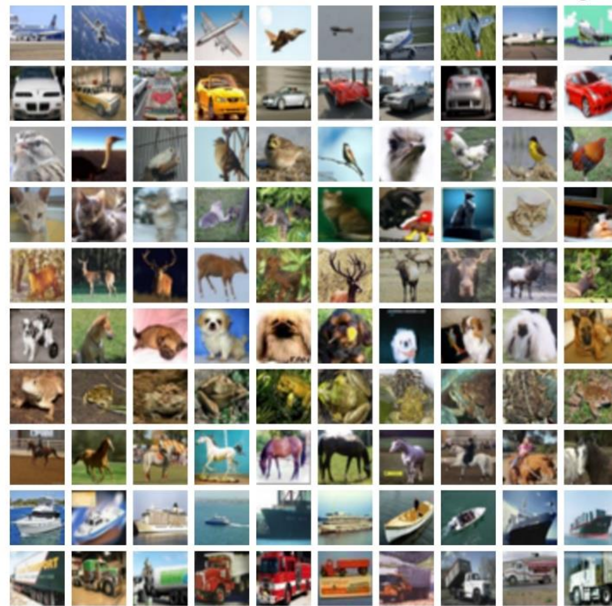
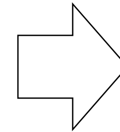
Réseau de neurones à propagation avant

Jian Tang
HEC Montréal
Institut IA Mila-Québec
Courriel : jian.tang@hec.ca

La tâche

- Le but est d’apprendre la fonction de mappage $y = f(x;\theta)$ (ex: pour une classification $f: R^d \rightarrow C$).

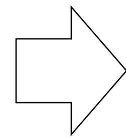




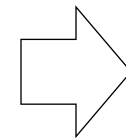
airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

Exemple: classification d’images

Apprentissage automatique traditionnel



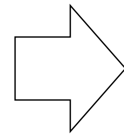
Extraction Artisanale de caractéristiques



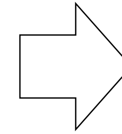
Classification Simple
Ex: SVM, LR

Expert du domaine

Apprentissage profond =
Apprentissage de caractéristiques / bout-à-bout



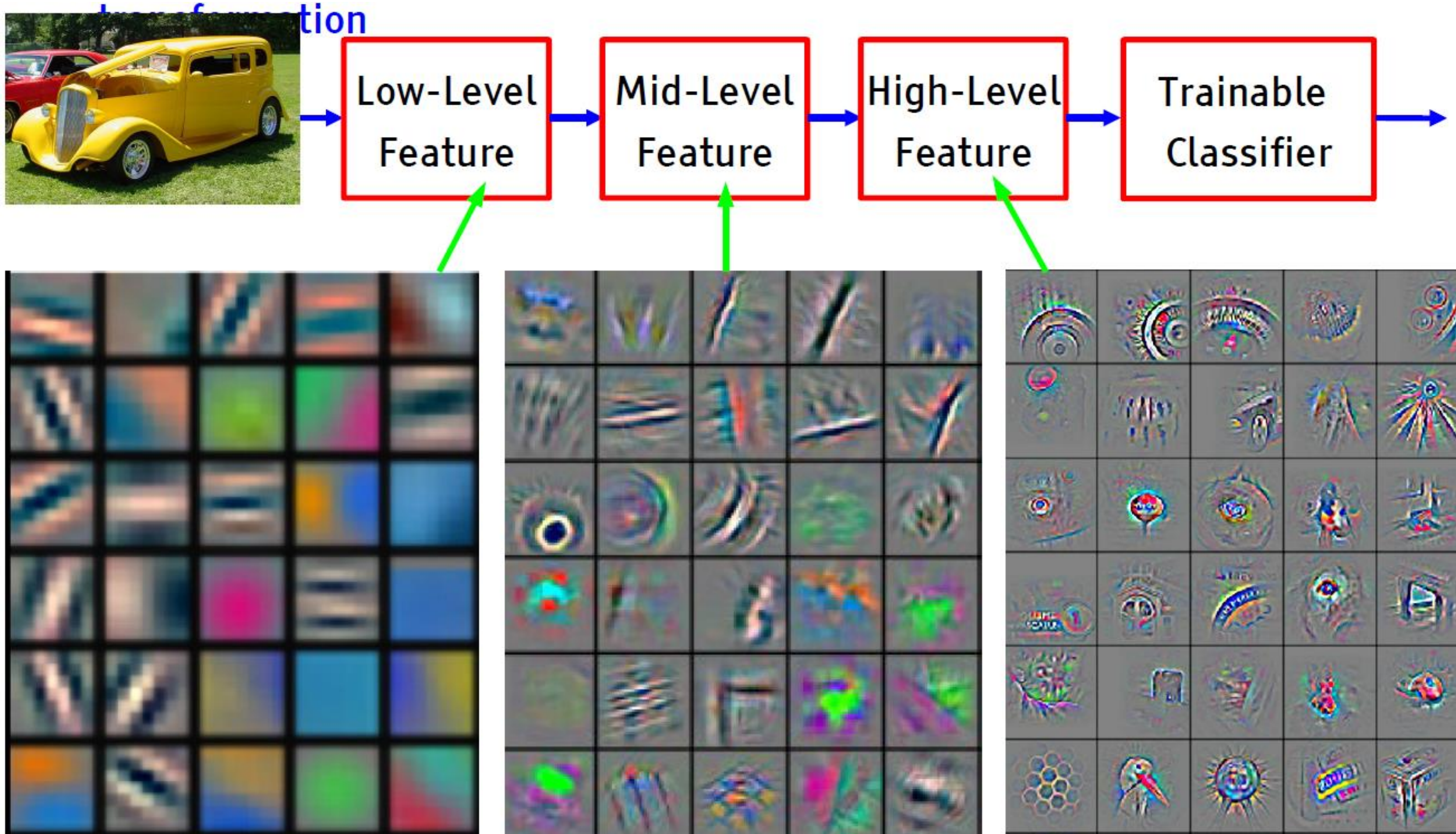
Extraction de caractéristiques entrainables



Classification simple et entrainable
Ex: SVM, LR

Expert du domaine

Apprentissage profond =
Apprendre les représentations hiérarchiques



(Figure from LeCun)

Représentations hiérarchiques avec niveau d'abstraction incrémental

- Reconnaissance d'images
  - Pixel -> bordure -> texture-> motif -> partie d'objet -> objet

- Reconnaissance de la parole
  - Échantillon -> bande spectrale-> son -> voix -> mot

- Texte
  - Caractère -> mot -> phrase -> clause -> paragraphe -> document

(slides from LeCun)

Plan

- Composants d’un réseau de neurones
  - Neurones (unités cachées)
  - Unités de sorties
  - Fonction de coût
- Design de l’architecture
  - Capacité du réseau de neurones
- Entrainement
  - Rétropropagation par algorithme du gradient stochastique

