

Apprentissage Statistique

Master DAC - Université Paris 6 et Master Math – Spécialité Big Data - Université Paris 6

P. Gallinari, patrick.gallinari@lip6.fr, <http://www-connex.lip6.fr/~gallinar/>

L. Denoyer, ludovic.denoyer@lip6.fr, O. Schwandler, olivier.schwandler@lip6.fr

Année 2015-2016

Plan du cours

- ▶ **Introduction**
 - ▶ Apprentissage à partir d'exemples
 - ▶ Exemple introductif : le perceptron
- ▶ **Formalisation du problème d'apprentissage**
- ▶ **Apprentissage supervisé**
 - ▶ Réseaux de neurones
 - ▶ Réseaux de neurones profonds (deep learning) et apprentissage de représentations
 - ▶ Réseaux récurrents
 - ▶ Machines à noyaux, machines à vecteurs de support
- ▶ **Apprentissage non supervisé**
 - ▶ Algorithme EM et mélange de densités
 - ▶ Clustering spectral
 - ▶ Factorisation matricielle
- ▶ **Introduction à la théorie de l'apprentissage**



Introduction



Apprentissage à partir d'exemples

- ▶ 3 ingrédients de base
 - ▶ Données $\{z^1, \dots, z^N\}$
 - ▶ Machine ou modèle F_θ
 - ▶ Critère C (apprentissage et évaluation)
- ▶ But
 - ▶ Extraire de l'information à partir des données
 - ▶ Information pertinente
 - pour la tâche étudiée
 - pour d'autres données du même type
- ▶ Utilisation
 - ▶ Inférence sur de nouvelles données
- ▶ Type d'apprentissage :
 - ▶ Supervisé
 - ▶ Non supervisé
 - ▶ Semi supervisé
 - ▶ Renforcement

Exemples - problèmes d'apprentissage

- ▶ Parole / Ecriture
 - ▶ Données : (signal, (transcription))
 - ▶ But : reconnaître signal
 - ▶ Critère : # mots correctement reconnus
- ▶ Conduite véhicule autonome
 - ▶ Données : (images routes, (commande volant)) e.g. S.Thrun Darpa Challenge + Google car
 - ▶ But : suivre route
 - ▶ Critère : distance parcourue
- ▶ Recherche d'information textuelle
 - ▶ Données : (texte + requête, (information pertinente)) – corpus d'apprentissage
 - ▶ But : extraire l'information correspondant à la requête
 - ▶ Critère : Rappel / Précision
- ▶ Diagnostic dans systèmes complexes
 - ▶ Données : (état capteurs + alarmes, (diagnostic))
 - ▶ But : diagnostic correct
 - ▶ Critère : ?

Exemples - problèmes d'apprentissage

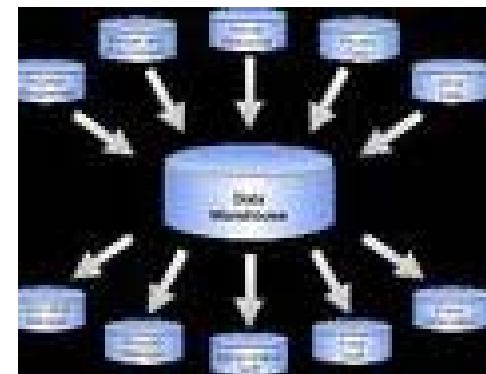
- ▶ Modélisation d'utilisateur
 - ▶ Données : (Traces utilisateur)
 - ▶ But : analyser/ modéliser le comportement de l'utilisateur
 - ▶ Exemples : ciblage clientèle, aide navigation, publicité, recommandation, assistants personnels e.g. Google Now, etc
 - ▶ Critère : ?
 - ▶ Evaluation : ?
 - ▶ Example Google Now
 - ▶ Google Now keeps track of searches, calendar events, locations, and travel patterns. It then synthesizes all that info and alerts you—either through notifications in the menu bar or cards on the search screen—of transit alerts for your commute, box scores for your favorite sports team, nearby watering holes, and more. You can assume it will someday suggest a lot more.



Exemples - problèmes d'apprentissage

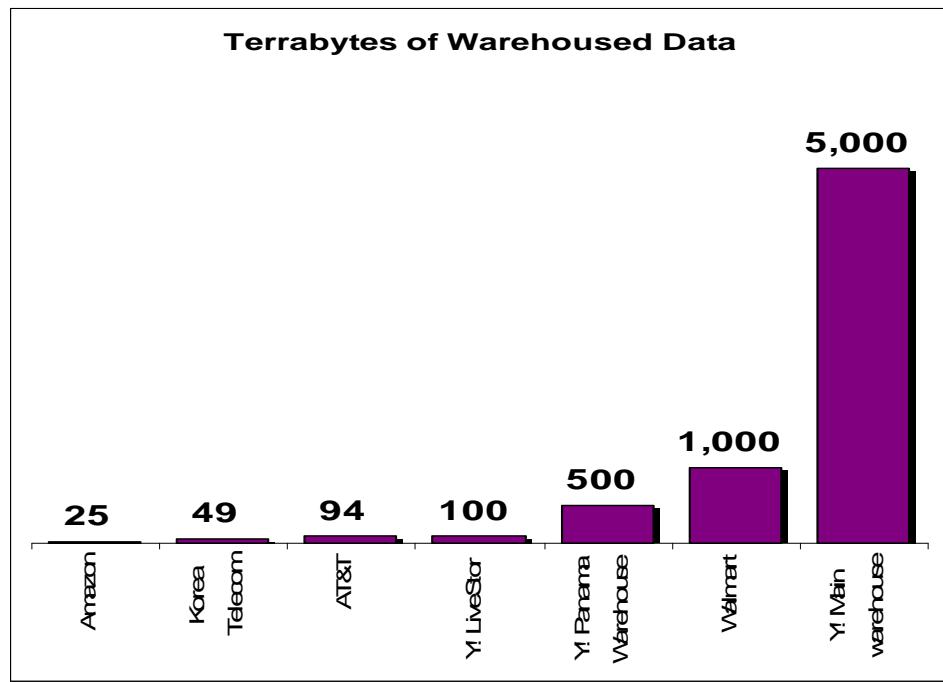
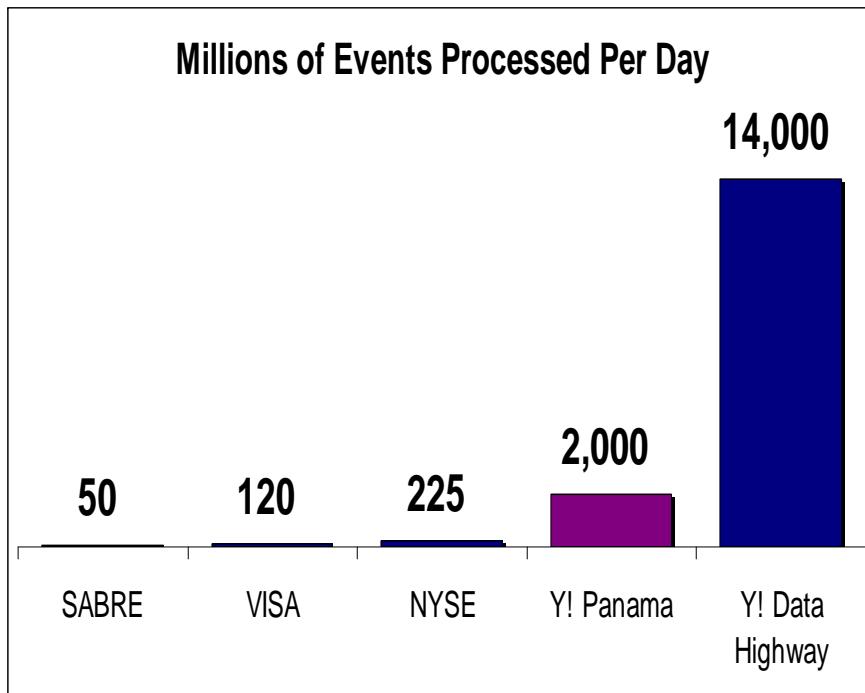
- ▶ Plus complexe:
 - ▶ Traduction
 - ▶ Extraction d'information (e.g. Never-Ending Language/ Image Learning)
 - ▶ Compréhension de texte / scène visuelle – extraction de sens
 - ▶ Découverte dans bases de données ou bases de connaissances, web, etc
 - ▶ Données : i.e. représenter l'information ??
 - ▶ But ??
 - ▶ Critère ??
 - ▶ Evaluation ??

Données : diversité



Données : quantités

Yahoo! Data – A league of its own... U. Fayyad KDD'07



GRAND CHALLENGE PROBLEMS OF DATA PROCESSING

TRAVEL, CREDIT CARD PROCESSING, STOCK EXCHANGE, RETAIL, **INTERNET**

Y! PROBLEM EXCEEDS OTHERS BY 2 ORDERS OF MAGNITUDE

Données : quantités Petabytes (10^{15}) (chiffres 2012)

- ▶ Google processes about 24 petabytes of data per day
- ▶ Google Street View Has Snapped 20 Petabytes of Street Photos
- ▶ Telecoms: AT&T transfers about 30 petabytes of data through its networks each day
- ▶ Physics: The experiments in the Large Hadron Collider produce about 15 petabytes of data per year
- ▶ Neurology: It is estimated that the human brain's ability to store memories is equivalent to about 2.5 petabytes of binary data

Big Data: Volume, Velocity, Variety, and Veracity

<http://www-01.ibm.com/software/data/bigdata/>

- ▶ **Volume: terabytes, petabytes**
 - ▶ Turn 12 terabytes of Tweets created each day into improved product sentiment analysis
 - ▶ Convert 350 billion annual meter readings to better predict power consumption
- ▶ **Velocity: streams**
 - ▶ Scrutinize 5 million trade events created each day to identify potential fraud
 - ▶ Analyze 500 million daily call detail records in real-time to predict customer churn faster
- ▶ **Variety:** Big data is any type of data - structured and unstructured data such as text, sensor data, audio, video, click streams, log files and more. New insights are found when analyzing these data types together.
 - ▶ Monitor 100's of live video feeds from surveillance cameras to target points of interest
 - ▶ Exploit the 80% data growth in images, video and documents to improve customer satisfaction
- ▶ **Veracity:** Establishing trust in big data presents a huge challenge as the variety and number of sources grows.

