentissage

- **Adam** [Kingma et al., 2014] : adaptive moments

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)

Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$.
(Suggested defaults: 0.9 and 0.999 respectively)

Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})

Require: Initial parameters θ

Initialize 1st and 2nd moment variables $\mathbf{s} = \mathbf{0}$, $\mathbf{r} = \mathbf{0}$

Initialize time step $t = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

$t \leftarrow t + 1$

Update biased first moment estimate: $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1 - \rho_1) \mathbf{g}$

Momentum

Update biased second moment estimate: $\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1 - \rho_2) \mathbf{g} \odot \mathbf{g}$

RMSprop

Correct bias in first moment: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \rho_1^t}$

Correct bias in second moment: $\hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1 - \rho_2^t}$

Compute update: $\Delta \theta = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}} + \delta}}$ (operations applied element-wise)

Apply update: $\theta \leftarrow \theta + \Delta \theta$

end while



Adaptation du pas d'apprentissage

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)

Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$.
(Suggested defaults: 0.9 and 0.999 respectively)

Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})

Require: Initial parameters θ

Initialize 1st and 2nd moment variables $\mathbf{s} = \mathbf{0}, \mathbf{r} = \mathbf{0}$

Initialize time step $t = 0$

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

 Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

$t \leftarrow t + 1$

 Update biased first moment estimate: $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1 - \rho_1) \mathbf{g}$

 Update biased second moment estimate: $\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1 - \rho_2) \mathbf{g} \odot \mathbf{g}$

 Correct bias in first moment: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \rho_1^t}$

 Correct bias in second moment: $\hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1 - \rho_2^t}$

Corrige les biais d'initialisation

 Compute update: $\Delta \theta = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}} + \delta}}$ (operations applied element-wise)

 Apply update: $\theta \leftarrow \theta + \Delta \theta$

end while

Quelle conséquence dans les premières itérations ?

Adaptation du pas d'apprentissage

- Il existe de nombreuses autres méthodes
- Certaines s'appuient sur des approximation d'ordre 2
 - Méthodes de Newton (avec estimation de la hessienne)
 - Gradient conjugué (non linéaire)
 - BFGS

Normalisation par batch (batch norm)

- Normaliser les données améliorent les performances (descripteurs de moyenne nulle et variance égale à 1)
 - Les plages de variations entre descripteurs sont plus homogènes
 - Evite les saturations
- **Idée : appliquer le même principe aux activations de chaque couche**
 - **(Beaucoup) plus de stabilité en apprentissage**

S. Ioffe et C. Szegedy. **Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift**, ICML 2015

Batch norm : en apprentissage

- Pour chaque batch (sous-ensemble d'exemples d'apprentissage), on **normalise chaque dimension** (i.e. chaque sortie de la couche précédente)

Moyenne et variance empirique de la sortie du k^{ème} neurone calculée à partir du batch courant

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

Fonction différentiable

- Est-ce toujours adapté ?

Plus d'infos : Why Does Batch Norm Work?: <https://www.youtube.com/watch?v=nUUqwaxLnWs>
Détails sur la passe de rétropropagation <https://kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-batch-normalization-layer.html>

Batch norm : en apprentissage

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

Est-ce toujours une bonne idée ?

On laisse au réseau la possibilité de modifier la dynamique des entrées en fonction des besoins (exemple ReLU)

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

Si il le souhaite, le réseau peut apprendre à « dénormaliser ».
Comment ?

Plus d'infos : Why Does Batch Norm Work?: <https://www.youtube.com/watch?v=nUUqwaxLnWs>
Détails sur la passe de rétropropagation <https://kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-batch-normalization-layer.html>

Batch norm : en apprentissage

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots x_m\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$

[Ioffe et Szegedy, 2015]

La BN peut s'appliquer :

- À la sortie de la fonction d'activation
- Ou juste avant d'appliquer la non-linéarité (c'est souvent le cas)

Quels paramètres utiliser en test ?

- Ceux estimés sur la base d'apprentissage (avec une moyenne glissante par exemple)

Que se passe-t-il si la distribution des données en test est très différente ?