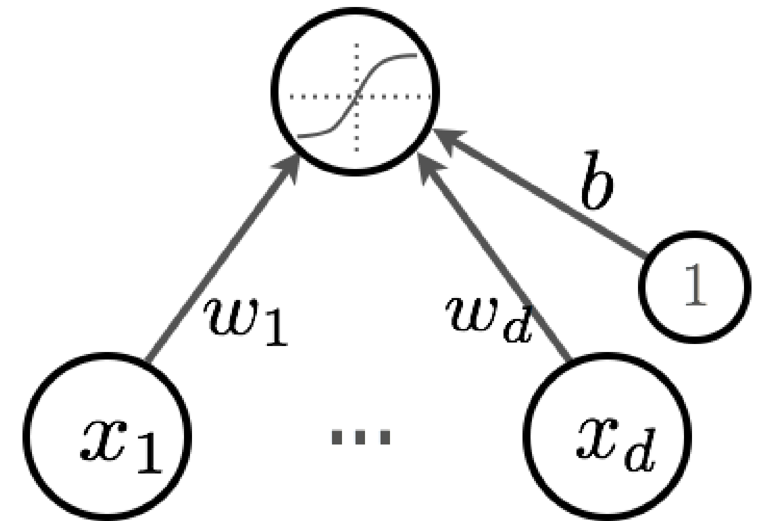
Neurones: fonctions non linéaires

- Entrée : combinaison linéaire:

$$a(\boldsymbol{x}) = b + \sum_i w_i x_i = \boldsymbol{w}^T \boldsymbol{x} + b$$

- Sortie: transformation non linéaire:

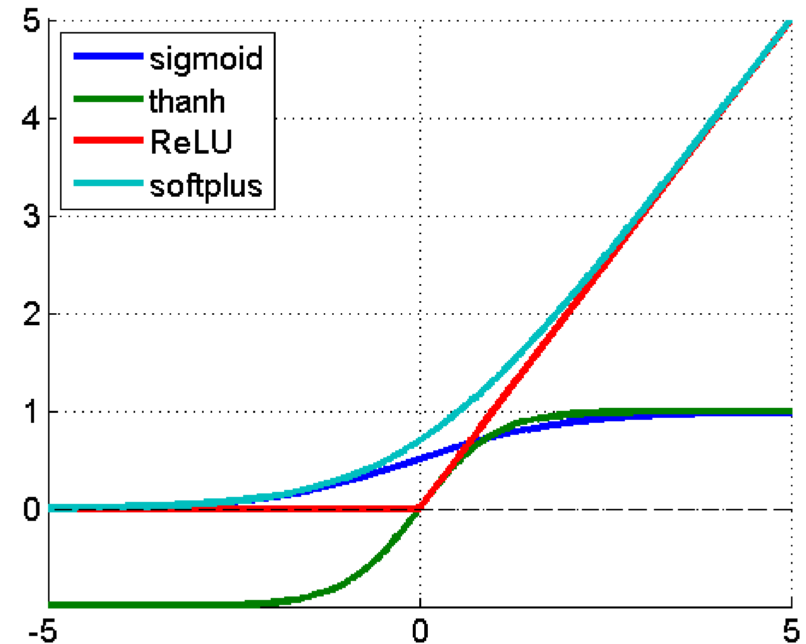
$$h(\boldsymbol{x}) = g(a(\boldsymbol{x})) = g(\boldsymbol{w}^T \boldsymbol{x} + b)$$



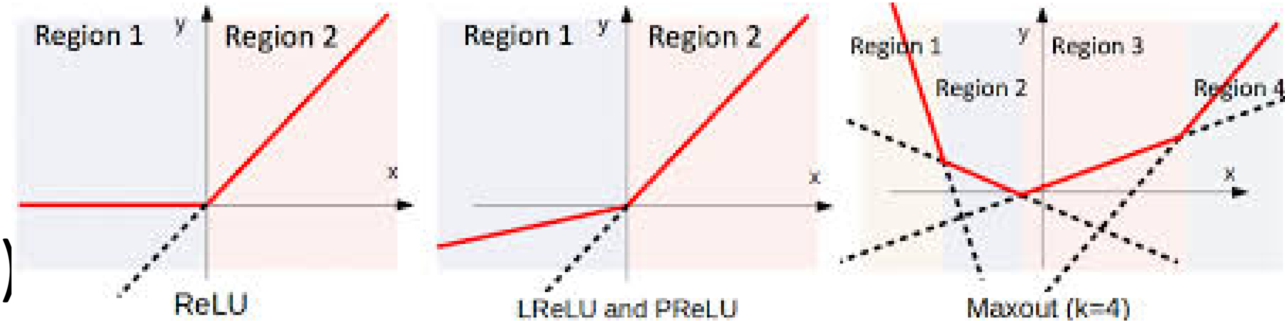
- w: sont les poids
- b: est le terme de biais
- g(.) est appelée la fonction d’activation

Fonctions d'activation / unités cachées

- Fonction sigmoïde
  - g(x) = 1/(1+exp(-x))
  - Mappage des valeurs entrantes à (0,1)
- Fonction tanh
  - g(x) = (1-exp(-2x))/(1+exp(-2x))
  - Mappage des valeurs entrantes à (-1,1)
- Fonction unité linéaire rectifiée (ReLU)
  - g(x) =max(0,x)
  - Aucune borne supérieure



Autres fonctions d'activation



- Leaky ReLU (Maas et al. 2013)
  - $g(x) = \max(0, x) + \alpha \min(0, x)$
  - Fixe α à une petite valeur, ex: 0.01
- ReLU Paramétrée (He et al. 2015)
  - Traite α comme un paramètre pouvant être appris
- Maxout units (Goodfellow et al. ,2013)
  - Généralisation de la fonction unité linéaire rectifiée
  - Divise les unités de sortie en des groupes de k valeurs, et prend la valeur maximale de chaque groupe
  - Permet d'apprendre une fonction linéaire par partie, en répondant au multiple directions pouvant être prises par les valeurs d'entrées.

Réseau de neurones à une couche cachée

- Entrée:

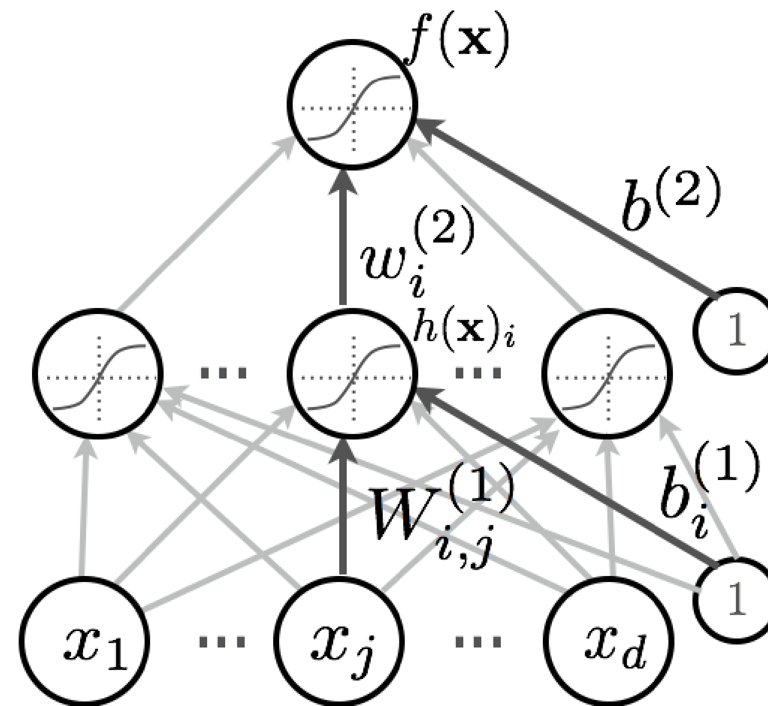
$$a(\boldsymbol{x}) = \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b}$$