Gartner Hype Cycle: Machine Learning



- ▶ 2015: 1ere apparition du Machine Learning qui ne serait déjà plus dans le Hype, Big Data sort de la figure ...

We're making data science a sport.™

Participate in competitions

Kaggle is an arena where you can match your data science skills against a global cadre of experts in statistics, mathematics, and machine learning.

Whether you're a world-class algorithm wizard competing for prize money or a novice looking to learn from the best, here's your chance to jump in and geek out, for fame, fortune, or fun.

[Join as a participant](#)

(Need convincing?)

Create a competition

Kaggle is a platform for data prediction competitions that allows organizations to post their data and have it scrutinized by the world's best data scientists. In exchange for a prize, winning competitors provide the algorithms that beat all other methods of solving a data crunching problem. Most data problems can be framed as a competition.

[Learn more about hosting](#)

 Masters

[Browse all »](#)


Allstate.
You're in good hands.

Will I Stay or Will I Go? 

Predict which of our current customers will stay insured with us for an entire policy term.

Ends 2 months
10 teams

Have a Competition idea?

For Kaggle contestants:
Got a lead on a neat dataset?
[Let us know!](#)

Host a competition for...

 **Analytics**
Get the world's best predictive model.

 **Data Exploration**
Find the diamonds in your data.

 **Recruitment**
Uncover objectively brilliant candidates.

 **Education**
Free, powerful classroom competitions.

 **Kaggle Prospect**

[Browse all »](#)



Follow the Money: Investigative Reporting Prospect

Find hidden patterns, connections, and ultimately compelling stories in a treasure trove of data about US federal campaign

Ends 11 days
\$1,000

On the Forums

sluggish website
"Last post" forum bug
Are there Google calendars for



Data driven science – le 4e paradigme (Jim Gray MSR – Prix Turing)

- ▶ SNRI 2013 : <https://www.allistene.fr/contribution-dallistene-et-des-poles-de-competitivite-a-la-strategie-nationale-de-recherche-sciences-et-technologie-du-numerique/>
 - ▶ Extrait : « À l'heure actuelle, la science vit une révolution qui conduit à un nouveau paradigme selon lequel "la science est dans les données", autrement dit la connaissance émerge du traitement des données. »... » **« Le traitement de données et la gestion de connaissances représentent ainsi le quatrième pilier de la science après la théorie, l'expérimentation et la simulation.** L'extraction de connaissances à partir de grands volumes de données (en particulier quand le nombre de données est bien plus grand que la taille de l'échantillon) , l'apprentissage statistique, lagrégation de données hétérogènes, la visualisation et la navigation dans de grands espaces de données et de connaissances sont autant d'instruments qui permettent d'observer des phénomènes, de valider des hypothèses, d'élaborer de nouveaux modèles ou de prendre des décisions en situation critique »

Place de l'apprentissage

- ▶ L'apprentissage constitue une brique dans le processus de fouille / traitement de données
 - ▶ qui arrive souvent à la fin du processus
 - ▶ qui est intégré dans une application ou dans le SI de l'entreprise
- ▶ Les différentes étapes de l'analyse des données
 - ▶ Collecte des données / stockage
 - ▶ Prétraitement des données, étiquetage éventuel
 - ▶ Analyses des données par des techniques exploratoires
 - ▶ Mise au point et test de différents modèles d'apprentissage
 - ▶ Evaluation

Domaines d'application en Data Mining

Exemples

- ▶ **Web**
 - ▶ recherche d'information, filtrage d'information
 - ▶ extraction d'information textuelle : e.g. recherche, bibliothèques virtuelles, veille technologique, Question Answering , ...
- ▶ **Multi-média**
 - ▶ image + son, vidéo
- ▶ **Données d'entreprise**
 - ▶ infos produits, infos clients, ciblage clientèle ...
- ▶ **Analyse comportement**
 - ▶ e.g. telecoms : serveurs web, accès services commerciaux, internet - intranet, aide accès information, publicité
- ▶ **Distribué**
 - ▶ Mobiles : personnalisation, accès information
 - ▶ Capteurs distribués, objets connectés
- ▶ **Biologie - analyse de séquences, de structures**
- ▶ **Automobile ...**



4 Familles de problèmes

4 familles de problèmes d'apprentissage

- ▶ L'apprentissage propose des outils pour traiter des problèmes génériques
- ▶ C'est un domaine transverse à des domaines « d'application » comme la finance, la publicité, la vision, etc.
- ▶ Les 4 grandes familles de problèmes d'apprentissage
 - ▶ Supervisé
 - ▶ Non supervisé
 - ▶ Semi-supervisé
 - ▶ Renforcement
- ▶ Chaque famille traite d'un ensemble de problèmes génériques propres
 - ▶ Exemple de problème générique
 - ▶ En supervisé : classification, regression, ordonnancement,

Apprentissage supervisé

- ▶ Ensemble d'apprentissage constitué de couples (entrée, sortie désirée) $(x^1, d^1), \dots, (x^N, d^N)$
- ▶ Objectif : apprendre à associer les entrées aux sorties
 - ▶ Avec une bonne généralisation, i.e. $d = F_\theta(x)$ si x hors de l'ensemble d'apprentissage mais généré par le même phénomène ou un phénomène proche
- ▶ Utilisation : classification, regression, ordonnancement

A handwritten digit recognition diagram. On the left, there is a grid of handwritten digits. In the second row, the fourth digit from the left is circled in red. An arrow points from this circled digit to the word "sept" (French for "September") on the right, indicating that the digit '7' is being identified as the month 'sept'.

7	2	1	0	4	1	4	9	5	9
0	6	9	0	1	5	9	7	3	4
9	6	6	5	4	0	7	4	0	1
3	1	3	4	7	2	7	1	2	1
1	7	4	2	3	5	1	2	4	4

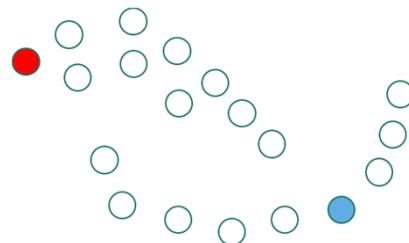
Apprentissage non supervisé

- ▶ Ensemble d'apprentissage
 - ▶ Uniquement des données d'entrée x^1, \dots, x^N , pas de sortie désirée associée
- ▶ But
 - ▶ Extraire des régularités des données
 - ▶ Similarités, relations entre données, facteurs sous jacent à la génération des données
- ▶ Utilisation
 - ▶ Estimation de densité, clustering, découverte de facteurs latents, ...



Apprentissage semi supervisé

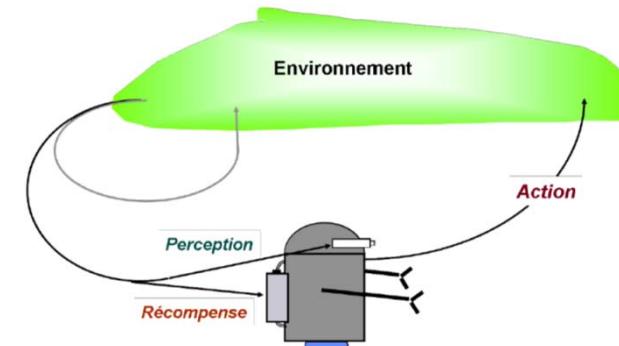
- ▶ Ensemble d'apprentissage
 - ▶ étiquetés – faible quantité $(x^1, d^1), \dots, (x^N, d^N)$
 - ▶ non étiquetés – grande quantité x^{N+1}, \dots, x^{N+M}
- ▶ But
 - ▶ Extraire l'information des exemples non étiquetés, utile pour l'étiquetage
 - ▶ Apprendre conjointement à partir des deux ensembles d'exemples



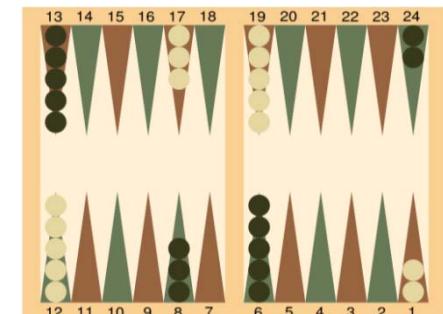
- ▶ Utilisation
 - ▶ grandes masses de données où l'étiquetage est possible mais coûteux, classification et calcul de scores associés aux noeuds

Apprentissage par Renforcement

- ▶ Ensemble d'apprentissage
 - ▶ Couples (entrée, sortie désirée qualitative) $(x^1, d^1), \dots, (x^N, d^N)$
 - ▶ Les x^i peuvent être des séquences (temporal credit assignment), les d^i sont des réponses qualitatives (e.g. 0,1), déterministes ou stochastiques.
- ▶ Paradigme
 - ▶ Système qui apprend à partir de l'exploration de son univers, à partir de récompense/ punitions
 - ▶ On parle de paradigme exploration/ exploitation



- ▶ Utilisation
 - ▶ commande, décision séquentielle, robotique, jeux à 2 joueurs, jeux collectifs, programmation dynamique, applications web ou sociales, ...
 - ▶ Exemple backgammon (TD Gammon Thesauro 1992)
 - ▶ Entrainé sur 1.5 M partie
 - ▶ Joue contre lui même