- AdaBN [Li et al. 2017]

- Limites des modèles MLP classiques
- Définition de l'architecture
 - La fonction de coût entropie croisée
 - fonctions d'activation
 - Initialisation des paramètres (poids)
- L'optimisation des paramètres
 - Limites de la descente de gradient
 - Momentum
 - Adaptation du pas d'apprentissage
- Régularisation
 - Pénalisation de la norme des paramètres
 - Dropout

Régularisation

- Généralement : réduire l'erreur de prédiction sur la base d'apprentissage ne suffit pas
- Régularisation : ensemble des méthodes permettant **d'améliorer l'erreur en test** (généralisation)

Pénalisation de la norme des paramètres

- Objectif : limiter la capacité d'apprentissage du modèle pour éviter le sur-apprentissage

$$\mathcal{L}(\hat{y}, y^*) = \left(\sum_{i=1}^{N_{exemples}} \ell(f(x^{(i)}; \theta), y^{*(i)}) \right) + \boxed{\alpha \Omega(\theta)}$$

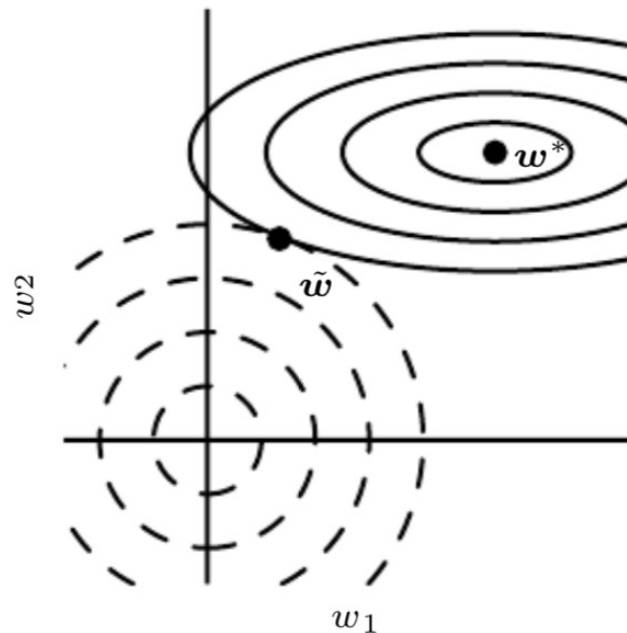
?

Norme des paramètres

Quelle norme choisir ?

Pénalisation de la norme des paramètres

- Norme L2 :
 - Weight decay
 - aka. ridge regression (ou Tikhonov regularization)
 - Favorise les faibles valeurs des paramètres en particulier dans les direction de faible courbure.



Pénalisation de la norme des paramètres

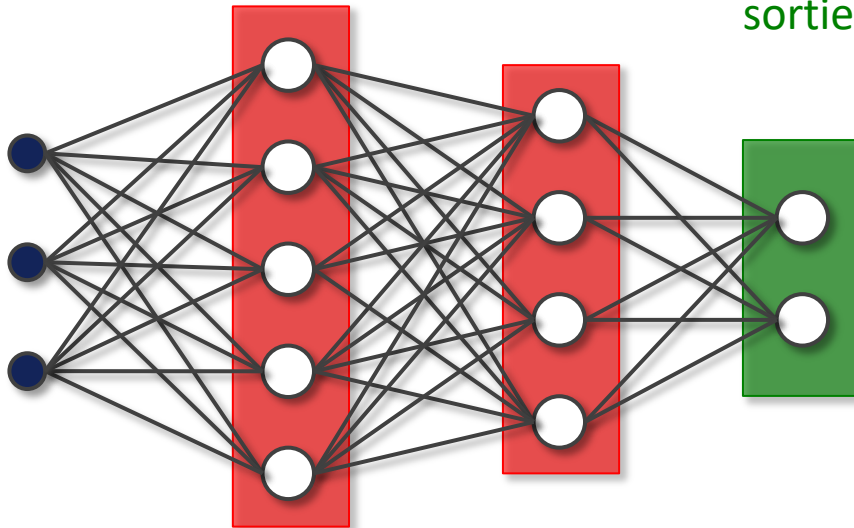
- Norme L2 :
 - Weight decay (somme des poids au carré)
 - aka. ridge regression (ou Tikhonov regularization)
 - Favorise les faibles valeurs des paramètres en particulier dans les directions de faible courbure.
- Norme L1 :
 - Somme des valeurs absolues des poids
 - aka. LASSO penalty
 - Favorise les solutions parcimonieuses (nombreuses valeurs nulles)