- Transformation non linéaire:

$$h(\boldsymbol{x}) = g_1(a(\boldsymbol{x}))$$

- Sortie:

$$\mathrm{f}(\boldsymbol{x}) = o\,(h(\boldsymbol{x}))$$



Plan

- Composants d’un réseau de neurones
  - Neurones (unités cachées)
  - Unités de sorties
  - Fonction de coût
- Design de l’Architecture
  - Capacité du réseau de neurones
- Entrainement
  - Rétropropagation par Algorithme du gradient stochastique

Unité de sortie pour la distribution gaussienne

- Pour la tâche de régression, nous supposons généralement que la variable de sortie $\boldsymbol{y}$ suit une distribution gaussienne

$$p(\boldsymbol{y}|\boldsymbol{x}) = \mathrm{N}(\boldsymbol{y}|\hat{\boldsymbol{y}}, \boldsymbol{I})$$

- Pour la sortie **h** de la couche cachée, une couche linéaire produit $\hat{\boldsymbol{y}} = \boldsymbol{W}^T\boldsymbol{h} + \boldsymbol{b}$

Unités sigmoïdes pour une sortie de distribution binary

- Sortie de distribution binary: classification binaire
- Le but est de définir $p(y = 1|\boldsymbol{x})$, qui peut être définie comme suit:

$$p(y = 1|\boldsymbol{x}) = \sigma(\boldsymbol{w}^T \boldsymbol{h} + b)$$

Unités softmax pour une sortie de distribution multinomiale

- Distributions de sortie multinomiale: classification multiclasse
- Premièrement, définir une couche linéaire pour prédire la log probabilités non normalisée d'une fonction softmax:

$$\boldsymbol{z} = \boldsymbol{W}^T\boldsymbol{h} + \boldsymbol{b},$$

où $z_i = \log p(y = i|\boldsymbol{x})$. La fonction softmax est donnée par

$$p(y = i|\boldsymbol{x}) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Réseau de neuronnes à couches multiples

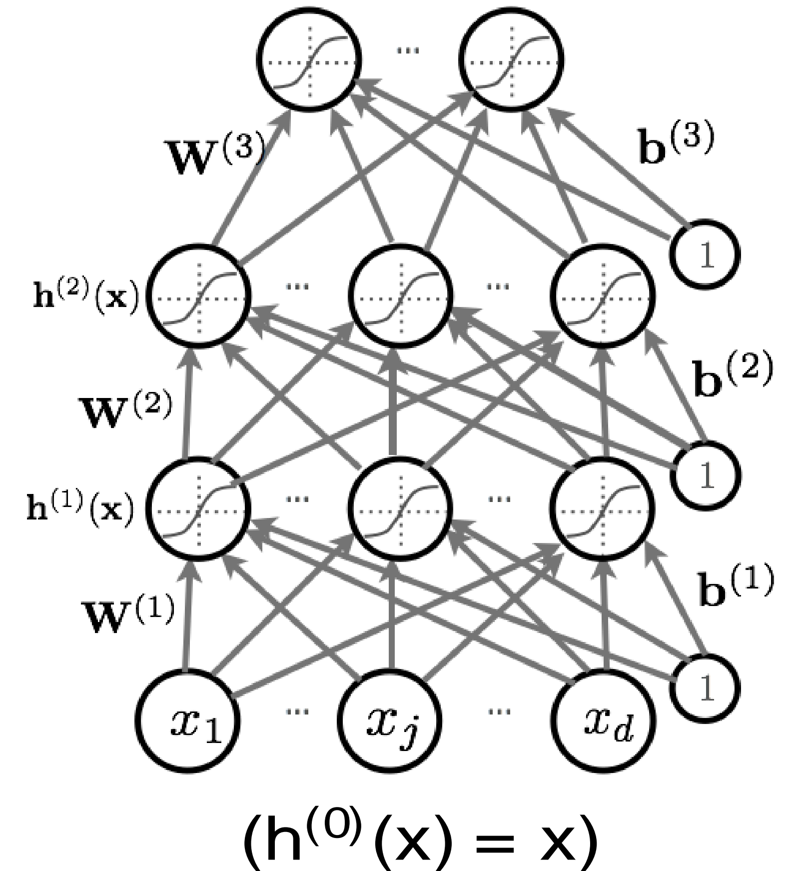
- Réseau de neurones à couches multiples
- La sortie des couches précédentes est l'entrée des couches suivantes:(k=1…, L)

$$a^{(k)}(x) = b^{(k)} + W^{(k)}h^{(k-1)}(x)$$

$$h^{(k)}(x) = g(a^{(k)}(x))$$

- Couche finale de sortie

$$\mathbf{f}(\mathbf{x}) = \mathbf{o}\left(\mathbf{h}^{L}(\mathbf{x})\right)$$



Plan

- Composants d’un réseau de neurones
  - Neurones (Unités Cachées)
  - Unités de sorties
  - Fonction de coût
- Design de l’Architecture
  - Capacité du réseau de neurones
- Entrainement
  - Rétropropagation par Algorithme du gradient stochastique

Maximum de vraisemblance

- La plupart du temps, les réseaux de neurones sont utilisés pour définir une distribution $\mathrm{p}(y^t|\boldsymbol{x}^t;\boldsymbol{\theta})$. De ce fait, l'objectif général est défini comme:

$$argmax_{\boldsymbol{\theta}} \frac{1}{T}\sum_t \log p(y^t|\boldsymbol{x}^t;\boldsymbol{\theta}) - \lambda\Omega(\boldsymbol{\theta})$$

- Ce qui est équivalent à minimiser **l'erreur d'entropie croisée**.

Plan

- Composants d'un réseau de neurones
  - Neurones (unités cachées)
  - Unités de sorties
  - Fonction de coût
- Design de l'architecture
  - Capacité du réseau de neurones
- Entrainement
  - Rétropropagation par algorithme du gradient stochastique

Approximation Universelle

- Théorème d'approximation universelle (Hornik, 1991)
  - “a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrary well, given enough hidden units”
- Cependant, il est possible que nous ne soyons pas capables de trouver les bons paramètres….
  - La couche permettant cette approximation pourrait être trop grande
  - Optimiser tel un réseau de neurones est difficile…