Algorithmes d'apprentissage numérique

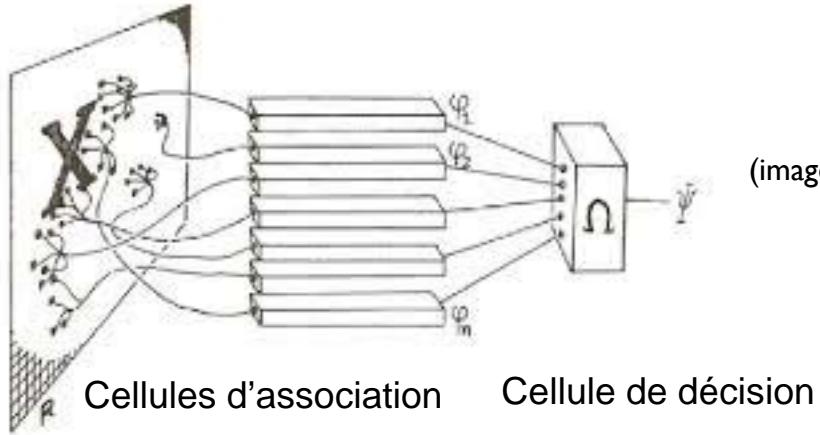
- ▶ Il existe un grand nombre d'algorithmes et de familles d'algorithmes pour les différents problèmes génériques abordés en apprentissage
- ▶ Il existe de nombreuses implémentations sous forme de programmes dédiés ou de logiciels plus généralistes
- ▶ Quelques exemples d'algorithmes populaires
 - ▶ **Données statiques**
 - ▶ Supervisé
 - Réseaux de neurones
 - Arbres décision / régression, random forest
 - Fonctions noyau, machines à vecteurs supports
 - Nombreux modèles pour l'estimation de densité paramétrique, non paramétrique
 - ▶ Non supervisé
 - Modèles à variables latentes
 - Modèles latents probabilistes
 - Factorisation matricielle
 - ▶ **Données structurées**
 - ▶ Séquences
 - Réseaux de neurones récurrents, modèles Markoviens, noyaux
 - ▶ Arbres et graphes
 - Modèles relationnels probabilistes, réseaux de neurones, noyaux

Des environnements de programmation pour l'apprentissage statistique

- ▶ Prototypes universitaires
 - ▶ Weka – Java based
- ▶ Systèmes industriels
 - ▶ Scikit-learn Python
 - ▶ Mlib – Apache Spark
 - ▶ Apache Mahout
 - ▶ RapidMiner : environnement pour data mining, Machine learning, predictive analytics, text ...
 - ▶ R langage et environnement pour statistiques + visualisation
 - ▶ KNIME environnement pour data mining, predictive analytics ...
 - ▶ Graphlab : environnement pour systèmes distribués
 - ▶ MS Azure Machine Learning
 - ▶ IBM Watson analytics
 - ▶
- ▶ Plus de nombreux systèmes dédiés à une famille d'algorithmes
 - ▶ Pour les réseaux de neurones
 - ▶ Torch
 - ▶ Theano
 - ▶ Caffe
 - ▶ ...

Exemple introductif : Perceptron

Un exemple : Perceptron (1960 Rosenblatt)



(image from Perceptrons, Minsky and Papert 1969)

- ▶ Le perceptron est utilisé pour la discrimination
- ▶ La cellule de décision calcule une fonction à seuil :
- ▶ $F(\mathbf{x}) = \text{sgn}(\sum_{i=1}^n w_i x_i + w_0) = \text{sgn}(\sum_{i=0}^n w_i x_i)$ avec $x_0 = 1$
 - Classe 1 : $\{\mathbf{x} : F(\mathbf{x}) = +1\}$
 - Classe 2 : $\{\mathbf{x} : F(\mathbf{x}) = -1\}$

L'algorithme du perceptron (2 classes)

Données

base d'apprentissage $\{(x^i, d^i, i = 1..N, x \in R^n, d \in \{-1,1\}\}$

Output

classifieur $w \in R^n$, décision $F(x) = sgn(\sum_{i=0}^n w_i x_i)$

Initialiser $w(0)$

Répéter (t)

choisir un exemple, $(x(t), d(t))$

Si $d(t)w(t) \cdot x(t) \leq 0$ alors $w(t+1) = w(t) + \epsilon d(t)x(t)$

Jusqu'à convergence

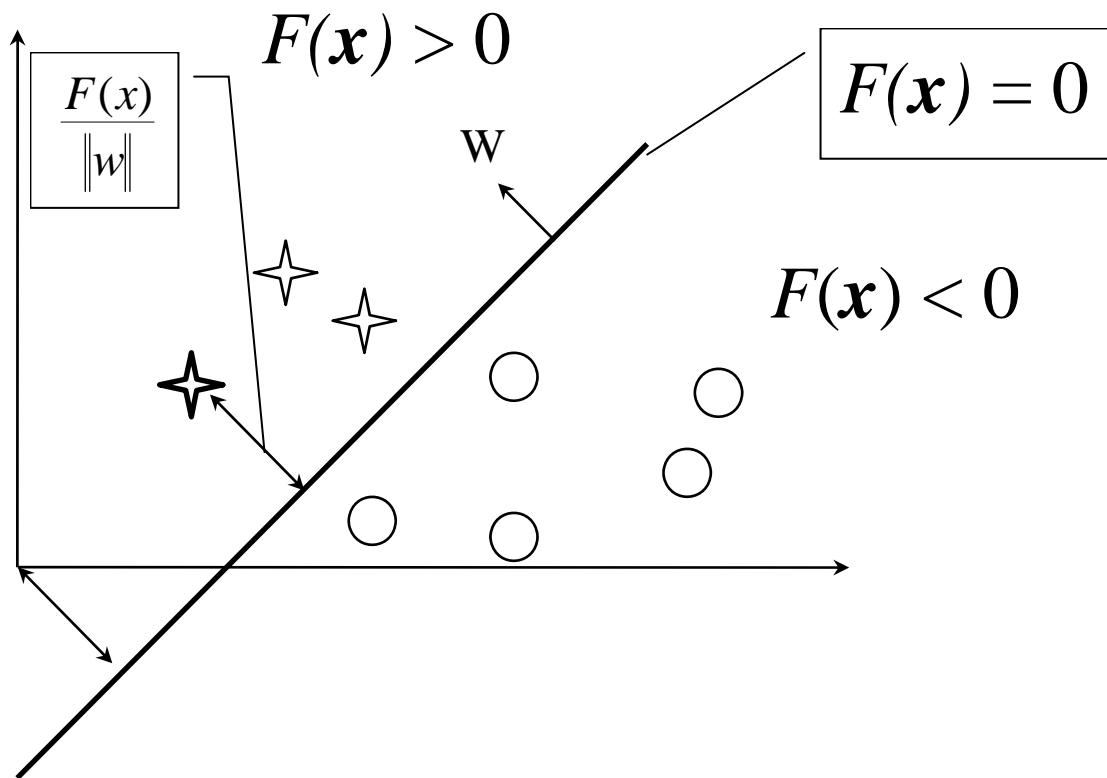
- ▶ C'est un algorithme à correction d'erreur
- ▶ si ϵ est constant : règle à incrément fixe
- ▶ si ϵ est fonction du temps : règle à incrément variable

Fonction discriminante linéaire

$$F(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + w_0 = \sum_{i=0}^n w_i x_i \text{ avec } x_0 = 1$$

- ▶ Surface de décision: hyperplan d'équation $F(\mathbf{x}) = 0$
- ▶ Quelques propriétés
 - ▶ \mathbf{w} est le vecteur normal de l'hyperplan, il définit son orientation
 - ▶ distance de \mathbf{x} à \mathcal{H} : $r = F(\mathbf{x})/\|\mathbf{w}\|$
 - ▶ $w_0 = 0$: \mathcal{H} passe par l'origine

Géométrie de la discrimination linéaire



Le perceptron effectue une descente de gradient

- ▶ Fonction de coût

- ▶ $C = - \sum_{(x,d) \text{ mal classé}} \mathbf{w} \cdot \mathbf{x} \cdot d$
- ▶ On veut minimiser C

- ▶ gradient

- ▶ $\text{grad}_{\mathbf{w}} C = \left(\frac{\partial C}{\partial w_1}, \dots, \frac{\partial C}{\partial w_n} \right)^T$ avec $\frac{\partial C}{\partial w_i} = - \sum_{(x,d) \text{ mal classé}} \mathbf{x} \cdot d$

- ▶ Règle d'apprentissage

- ▶ $\mathbf{w} = \mathbf{w} - \epsilon \text{grad}_{\mathbf{w}} C$
- ▶ Règle d'apprentissage du perceptron : algorithme adaptatif pour optimiser cette fonction de coût

- ▶ Demo

- ▶ <http://lcn.epfl.ch/tutorial/english/>

Cas multi-classes

- ▶ **Approche générale : one vs all**
 - ▶ p classes = p " problèmes 2 classes " : C_i contre le reste
 - ▶ construire p fonctions discriminantes $F_i(\mathbf{x})$, $i = 1 \dots p$
 - ▶ règle de décision: $\mathbf{x} \in C_i$ si $F_i(\mathbf{x}) > F_j(\mathbf{x})$ pour $j \neq i$
 - ▶ crée une partition de l'espace d'entrée
 - ▶ chaque classe est un polygone avec au plus $p - 1$ faces.
 - ▶ **Régions convexes : limitation des classificateurs linéaires**

Théorème de convergence du perceptron

(Novikov 1962)

- ▶ Si
 - ▶ $\exists R: \forall x, \|x\| \leq R$
 - ▶ les données peuvent être séparées avec une marge ρ , i.e.
 - ▶ $\sup_w \min_i d^i(x^i \cdot w) > \rho$
 - ▶ l'ensemble d'apprentissage est présenté au perceptron un nombre suffisant de fois
- ▶ Alors après au plus $[R^2/\rho^2]$ corrections, l'algorithme converge

Borne sur l'erreur de généralisation (Aizerman et al. 1964)