Dropout

Entrée

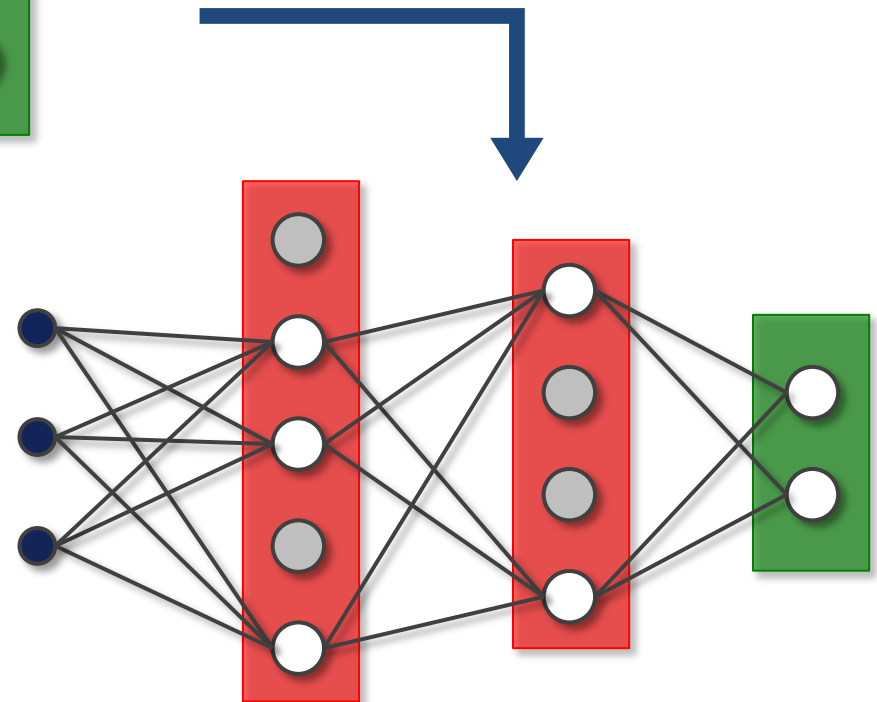
Couches cachées

Couche de
sortie



A chaque mise à jour des poids, un ensemble de neurones sont désactivés aléatoirement.

La probabilité p_{do} est un hyper-paramètre



Dropout

- Peut s'interpréter **comme une méthode d'ensemble** dont les modèles partagent certains paramètres
- **Force le réseau à être moins sensible au bruit** (surtout pour les descripteurs porteurs de beaucoup d'information)
- Temps d'apprentissage augmenté (+ d'epochs nécessaires)
- Capacité de généralisation améliorée

- **Et en test ?**

- Le modèle est stochastique et dépend d'un nouveau paramètre

$$\hat{y} = f(x, \boxed{z}; \theta)$$

Masque des désactivation aléatoires

- On ne souhaite pas dépendre de ce paramètre aléatoire

$$\hat{y} = \mathbb{E}_{\theta} [f(x, z; \theta)] = \int p(z) f(x, z; \theta) dz$$





- Autre solution : multiplier la prédiction par la valeur la probabilité du dropout

$$\hat{y} = p_{do} f(x; \theta)$$

- dropout inversé : on pondère les activations des neurones par $1/p_{do}$ en apprentissage et on ne modifie pas la sortie en test. (plus courant)

Autres méthodes de régularisation

- Augmentation des données
- Mixup / Cutout / CutMix
- Early stopping
- Injection de bruit (en entrée ou en sortie)
- Apprentissage multi-tâches
- Poids partagés (cf. CNN)
- Méthodes d'ensemble

	ResNet-50	Mixup [47]	Cutout [3]	CutMix
Image				
Label	Dog 1.0	Dog 0.5 Cat 0.5	Dog 1.0	Dog 0.6 Cat 0.4
ImageNet Cls (%)	76.3 (+0.0)	77.4 (+1.1)	77.1 (+0.8)	78.6 (+2.3)
ImageNet Loc (%)	46.3 (+0.0)	45.8 (-0.5)	46.7 (+0.4)	47.3 (+1.0)
Pascal VOC Det (mAP)	75.6 (+0.0)	73.9 (-1.7)	75.1 (-0.5)	76.7 (+1.1)

- Reprendre le réseau de reconnaissance des chiffres manuscrits (base MNIST) et tester différentes stratégies d'optimisation et de régularisation
- Quelle est la meilleure configuration trouvée ? Quelles performances ?
- Qu'apprend le réseau ?
 - Afficher sous forme d'images les poids des neurones de la première couche
 - Quelle différence observez-vous en fonction de l'algorithme d'optimisation utilisé ?