Des réseaux plus profonds sont préférables

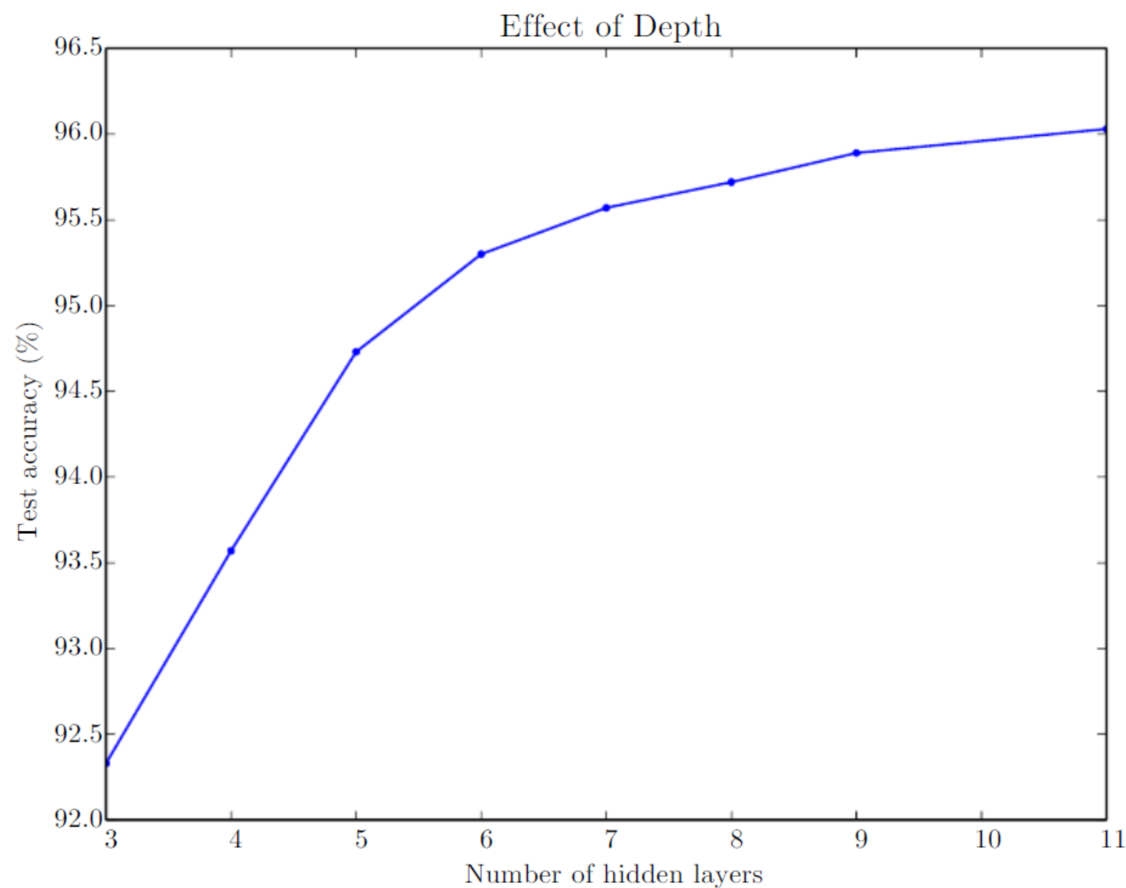


Figure: Résultats empiriques montrant que les réseaux plus profonds ont un plus grand pouvoir de généralisation

Des réseaux plus profonds sont préférables

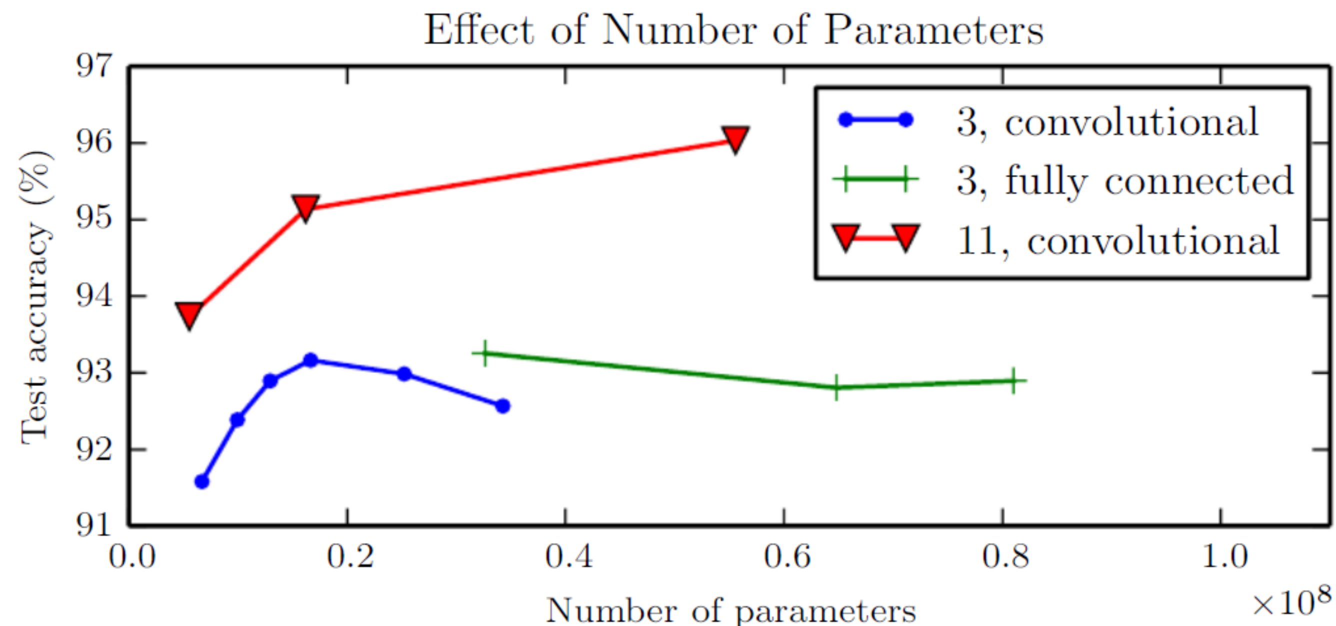


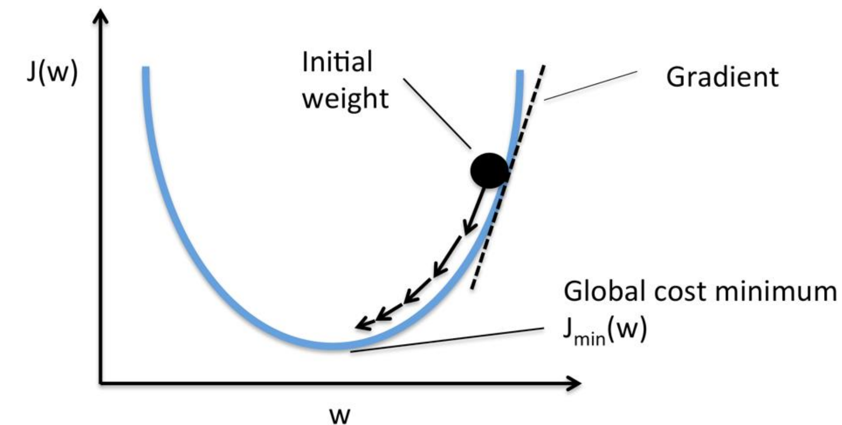
Figure: Des modèles plus profonds ont tendance à mieux performer pour un nombre de paramètres constant

Plan

- Composants d'un réseau de neurones
  - Neurones (Unités Cachées)
  - Unités de sorties
  - Fonction de coût
- Design de l'architecture
  - Capacité du réseau de neurones
- Entrainement
  - Rétropropagation par algorithme du gradient stochastique

Algorithme du gradient stochastique

- Algorithme du gradient (Gradient descent):
  - Mettre à jour les paramètres dans la direction negative du gradient
  - Besoin de calculer le gradient sur tous les exemples à chaque étape
- Algorithme du gradient stochastique