▶ Si

- ▶ les données sont séparables
- ▶ elles sont en nombre infini
- ▶ règle arrêt : après la $k^{\text{ème}}$ correction, les

$$\triangleright m_k = \frac{1+2 \ln k - \ln \eta}{-\ln(1-\epsilon)}$$

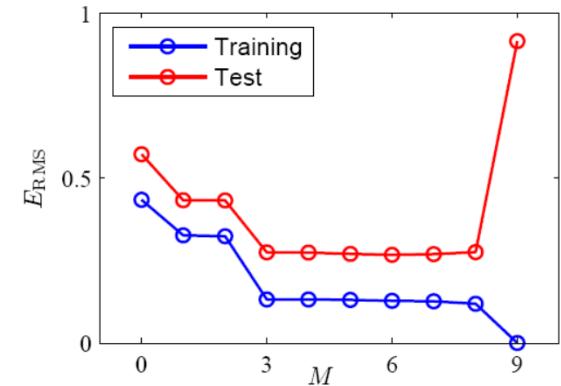
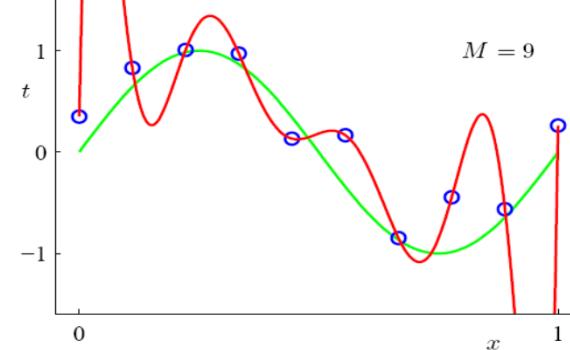
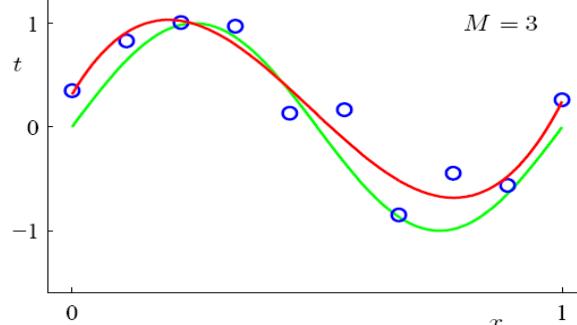
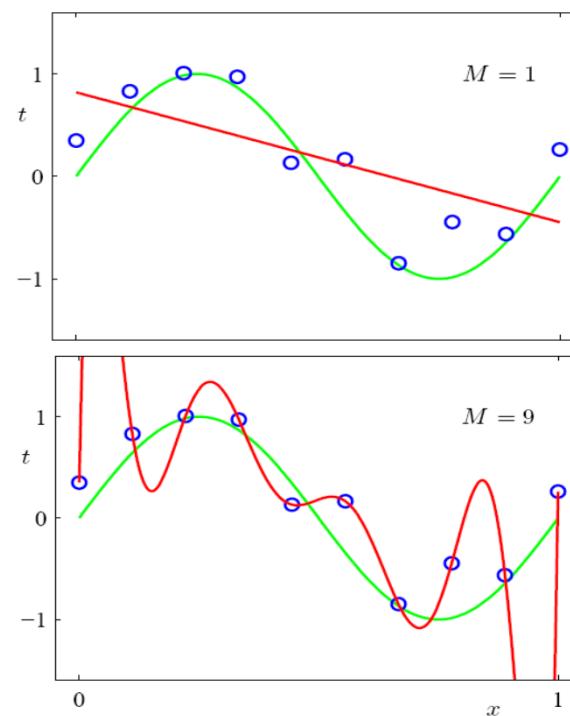
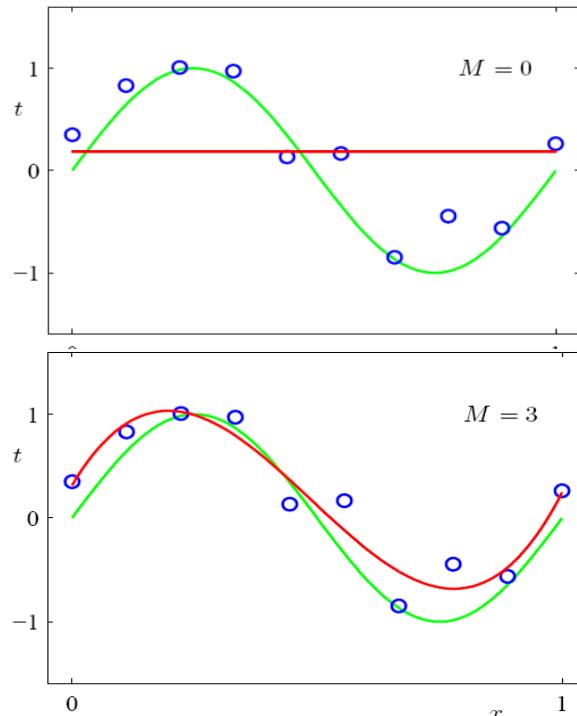
données présentées sont reconnues correctement

▶ alors

- ▶ le perceptron converge en $l \leq \frac{1+4 \ln R/\rho - \ln \eta}{-\ln(1-\epsilon)} [R^2/\rho^2]$ étapes
- ▶ avec une probabilité $1 - \eta$, l'erreur de test est $< \epsilon$

Overtraining / généralisation en regression

► Exemple (Bishop 06)



- Nécessité de contrôler lors de l'apprentissage la complexité des modèles
 - Techniques de régularisation

Données dans la pratique de l'apprentissage

- ▶ Distinguer les ensembles
 - ▶ d'apprentissage
 - ▶ Mettre au point le modèle
 - ▶ de test
 - ▶ Evaluer les performances du modèle appris
 - ▶ de validation
 - ▶ Apprentissage de méta-paramètres
- ▶ Remarque
 - ▶ On fera en général l'hypothèse que toutes les données sont générées suivant une même loi



Formalisation du problème de l'apprentissage



Formalisme probabiliste

- ▶ **Données**
 - ▶ vecteurs aléatoires (\mathbf{z}) générés suivant une densité $p(\mathbf{z})$
- ▶ **Machine d'apprentissage**
 - ▶ $\mathcal{F} = \{F_\theta\}_\theta$ avec θ les paramètres du modèle à valeur dans un espace réel
- ▶ **Coût**
 - ▶ $c_\theta(\mathbf{z})$ pour la machine F_θ et l'exemple \mathbf{z}
- ▶ **Risque théorique**
 - ▶ $R_\theta = E_{\mathbf{z}}[c_\theta(\mathbf{z})] = \int_{\mathbf{z}} c_\theta(\mathbf{z})p(\mathbf{z})d\mathbf{z}$
- ▶ **Solution optimale**
 - ▶ $F_{\theta^*} = \operatorname{argmin}_\theta R_\theta$

Apprentissage à partir d'exemples

- ▶ **Données**

- ▶ $D = \{\mathbf{z}^i\}_{i=1..N}$

- ▶ **Risque empirique**

- ▶ $C = \frac{1}{N} \sum_{i=1}^N c_\theta(\mathbf{z}^i)$

- ▶ **Principe inductif**

- ▶ **Exemple : Minimisation du risque empirique**

- ▶ La fonction F_{θ^*} qui minimise le risque théorique est approximée par $F_{\hat{\theta}}$ qui optimise le risque empirique
 - ▶ Est-ce un bon principe ?
 - ▶ Propriété de généralisation ?

Problèmes d'apprentissage : exemples

▶ Discrimination

- ▶ $\mathbf{z} = (\mathbf{x}, d), d \in \{0,1\}$
- ▶ F_θ ensemble des fonctions à seuil
- ▶ R : probabilité de mauvaise classification
- ▶ C : fréquence des erreurs

$$c_\theta(\mathbf{z}) = \begin{cases} 0 & \text{si } d = F_\theta(\mathbf{x}) \\ 1 & \text{sinon} \end{cases}$$

▶ Régression

- ▶ $\mathbf{z} = (\mathbf{x}, d), d \in R$
- ▶ F_θ un ensemble de fonctions réelles
- ▶ R : espérance des erreurs quadratiques
- ▶ C : somme des erreurs quadratiques

$$c_\theta(\mathbf{z}) = \|d - F_\theta(\mathbf{x})\|^2$$

▶ Estimation de densité

- ▶ $\mathbf{z} = \mathbf{x}$
- ▶ F_θ ensemble de fonctions réelles
- ▶ R : espérance
- ▶ C : vraisemblance

$$c_\theta(\mathbf{z}) = -\ln p_\theta(\mathbf{x})$$

Apprentissage supervisé

Préliminaires : gradient, regression, regression logistique

Modèles discriminants

- Réseaux de neurones et Deep Learning

- Machines à noyaux et SVM

- ▶ L'apprentissage supervisé recouvre un ensemble de problèmes génériques
 - ▶ Regression
 - ▶ Classification
 - ▶ Ranking
 - ▶ ...
- ▶ Dans la première partie du cours on examine des problèmes de régression et de classification

Optimisation

Algorithmes de gradient

- ▶ Objectif
 - ▶ Optimiser une fonction de coût $C(\mathbf{w})$ dépendant de paramètres \mathbf{w}
- ▶ Principe :
 - ▶ Initialiser \mathbf{w}
 - ▶ Itérer jusqu'à convergence
 - $\mathbf{w}(t + 1) = \mathbf{w}(t) + \epsilon(t)\mathbf{D}(t)$
 - ▶ la direction de descente \mathbf{D} , le pas de gradient ϵ sont déterminés à partir d'informations locales sur la fonction de coût $\mathbf{C}(\mathbf{w})$, i.e. approximations au 1er ou 2nd ordre.
- ▶ Exemple :

Gradient de la plus grande pente (batch)
Ensemble d'apprentissage
 $D = \{(x_1, d_1), \dots, (x_N, d_N)\}$

Initialiser \mathbf{w}_0
Itérer
 $\mathbf{w}(t + 1) = \mathbf{w}(t) - \epsilon \nabla_{\mathbf{w}} C(t)$
Critère d'arrêt
Avec $C = \sum_{i=1}^N c(x_i)$

Autres procédures de minimisation basées sur le gradient

- ▶ Approximation quadratique locale de la fonction à optimiser (on approxime C par une parabole)

$$C(w) = C(w_0) + (w - w_0)^T \nabla C(w_0) + \frac{1}{2} (w - w_0)^T H (w - w_0)$$

□ Avec H le Hessien de $C(\cdot)$: $H_{ij} = \frac{\partial^2 C}{\partial w_i \partial w_j}$

- ▶ En différentiant par rapport à w

$$\nabla C(w) = \nabla C(w_0) + H(w - w_0)$$

- ▶ On cherche le minimum de C

$$\nabla C(w) = 0$$

- ▶ Méthode de Newton

$$w = w_0 - H^{-1} \nabla C(w_0)$$

- ▶ Trop couteux en pratique ($O(n^3)$ pour l'inverse, + dérivées partielles)

- ▶ En pratique on utilise des gradients du 1er ordre ou des approximations du 2nd ordre
- ▶ Exemple approximation du 2nd ordre :
 - ▶ Méthodes de quasi-Newton : approximation de H^{-1} itérativement.
 - ▶ Forme générale :
$$w = w_0 - \varepsilon H_t' \nabla C(w_0)$$

$$H_{t+1}' = H_t' + F(H_t', w, w_0, \nabla C(w_0), \nabla C(w))$$
- ▶ H' : approximation de H^{-1} sans calculer les dérivées seconde