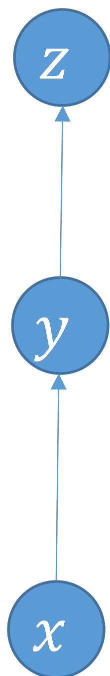
  – Initialize: $\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$
  – For t=1:T
    – for each training example $(\mathbf{x}^{(t)}, y^{(t)})$

$$\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\, \Delta$$

Training epoch
=
Iteration of all examples

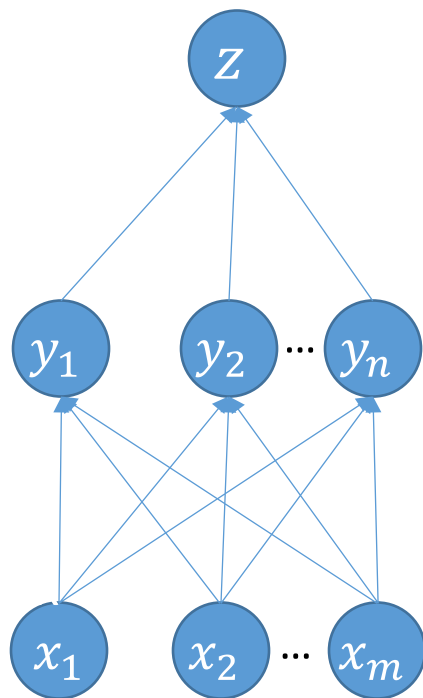
Rétropropagation: règle de la chaîne



$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

$$y = g(x)$$
$$z = f(y) = f(g(x))$$

Rétropropagation: simple dérivée en chaine



$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

$$\vec{y} = g(\vec{x})$$
$$z = f(\vec{y}) = f(g(\vec{x}))$$

Propagation Avant

- Pour chaque exemple d'entrainement (x, y), calculer la valeur de sortie ŷ basée sur le réseau de neurones courant et la valeur de pertes (supervisée) $\text{loss}(y, \hat{y})$

Require: Network depth, l
Require: $\boldsymbol{W}^{(i)}, i \in \{1, \dots, l\}$, the weight matrices of the model
Require: $\boldsymbol{b}^{(i)}, i \in \{1, \dots, l\}$, the bias parameters of the model
Require: $\boldsymbol{x}$, the input to process
Require: $\boldsymbol{y}$, the target output

$\boldsymbol{h}^{(0)} = \boldsymbol{x}$
for $k = 1, \dots, l$ **do**
  $\boldsymbol{a}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)} \boldsymbol{h}^{(k-1)}$
  $\boldsymbol{h}^{(k)} = f(\boldsymbol{a}^{(k)})$
end for
$\hat{\boldsymbol{y}} = \boldsymbol{h}^{(l)}$
$J = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \lambda \Omega(\theta)$

Propagation Arrière

- Calculer les gradients par rapport aux paramètres de chaque couche
  - Propager vers l'arrière les erreurs dans la sortie aux paramètres de chaque couche

---

After the forward computation, compute the gradient on the output layer:

$\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, y)$

for $k = l, l-1, \ldots, 1$ **do**

Convert the gradient on the layer's output into a gradient into the pre-nonlinearity activation (element-wise multiplication if f is element-wise):

$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = \boldsymbol{g} \odot f'(\boldsymbol{a}^{(k)})$

Compute gradients on weights and biases (including the regularization term, where needed):

$\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g} + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega(\theta)$

$\nabla_{\boldsymbol{W}^{(k)}} J = \boldsymbol{g}\, \boldsymbol{h}^{(k-1)\top} + \lambda \nabla_{\boldsymbol{W}^{(k)}} \Omega(\theta)$

Propagate the gradients w.r.t. the next lower-level hidden layer's activations:

$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)\top}\, \boldsymbol{g}$

end for

Régularisation et Optimisation

Qu’est-ce que la régularisation?

- Le but d’un algorithme d’apprentissage automatique est de bien performer sur le jeu de données d’entrainement et de bien généraliser sur de nouvelles données
- La régularisation permet d’améliorer la capacité de généralisation
  - C.-à-d. d’éviter le surapprentissage

Plan

- Régularisation
  - Pénalité sur la norme de paramètres
  - Augmentation du jeu de données
  - Robustesse au bruit
  - Apprentissage semi-supervisé
  - Apprentissage multi-tâche
  - Arrêt prématuré
  - Décrochage (Dropout)

Pénalité sur la norme de paramètres

- Ajouter une pénalité sur la norme de paramètre $\Omega(\boldsymbol{\theta})$ à la fonction objective J; La fonction objective régularisée est notée:

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \mathbf{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

- $\alpha \in [0, \infty)$ est un hyperparamètre contrôlant le poids du terme de régularisation
- Pour la régularisation dans les réseaux de neurones
  - Seulement les poids des transformations linéaires à chaque couche sont régularisés
  - Les biais ne sont pas régularisés (ils ont besoin de moins de données que les poids pour être bien ajustés)

Régularisation de Paramètres L^2

- $\Omega(\theta) = \frac{1}{2}||\mathbf{w}||^2$, aussi connu comme dégradation des pondérations (« weight decay») ou régression Ridge
- La fonction objective :

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

- Mettre à jour **w** avec SGD:

$$\boldsymbol{w} = (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon\nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

- Pousser **w** vers zéro

Régularisation de Paramètres L^1

- $\Omega(\theta) = ||\mathbf{w}||_1 = \sum_i w_i,$
- La fonction objective:

$$\tilde{J}(\boldsymbol{w}; \mathbf{X}, \boldsymbol{y}) = \alpha||\mathbf{w}||_1 + J(\boldsymbol{w}; \mathbf{X}, \boldsymbol{y})$$

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \mathbf{X}, \boldsymbol{y}) = \alpha \text{sign}(\mathbf{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \mathbf{X}, \boldsymbol{y})$$

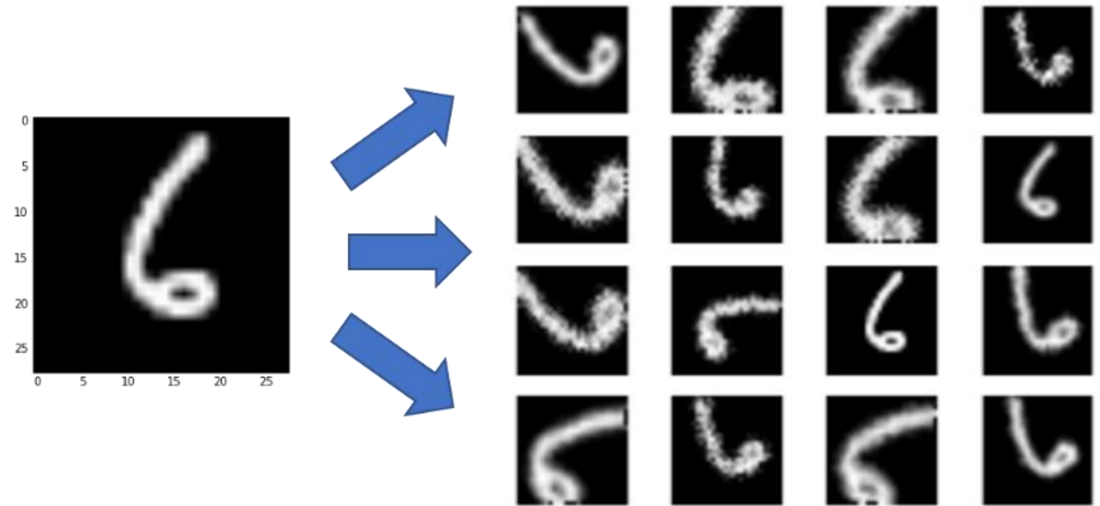
- Comparativement à la régularisation L2, la régularisation L1 trouve des parameters moins dense (« more sparse »)
  - Certains paramètres ont une valeur optimale à zéro
  - Peut être très utile pour la sélection de variables