Regression

▶ Regression simple

▶ Objectif : prédire une fonction réelle

▶ Ensemble d'apprentissage

▶ $(\mathbf{x}^1, \mathbf{d}^1), \dots, (\mathbf{x}^N, \mathbf{d}^N)$

▶ $\mathbf{x} \in R^n, d \in R$: regression simple

▶ Modèle linéaire

▶ $F(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} = \sum_{i=0}^n w_i x_i$ avec $x_0 = 1$

▶ Fonction de coût

▶ Moindres carrés

$$\square C = \frac{1}{2} \sum_{i=1}^N (d^i - \mathbf{w} \cdot \mathbf{x}^i)^2$$

▶ Gradient de la plus grande pente

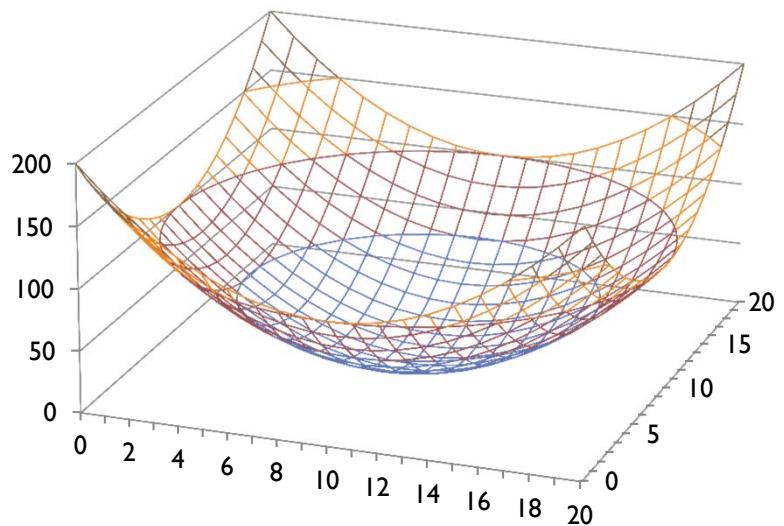
▶ $\mathbf{w} = \mathbf{w}(t) - \epsilon \nabla_{\mathbf{w}} C, \nabla_{\mathbf{w}} C = \left(\frac{\partial C}{\partial w_1}, \dots, \frac{\partial C}{\partial w_n} \right)^T$

▶ $\frac{\partial C}{\partial w_k} = \frac{1}{2} \sum_{i=1}^N \frac{\partial}{\partial w_k} (d^i - \mathbf{w} \cdot \mathbf{x}^i)^2 = - \sum_{i=1}^N (d^i - \mathbf{w} \cdot \mathbf{x}^i) x_k^i$

▶ $\mathbf{w} = \mathbf{w}(t) + \epsilon \sum_{i=1}^N (d^i - \mathbf{w} \cdot \mathbf{x}^i) \mathbf{x}^i$

Regression

► Géométrie des moindres carrés



Regression multi-variée

- ▶ Le modèle s'étend directement au cas où $d \in R^p$
 - ▶ On a alors p regressions linéaires indépendantes
 - ▶ L'algorithme est identique

Interprétation probabiliste

- ▶ Modèle statistique de la regression
 - ▶ $d = \mathbf{w} \cdot \mathbf{x} + \epsilon$, où ϵ est une v.a. qui modélise l'erreur
 - ▶ On suppose que ϵ est une v.a. i.i.d. gaussienne
 - ▶ $\epsilon \sim N(0, \sigma^2)$, $p(\epsilon) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\epsilon^2}{2\sigma^2}\right)$
 - ▶ La distribution a posteriori de d est alors $p(d | \mathbf{x}; \mathbf{w}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(d - \mathbf{w} \cdot \mathbf{x})^2}{2\sigma^2}\right)$
 - ▶ Vraisemblance
 - ▶ $L(\mathbf{w}) = \prod_{i=1}^N p(d^i | \mathbf{x}^i; \mathbf{w})$
 - C'est une fonction de \mathbf{w} , calculée pour un ensemble de données, ici les données d'apprentissage
 - ▶ Principe du maximum de vraisemblance
 - ▶ Choisir le paramètre \mathbf{w} qui maximise $L(\mathbf{w})$ ou toute fonction croissante de $L(\mathbf{w})$
 - ▶ on va maximiser la log vraisemblance $l(\mathbf{w}) = \log L(\mathbf{w})$ qui donne des calculs plus adaptés en pratique
 - ▶
$$l(\mathbf{w}) = N \log \left(\frac{1}{\sqrt{2\pi}\sigma} \right) - \frac{1}{2\sigma^2} \sum_{i=1}^N (d^i - \mathbf{w} \cdot \mathbf{x}^i)^2$$
 - ▶ On retrouve le critère des moindres carrés
 - ▶ Sous les hyp. précédentes, on a une interprétation probabiliste de la régression

Regression logistique

- ▶ La regression linéaire peut être utilisée pour des problèmes de regression ou de classification
- ▶ Un modèle mieux adapté pour la classification (avec des sorties binaires) est celui de la régression logistique

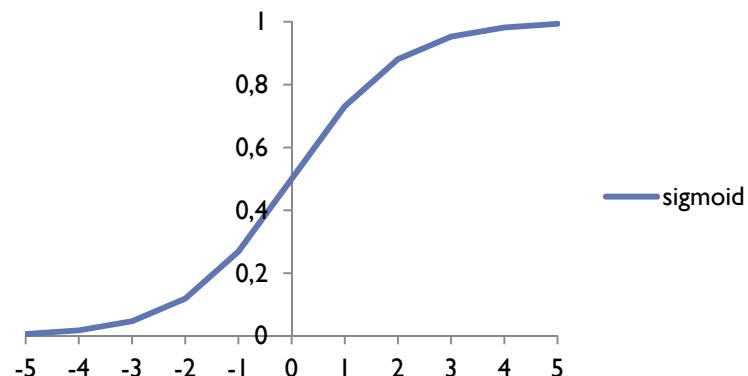
- ▶ $F_w(x) = g(w \cdot x) = \frac{1}{1+\exp(-w \cdot x)}$

- ▶ Fonction logistique

- ▶ La fonction $g(z) = \frac{1}{1+\exp(-z)}$ est appelée fonction logistique ou sigmoïde

- hint

- $g'(z) = g(z)(1 - g(z))$



Regression logistique

Interprétation probabiliste

- ▶ Comme $d \in \{0,1\}$, on va faire l'hypothèse d'une distribution conditionnelle de Bernouilli
 - ▶ $p(d = 1|x; w) = F_w(x)$ et $p(d = 0|x; w) = 1 - F_w(x)$
 - ▶ En plus compact
 - $p(d|x; w) = (F_w(x))^d (1 - F_w(x))^{1-d}$ avec $d \in \{0,1\}$
- ▶ Vraisemblance
 - ▶ $L(w) = \prod_{i=1}^N (F_w(x^i))^{d^i} (1 - F_w(x^i))^{1-d^i}$
- ▶ Log-vraisemblance
 - ▶ $l(w) = \sum_{i=1}^N d^i \log F_w(x^i) + (1 - d^i) \log(1 - F_w(x^i))$
 - qui est une entropie croisée
- ▶ Gradient de la plus grande pente
 - ▶ $\frac{\partial l(w)}{\partial w_k} = \sum_{i=1}^N (d^i - F_w(x^i)) x_k^i$
 - ▶ $\nabla_w l = \sum_{i=1}^N (d^i - F_w(x^i)) x^i$
 - ▶ Algorithme
 - $w = w - \epsilon \nabla_w C = w + \epsilon \sum_{i=1}^N (d^i - F_w(x^i)) x^i$

Regression logistique multivariée

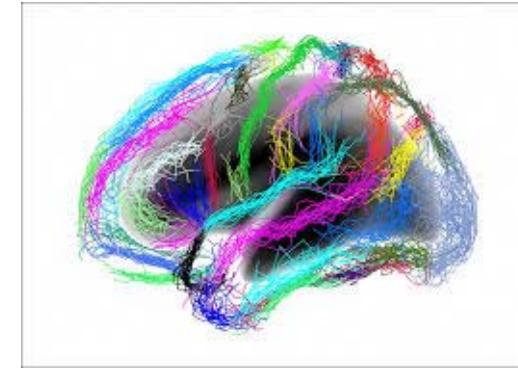
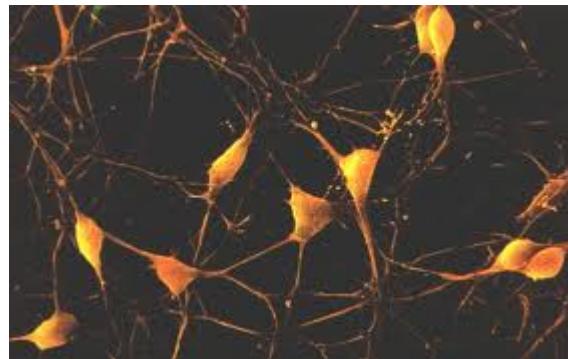
- ▶ On suppose que l'on a un problème de classification à plusieurs classes 1...p
- ▶ On code suivant la fonction indicatrice suivante
 - ▶ Classe 1: $d = (1, 0, \dots, 0)^T$
 - ▶ Classe 2 : $d = (0, 1, \dots, 0)^T$
 - ▶ ...
 - ▶ Classe I: $d = (0, 0, \dots, 1)^T$
 - ▶ On a un vecteur de sorties à p dimensions – codage “one out of p”
- ▶ La fonction $F_{\mathbf{W}}(\mathbf{x})$ est une fonction vectorielle à p dimensions
 - ▶ La composante i sera une **fonction softmax**
 - ▶
$$F_{\mathbf{W}}(\mathbf{x})_i = \frac{\exp(\mathbf{w}_i \cdot \mathbf{x})}{\sum_{j=1}^p \exp(\mathbf{w}_j \cdot \mathbf{x})}$$
 - Attention : ici $\mathbf{w}_j \in R^n$ est un vecteur
- ▶ Le modèle probabiliste sous jacent est un modèle multinomial pour les densités conditionnelles
 - ▶
$$p(\text{Classe} = i | \mathbf{x}; \mathbf{w}) = \frac{\exp(\mathbf{w}_i \cdot \mathbf{x})}{\sum_{j=1}^p \exp(\mathbf{w}_j \cdot \mathbf{x})}$$
- ▶ Algorithme d'apprentissage
 - ▶ Comme précédemment on peut utiliser un algorithme de gradient pour maximiser la log vraisemblance
 - ▶ Si le nombre de classes est très grand l'estimation du softmax est couteuse
 - ▶ Différentes techniques pour limiter la complexité du calcul (vu plus tard)

Apprentissage supervisé

Réseaux de neurones

Réseaux de neurones

- ▶ Les RN servent de métaphore pour réaliser des systèmes d'apprentissage



- ▶ Ils constituent un paradigme important en apprentissage statistique
 - ▶ Le cerveau humain est constitué par un réseau dense (10^{11}) d'unités simples, les neurones. Chaque neurone est connecté en moyenne à 10^4 neurones.
 - ▶ Concepts importants
 - ▶ Représentation et contrôle sont distribué
 - ▶ Apprentissage à partir d'exemples ou d'essais/ erreurs

Apports pluridisciplinaires

► Domaines

- ▶ Neurosciences
- ▶ Sciences cognitive (AI, psychologie, linguistique)
- ▶ Informatique
- ▶ Maths
- ▶ Physique

► Buts

- ▶ Modélisation (neurophysiologie, biologie.....)
- ▶ Modèle de calcul (applications, computational theory, apprentissage...)

Fondements biologiques

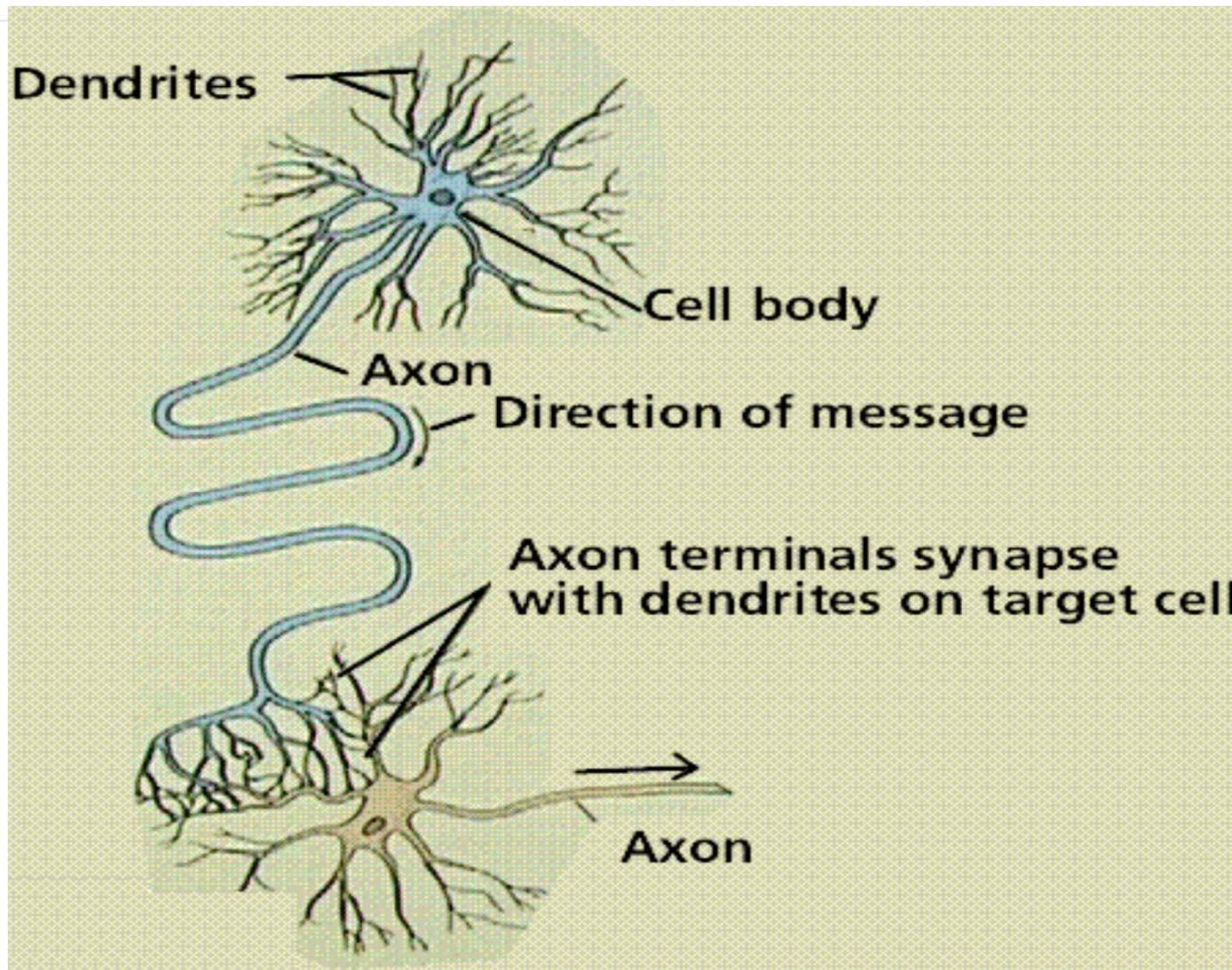
▶ Le neurone

- ▶ Soma
- ▶ Arbre des dendrites
- ▶ Axone
- ▶ Flot d'information
 - ▶ axone : impulsions électriques
 - ▶ dendrites : transmission chimique avec le soma via synapses

▶ Synapses

- ▶ contact : émission - réception
- ▶ Poids synaptique = modulation de l'information transmise vers le soma.
- ▶ Comportement du neurone + mémoire ?

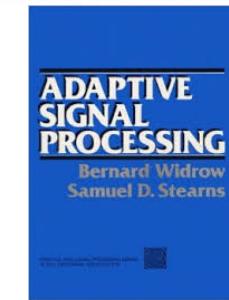
Composants du neurone



Historique rapide des réseaux de neurones

- ▶ 43 Mc Culloch & Pitts : neurone formel
 - ▶ "A logical calculus of the ideas immanent in nervous activities"
- ▶ 40 – 45
 - ▶ Wiener (USA)
 - ▶ Kolmogorov (URSS)
 - ▶ Türing (UK)
 - ▶ Théorie de l'estimation et de la prédiction (contrôle batteries anti-aériennes)
 - ▶ Boucle de rétro-action
- ▶ 48 – 50 Von Neuman : réseaux d'automates
- ▶ 49 Hebb : apprentissage dans les réseaux d'automates

- ▶ 55 – 60
 - ▶ Rosenblatt : Perceptron
- ▶ Widrow - Hoff : Adaline
- ▶ 70 – 80 Mémoires associatives, ART, SOM ...
- ▶ 90 – 95
 - ▶ Réseaux non linéaires
 - ▶ Réseaux de Hopfield, Machine de Boltzmann
 - ▶ Perceptron multicouches ...
- ▶ 2006 - ..
 - ▶ Deep neural networks, restricted Boltzmann machines, ...
 - ▶ Representation learning



Optimisation dans les RN - Algorithmes de gradient

▶ Principe :

- ▶ $\mathbf{w}(t + 1) = \mathbf{w}(t) + \epsilon(t)\mathbf{D}(t)$
- ▶ la direction de descente \mathbf{D} , le pas de gradient ϵ sont déterminés à partir d'informations locales sur la fonction de coût $C(\mathbf{w})$, i.e. approximations au 1er ou 2nd ordre.

▶ Exemples :

Gradient adaptatif (on line – stochastic gradient)
 $c(x(t))$ est le coût sur l'exemple $x(t)$
Historiquement les algorithmes de RN utilisent des gradients stochastiques

Initialiser w_0
Iterer
choisir un exemple $x(t)$
 $\mathbf{w}(t + 1) = \mathbf{w}(t) - \epsilon \nabla_{\mathbf{w}} c(x(t))$
Critère d'arrêt

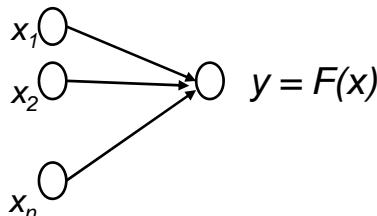
▶ Que l'on compare avec l'algorithme batch

Plus grande pente (batch)
 $C = \sum_{i=1}^N c(x^i)$ est le coût calculé sur l'ensemble des exemples d'apprentissage

Initialiser w_0
Iterer
 $\mathbf{w}(t + 1) = \mathbf{w}(t) - \epsilon \nabla_{\mathbf{w}} C(t)$
Critère d'arrêt

Le neurone formel

- ▶ C'est un automate caractérisé par
 - ▶ l'espace des signaux d'entrée $x = (x_1, \dots, x_n) \in R^n$
 - ▶ une fonction de transition $F_w(x)$
- ▶ Fonctions de transition
 - ▶ Historiquement, fonction à seuil (Neurone formel de McCulloch et Pitts)
 - ▶ En pratique on utilise des fonctions différentiables
 - Soit $a = \sum_{i=1}^n w_i x_i + w_0$, on considère des fonctions de la forme $F(x) = g(a)$
 - Fonction identité $g = Id$
 - Fonction sigmoïde $g(a) = \frac{1}{1+\exp(-ka)}$
 - Fonction tangente hyperbolique $g(a) = \frac{\exp(ka)-\exp(-ka)}{\exp(ka)+\exp(-ka)}$
 - On verra par la suite d'autres fonctions de transition



Adaline

Modèle

- ▶ Neurone linéaire
- ▶ Risque empirique : moindres carrés
 - ▶ $C = \frac{1}{N} \sum_{i=1}^N (w \cdot x^i - d^i)^2$
- ▶ On suppose que l'on est dans un contexte « en ligne »
 - ▶ Les données arrivent et sont traitées séquentiellement
 - ▶ Les paramètres sont modifiés pour chaque donnée
 - ▶ Algorithme d'apprentissage : gradient stochastique (Robbins-Monro, 1951), popularisé pour les réseaux de neurones par (Widrow et Hoff, 1959)