Plan

- Régularisation
  - Pénalité sur la norme de paramètres
  - Augmentation des données
  - Robustesse au bruit
  - Apprentissage Semi-supervisé
  - Apprentissage Multi-tâche
  - Arrêt Prématurée
  - Décrochage (Dropout)

Augmentation des données

- Meilleure façon d’améliorer la performance en apprentissage automatique
  - Entrainer avec plus de données
- Ajouter des données artificielles au données d’entrainements
  - Translation
  - Rotation
  - Recadrage Aléatoire
  - Ajouter du bruit
  - …



Plan

- Régularisation
  - Pénalité sur la norme de paramètres
  - Augmentation du jeu de données
  - Robustesse au bruit
  - Apprentissage Semi-supervisé
  - Apprentissage Multi-tâche
  - Arrêt Prématurée
  - Décrochage (Dropout)

Robustesse au bruit

- Ajouter du bruit aux poids
  - Pousser le modèle dans des régions où le modèle est relativement insensible à de petites variations de poids
  - Trouver des points qui ne sont pas seulement des minimums mais des minimums entourés de régions plate.
- Ajouter du bruit aux données cibles
  - La plupart des jeu de données ont un certain niveau d'erreurs involontaires dans leur catégories cibles: y
  - On peut explicitement modéliser ce bruit dans les valeurs cibles
  - Par exemple, le valeur cible d'entrainement y est bonne avec une probabilité de $1 - \epsilon$, et peut prendre une autre valeur cible avec probabilité ϵ

Plan

- Régularisation
  - Pénalité sur la norme de paramètres
  - Augmentation du jeu de données
  - Robustesse au bruit
  - Apprentissage Semi-supervisé
  - Apprentissage Multi-tâche
  - Arrêt Prématurée
  - Décrochage (Dropout)

Apprentissage Semi-supervisé

- Apprentissage semi-supervisé: des exemples non libellés $p(x)$ et des exemples libellés $p(x,y)$ sont utilisés pour estimer $p(y|x)$
- Paramètres partagés entre l’objectif non-supervisé $p(x)$ et l’objectif supervisé $p(y|x)$
  - Dans les deux objectifs le but est d’apprendre la représentation $h = f(x)$, et cette représentation peut être utilisée dans les deux cas
- Un sujet très tendance actuellement
  - Spécifiquement dans le pré entrainement des modèles linguistiques (NLP).

Exemple:

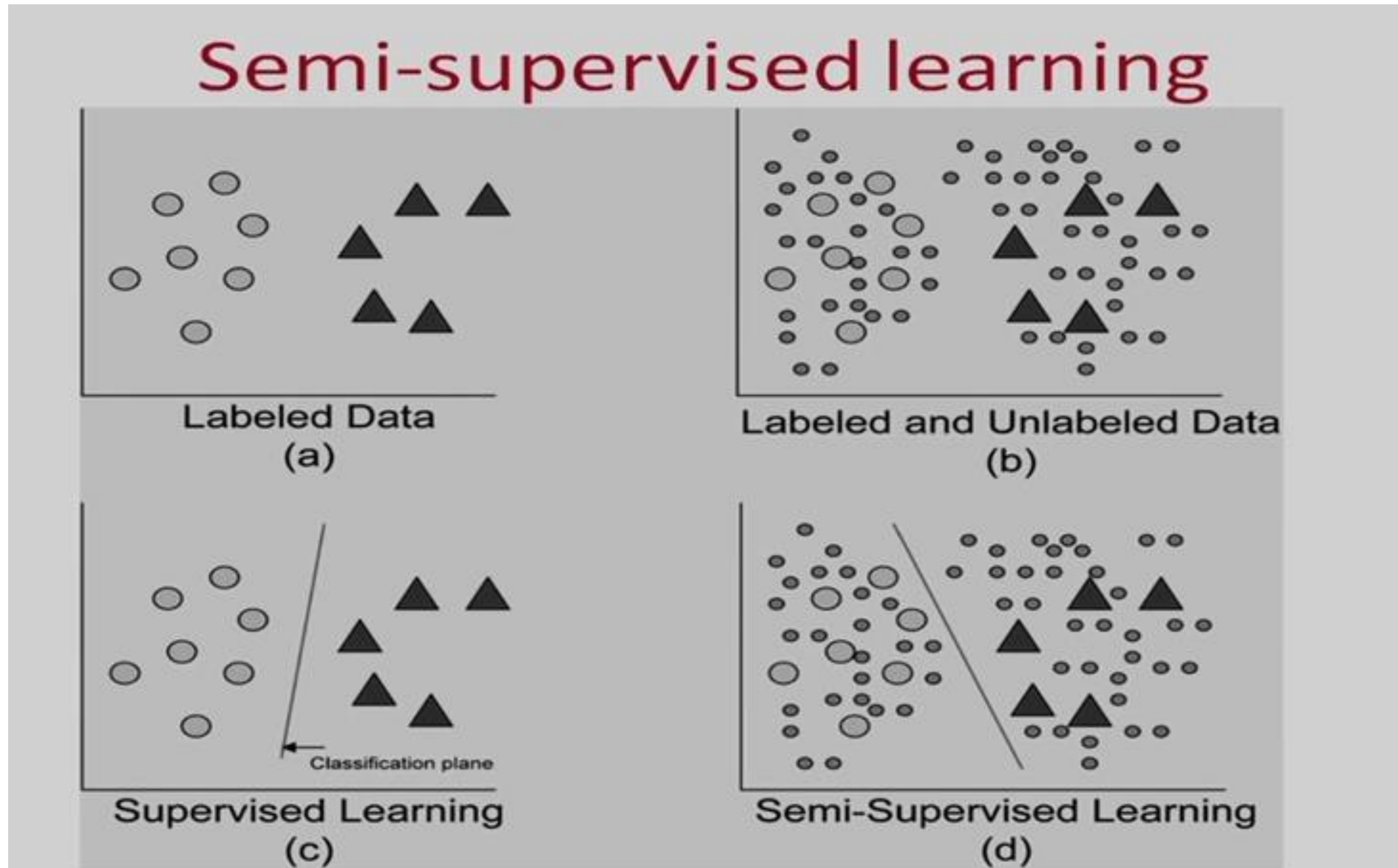


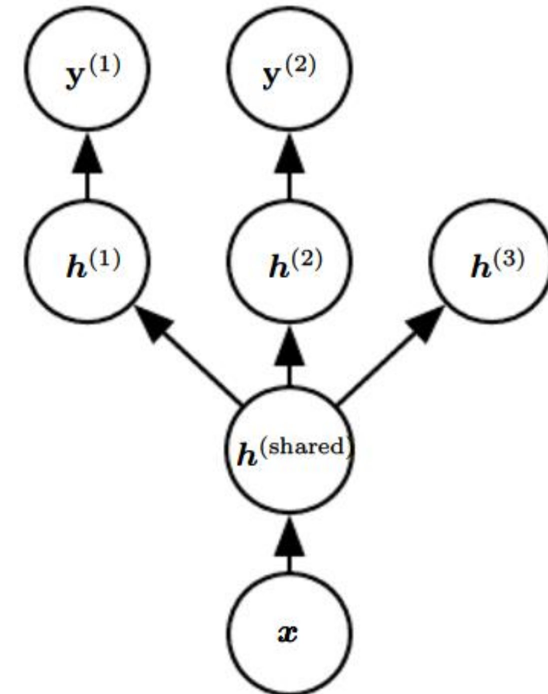
Image provenant du web

Plan

- Régularisation
  - Pénalité sur la norme de paramètres
  - Augmentation du jeu de données
  - Robustesse au bruit
  - Apprentissage Semi-supervisé
  - Apprentissage Multi-tâche
  - Arrêt Prématurée
  - Décrochage (Dropout)