Adaline : règle de Widrow-Hoff

initialiser $w(0)$

Iterer

choisir un exemple $x(t)$

$$w(t + 1) = w(t) - \epsilon(t)(w(t) \cdot x(t) - d(t))x(t)$$

Critère d'arrêt

- $x(t)$ est l'exemple présenté à l'instant t , le gradient est calculé sur le coût local
- A comparer avec le gradient classique qui calcule le gradient sur le risque empirique

Plus grande pente

initialiser $w(0)$

Iterer

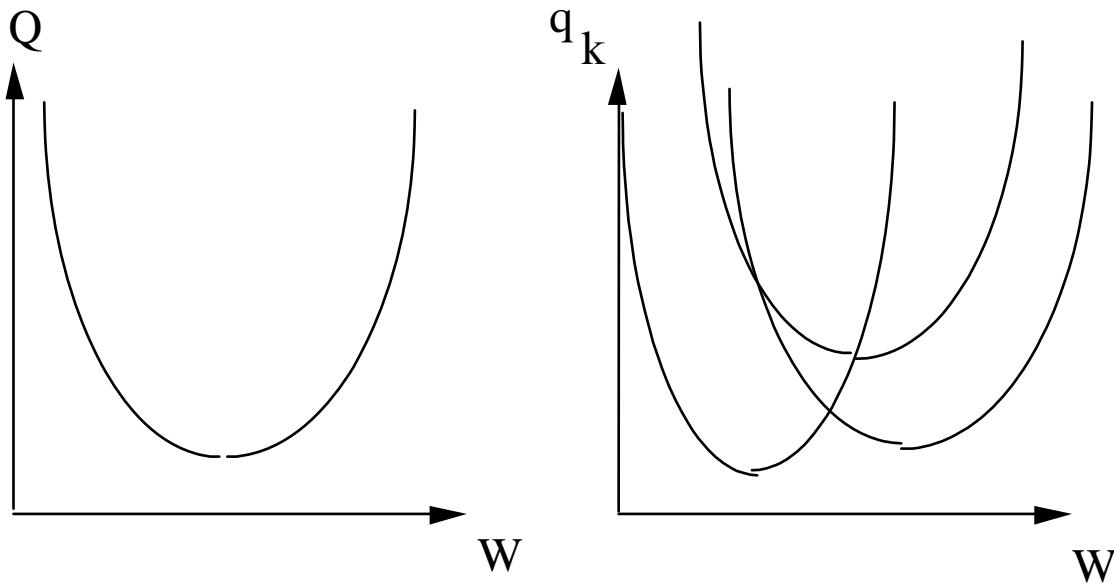
$$w(t + 1) = w(t) - \epsilon(t) \sum_{i=1}^N (w(t) \cdot x^i - d^i) x^i$$

Critère d'arrêt

- ▶ Apprentissage hors ligne vs apprentissage adaptatif
- ▶ c_k erreur sur la forme k de l'ensemble d'apprentissage

$$C = \frac{1}{N} \sum_k c_k$$

Gradient sur C Gradient adaptatif sur c



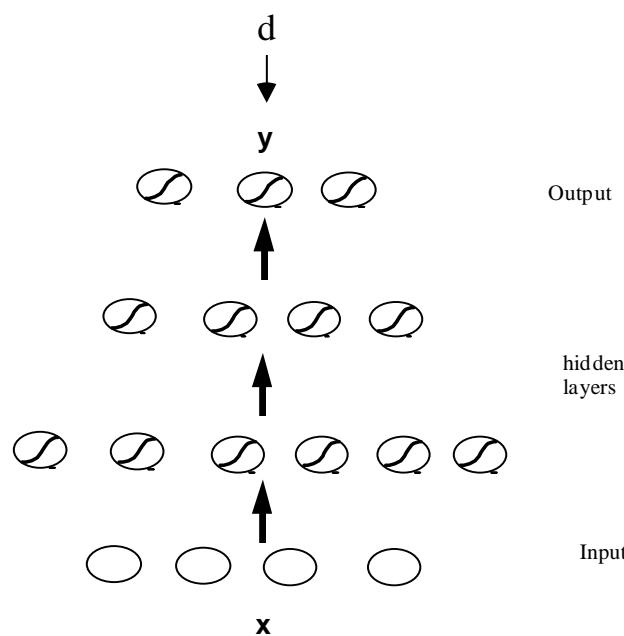
Optimisation dans les RN

Algorithmes de gradient

- ▶ Les algorithmes d'optimisation pour les réseaux de neurones utilisent souvent ces techniques simples de gradient stochastique
- ▶ Raisons
 - ▶ Historique
 - ▶ Complexité et taille de données
 - ▶ Ces méthodes sont devenues populaires également pour de nombreux autres modèles (complexité)
 - ▶ Ils exploitent bien la redondance présente dans les données
 - ▶ Bien adaptés aux grandes dimensions
 - ▶ En pratique utilisation d'algorithmes « mini-batch »
 - ▶ À chaque itération on considère un petit sous ensemble des données

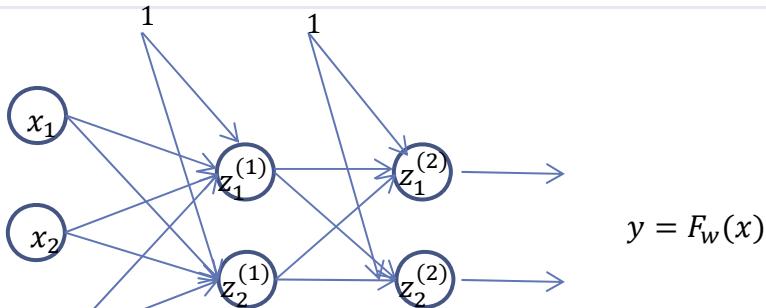
Perceptron Multicouches

- ▶ Réseau avec :
 - ▶ des couches externes: entrée et sortie
 - ▶ des couches internes dites cachées
 - ▶ Les cellules sur les couches internes et la couche de sortie sont des fonctions sigmoïdes ou tangente hyperbolique.



Perceptron Multicouches

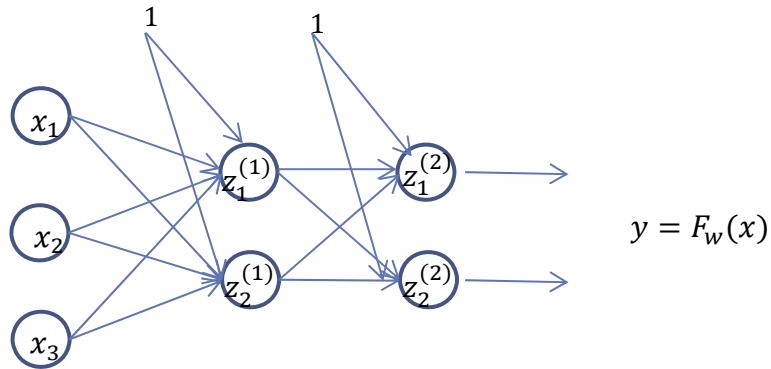
- ▶ Les exemples sont présentés séquentiellement
- ▶ Inférence : passe avant
 - ▶ Pour chaque exemple on calcule la sortie $y = F_w(x)$
- ▶ Apprentissage : passe arrière
 - ▶ On calcule l'erreur $(d - F_w(x))$ entre sortie désirée et sortie calculée.
 - ▶ On propage cette erreur pour calculer le gradient des poids du réseau



▶ Notations

- $\mathbf{z}^{(i)}$ vecteur activation de la couche i
- $z_j^{(i)}$ activation du neurone j de la couche i
- $W^{(i+1)}$ matrice de poids de la couche i à la couche $i+1$
- $w_{jk}^{(i)}$ poids de la cellule k sur la couche i à la cellule j sur la couche $i+1$
- \mathbf{y} vecteur de sortie
- $y_1 = z_1^{(2)} = g(w_{10}^{(2)} + w_{11}^{(2)}z_1^{(1)} + w_{12}^{(2)}z_2^{(1)})$
- $z_1^{(1)} = g(w_{10}^{(1)} + w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3)$
- $W^1 = \begin{pmatrix} w_{10}^{(1)} & w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{20}^{(1)} & w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{pmatrix}$

Inférence : passe avant



▶ Pour un exemple x

- ▶ On calcule en parallèle les activités de la couche de neurones 1
 - ▶ $a^{(1)} = W^{(1)}x$ puis $z^{(1)} = g(a^{(1)})$
 - Avec $g(a^{(1)}) = (g(a_1^{(1)}), g(a_2^{(1)}))^T$
 - ▶ On utilise les sorties de la couche 1 comme entrées de la couche 2 et on calcule en parallèle les activités de la couche 2
 - ▶ $a^{(2)} = W^{(2)}z^{(1)}$ puis $y = z^{(2)} = g(a^{(2)})$
 -

Apprentissage : passe arrière

- ▶ On décrit l'algorithme dans le cas d'une fonction de coût moindre carrés
 - ▶ On présente un exemple (x, d) , $x \in R^n$, $d \in R^p$
 - ▶ On calcule la sortie $y = F_W(x)$, $y \in R^p$
 - ▶ Calculer l'erreur $\delta = (y - d) = (y_1 - d_1, \dots, y_p - d_p)^T$
 - ▶ Rétropropager cette erreur de la couche de sortie vers la couche d'entrée :
 - $w_{ij}^{(k)} = w_{ij}^{(k)} + \Delta w_{ij}^{(k)}$ → mise à jour des poids par sgd
 - $\Delta w_{ij}^{(k)} = -\epsilon e_i^{(k+1)} z_j^{(k)}$ → gradient pour $w_{ij}^{(k)}$
 - C'est cette quantité « e » que l'on va rétropropager
 - $e_i^{(k+1)} = 2\delta_i g'(a_i^{(k+1)})$ si i est une cellule de sortie
 - $e_i^{(k+1)} = \sum_{h=1}^{NbCell(k+1)} w_{hi}^{(k)} e_h^{(k+1)} g'(a_i^{(k+1)})$ si i n'est pas une cellule de sortie

▶ Remarques

- ▶ Comme pour la passe avant, on modifie tous les poids d'une couche en parallèle
- ▶ C'est une implémentation particulière de l'algorithme du gradient stochastique qui évite de dupliquer les calculs
- ▶ Elle peut être adaptée à toute fonction différentiable
- ▶ Fonctions de coût
 - ▶ Pour des tâches de regression : LMSE
 - ▶ Pour des tâches de classification, différentes fonctions employées
 - Entropie croisée
 - Hinge, logistique (cf slide suivant)