Apprentissage Multi-tâche

- Apprendre conjointement plusieurs tâches en partageant les mêmes valeurs entrantes et certaines représentations intermédiaires capturant un ensemble de facteurs communs
- Modèle
  - Paramètres spécifiques à la tâche
  - Paramètres génériques partagées dans différentes tâches

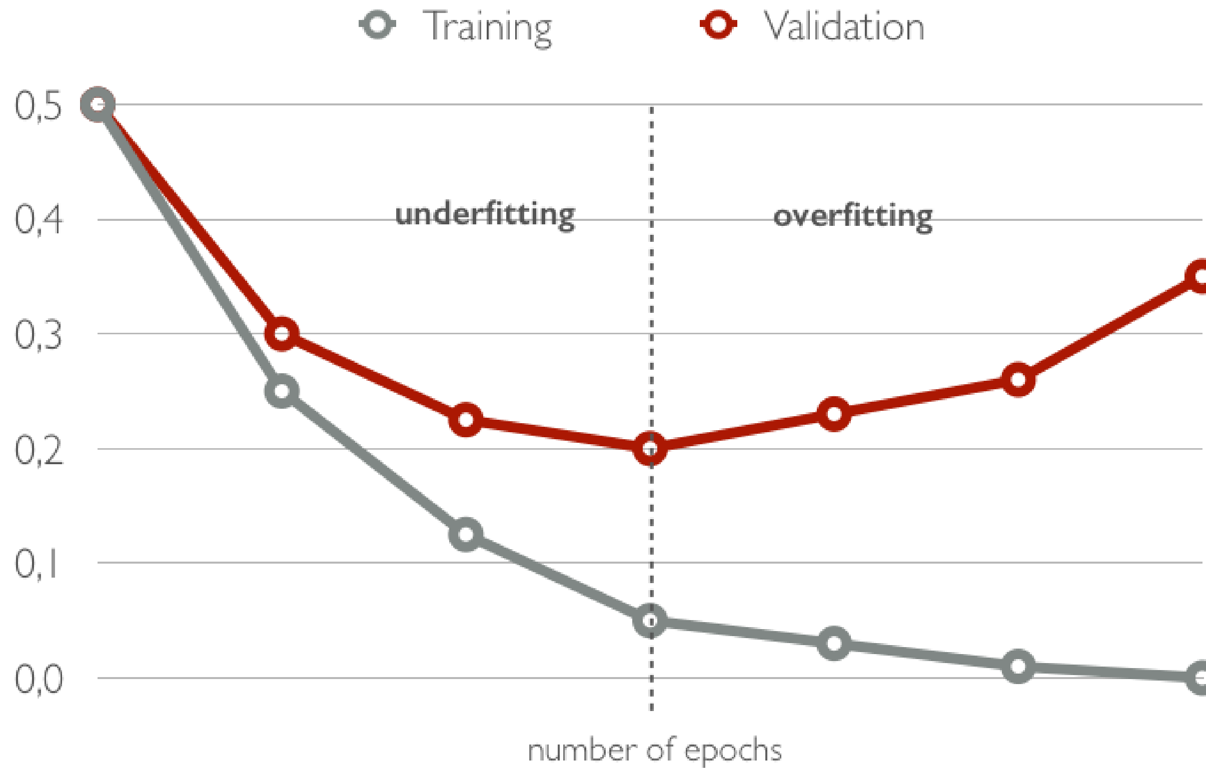


Plan

- Régularisation
  - Pénalité sur la norme de paramètres
  - Augmentation du jeu de données
  - Robustesse au bruit
  - Apprentissage Semi-supervisé
  - Apprentissage Multi-tâche
  - Arrêt Prématuré
  - Décrochage (Dropout)

Arrêt Prématuré

Pour sélectionner le nombre d'epochs optimal, arrêter l'entrainement lorsque l'erreur du jeu de données de validation augmente (de manière consécutive)



Plan

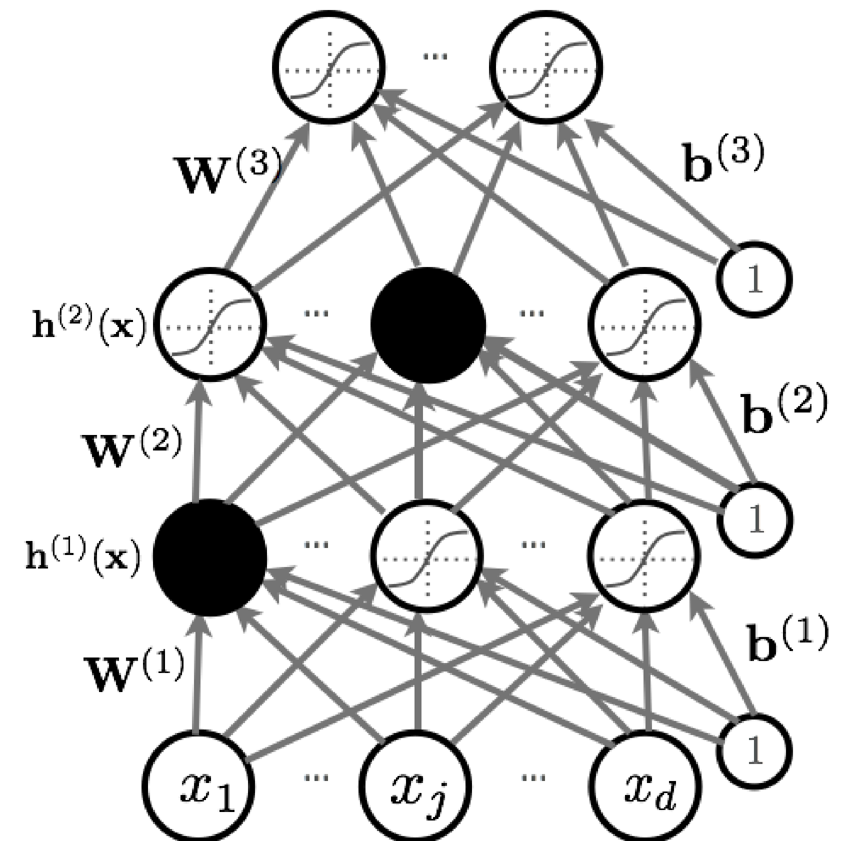
- Régularisation
  - Pénalité sur la norme de paramètres
  - Augmentation du jeu de données
  - Robustesse au bruit
  - Apprentissage semi-supervisé
  - Apprentissage multi-tâche
  - Arrêt prématuré
  - Décrochage (Dropout)

Décrochage

- Surmonter le surapprentissage en utilisant plusieurs modèles différents
  - Entrainés avec différentes architectures
  - Entrainés sur différents jeux de données
- Trop coûteux avec des réseaux de neurones
- Décrochage (Dropout)
  - Entrainer plusieurs réseaux de neurones ensemble en partageant les paramètres

Décrochage

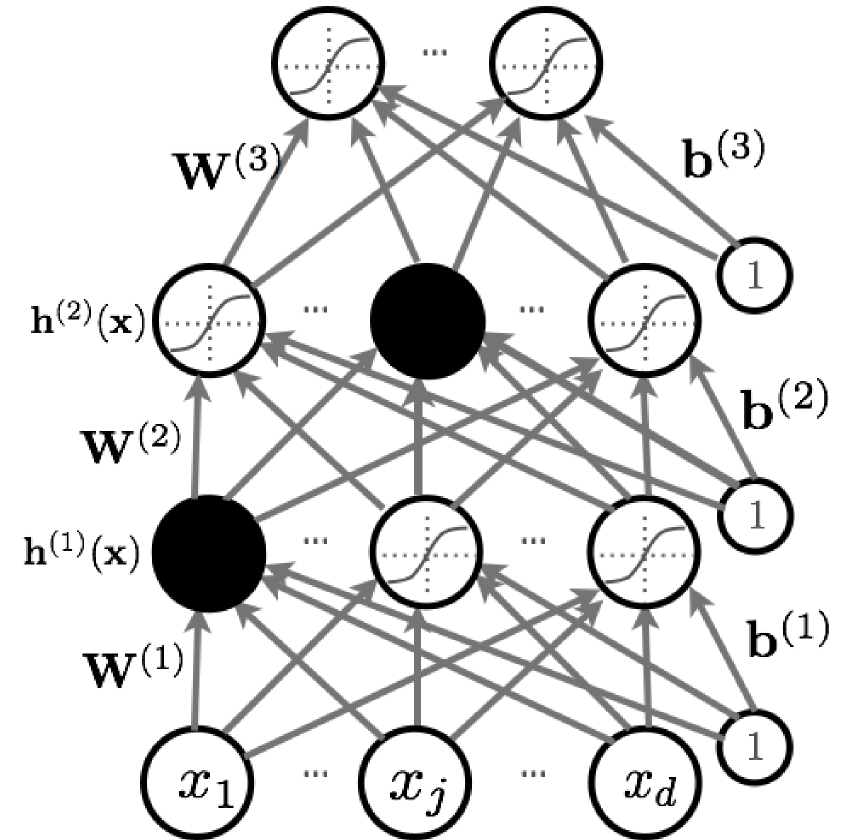
- **Idée principale**: enlever certains neurones de manière stochastique
  - Chaque neurone est fixé à 0 avec une probabilité de 0.5
- On peut utiliser une autre probabilité mais généralement 0.5 fonctionne bien



Décrochage

- Utiliser des masques binaires aléatoires $m^{(k)}$
  - Pré-activation de la couche pour $k > 0$
    - $a^{(k)}(x) = b^{(k)} + W^{(k)} h^{(k-1)}(x)$
  - Activation de la couche d'activation ($k = 1$ à L)

$$h^{(k)}(x) = g(a^{(k)}(x)) \odot m^{(k)}$$



Décrochage au test

- Au moment du test, on remplace les masques par leur espérance
    - Dans le cas où la probabilité est 0.5, on a un vecteur de valeurs constantes 0.5

Optimisation

- Stratégie d'initialisation des paramètres
- Astuces et conseils
- Momentum
- Taux d'apprentissage adaptatif (AdaGrad, RMSProp, Adam)

Initialisation de paramètres (Glorot and Bengio, 2010)

- Pour un réseau de neurones complètement connecté avec *m* valeurs entrantes et n valeurs sortantes, les poids sont échantillonnés selon:

$$W_{ij} \sim U\left(-\frac{6}{\sqrt{m+n}}, \frac{6}{\sqrt{m+n}}\right).$$

- Cela cherche à être un compromis entre le but d'initialiser toutes les couches pour avoir la même **variance d'activation** et le but d'initialiser toutes les couches pour avoir la même **variance de gradient**

Astuces

- Normaliser vos données (de valeurs réelles):
  - Pour chacune des dimensions x_i soustraire sa moyenne obtenue dans le jeu de données d'entraînement
  - Diviser chacune des dimensions x_i par son écart-type obtenu dans le jeu de données d'entraînement
  - Ceci peut améliorer le temps d'entraînement

- Réduire le taux d'entraînement : Lorsqu'on s'approche de l'optimum, faire un plus petit pas de mise à jour
  1. Commencer avec un grand taux d'apprentissage (« Learning Rate ») (ex: lr= 0.1)
  2. Garder ce taux jusqu'à ce que l'erreur de validation arrête de diminuer
  3. Diviser le taux d'apprentissage par 2 et retourner à (ii)

Mini-Lot et Momentum

- Faire des mises à jour sur de mini-lot d'observations (au lieu d'une observation):
  - Peut donner des estimés plus précis du gradient
  - Peut utiliser des opérations Matrice/Matrice qui sont plus efficaces

- Momentum: Peut utiliser une moyenne exponentielle des gradients précédents:

$$\overline{\nabla}_{\boldsymbol{\theta}}^{(t)} = \nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}), y^{(t)}) + \beta \overline{\nabla}_{\boldsymbol{\theta}}^{(t-1)}$$

Pourquoi le Momentum fonctionne vraiment?

Le momentum **réduit les mises à jour des dimensions pour lesquelles le gradient change de directions**

Le momentum augmente pour les **dimensions pour lesquelles le gradient pointe dans la même direction**

Demo : http://distill.pub/2017/momentum/

Taux d’apprentissage adaptatif

- Mise à jour avec taux d’apprentissage adaptatif (Un taux d’apprentissage par paramètre)
  - **Adagrad**: Taux d’apprentissage sont mis à l’échelle selon la racine carrée de la somme cumulative des carrés des gradients

$$\gamma^{(t)} = \gamma^{(t-1)} + \left(\nabla_\theta l(\mathbf{f}(\mathbf{x}^{(t)}), y^{(t)})\right)^2 \quad \overline{\nabla}_\theta^{(t)} = \frac{\nabla_\theta l(\mathbf{f}(\mathbf{x}^{(t)}), y^{(t)})}{\sqrt{\gamma^{(t)} + \epsilon}}$$

  - **RMSProp**: au lieu de la somme cumulative, on utilise une moyenne mobile exponentielle

$$\gamma^{(t)} = \beta\gamma^{(t-1)} + (1-\beta)\left(\nabla_\theta l(\mathbf{f}(\mathbf{x}^{(t)}), y^{(t)})\right)^2$$

$$\overline{\nabla}_\theta^{(t)} = \frac{\nabla_\theta l(\mathbf{f}(\mathbf{x}^{(t)}), y^{(t)})}{\sqrt{\gamma^{(t)} + \epsilon}}$$

  - **Adam**: combine essentiellement RMSProp avec le momentum

Références

- Livre “Deep Learning”
  - Chapitre 7-8

Précision

- Certaines diapos on été prises depuis le cours d’apprentissage profond du CMU de Ruslan Salakhutdinov.