Fonctions de coût

- ▶ Différentes fonctions de coût sont utilisées, suivant les problèmes, ou les modèles
- ▶ LMSE
 - ▶ Regression
 - ▶ Souvent utilisé en classification
- ▶ Classification, Hinge, logistique
 - ▶ Classification
 - ▶ Exemples
 - ▶ $y \in R^p, d \in \{-1,1\}^p$
- ▶ Hinge, logistique sont ici des approximations de l'erreur de classification

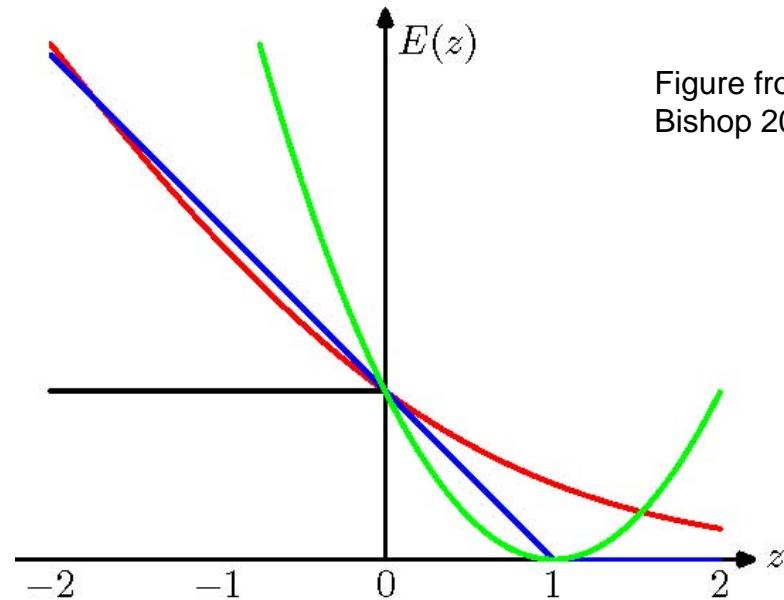


Figure from
Bishop 2006

En abscisse : $z = \mathbf{y} \cdot \mathbf{d}$ (marge)

$$C_{MSE}(\mathbf{y}, \mathbf{d}) = ||\mathbf{y} - \mathbf{d}||^2$$

$$C_{hinge}(\mathbf{y}, \mathbf{d}) = [1 - \mathbf{y} \cdot \mathbf{d}]_+ = \max(0, 1 - \mathbf{y} \cdot \mathbf{d})$$

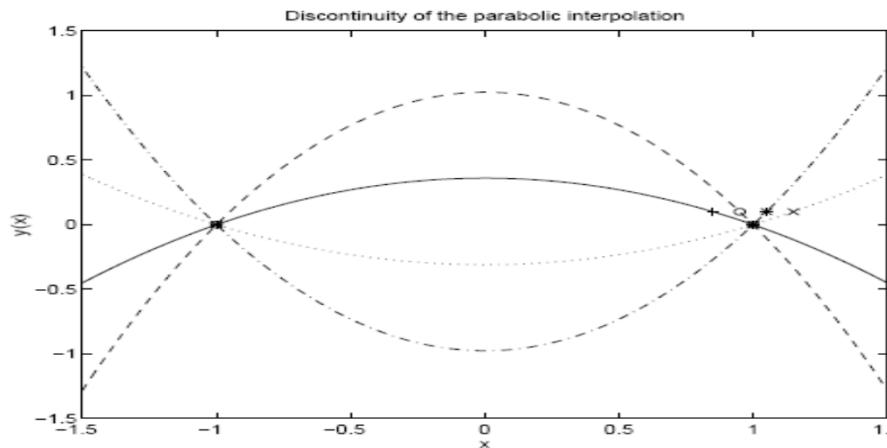
$$C_{logistique}(\mathbf{y}, \mathbf{d}) = \ln(1 + \exp(-\mathbf{y} \cdot \mathbf{d}))$$

Contrôle de la complexité

- ▶ En pratique, on n'optimise jamais le risque empirique seul
- ▶ On optimise le risque tout en contrôlant la complexité
 - ▶ cf partie théorique du cours
- ▶ Nombreuses méthodes
 - ▶ Régularisation (Hadamard ... Tikhonov)
 - ▶ Théorie des problèmes mal posés
 - ▶ Minimisation du risque structurel (Vapnik)
 - ▶ Estimateurs algébriques de l'erreur de généralisation (AIC, BIC, LOO, etc)
 - ▶ Apprentissage bayesien
 - ▶ Fournit une interprétation statistique de la régularisation
 - ▶ Le terme de régularisation apparaît comme un a priori sur les paramètres du modèle
 - ▶ Méthodes d'ensembles
 - ▶ Boosting, bagging, etc
 - ▶

Regularisation

- ▶ Hadamard
 - ▶ Un problème est bien posé si
 - ▶ Il existe une solution
 - ▶ Elle est unique
 - ▶ La solution est stable
 - ▶ Exemple de problème mal posé (Goutte 1997)



- ▶ Tikhonov
 - ▶ Propose des méthodes pour transformer un problème mal posé en problème bien posé

Régularisation

- ▶ Principe: Contrôler la variance de la solution en contraignant la fonctionnelle F
 - ▶ Optimiser $C = C_1 + \lambda C_2(F)$
 - ▶ C est un compromis entre
 - ▶ C_1 : mesure du but poursuivi e.g. MSE, Entropie, ...
 - ▶ C_2 : contraintes sur la forme de la solution (e.g. distribution des poids)
 - ▶ λ : poids de la contrainte
- ▶ Moindres carrés régularisés
 - ▶ On reprend le cas de la régression linéaire simple
 - ▶ $C = \frac{1}{N} \sum_{i=1}^N (d^i - w \cdot x^i)^2 + \frac{\lambda}{2} \sum_{j=1}^n |w_j|^q$
 - ▶ $q = 2$ régularisation L_2 , $q = 1$ régularisation L_1 connu aussi sous le nom de « Lasso »

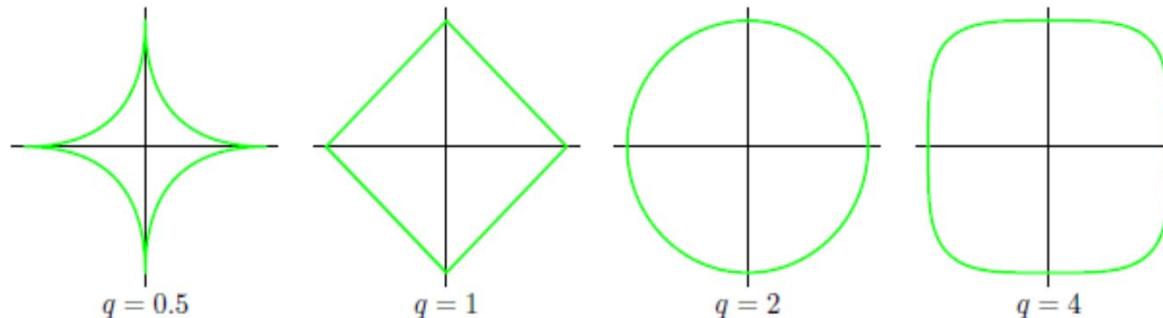


Fig. from Bishop 2006

Figure 3.3 Contours of the regularization term in (3.29) for various values of the parameter q .

▶ Résoudre

▶ $\text{Min}_{\mathbf{w}} C = \frac{1}{N} \sum_{i=1}^N (d^i - \mathbf{w} \cdot \mathbf{x}^i)^2 + \frac{\lambda}{2} \sum_{j=1}^n |w_j|^q, \lambda > 0$

▶ Revient à résoudre le problème d'optimisation sous contrainte

▶ $\text{Min}_{\mathbf{w}} C = \frac{1}{N} \sum_{i=1}^N (d^i - \mathbf{w} \cdot \mathbf{x}^i)^2$

▶ Sous contrainte $\sum_{j=1}^n |w_j|^q \leq s$ pour une certaine valeur de s

▶ Effet de cette contrainte

Figure 3.4 Plot of the contours of the unregularized error function (blue) along with the constraint region (3.30) for the quadratic regularizer $q = 2$ on the left and the lasso regularizer $q = 1$ on the right, in which the optimum value for the parameter vector \mathbf{w} is denoted by \mathbf{w}^* . The lasso gives a sparse solution in which $w_1^* = 0$.

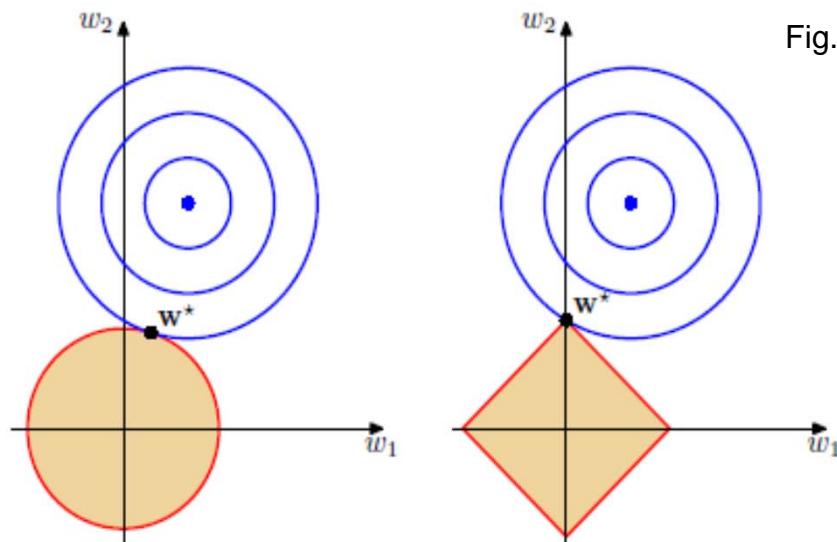


Fig. from Bishop 2006



▶ Penalisation L_2

▶ Coût

$$\triangleright C = C_1 + \lambda \sum_{j=1}^n |w_j|^2$$

▶ Gradient

$$\triangleright \nabla_w C = \lambda w + \nabla_w C_1$$

▶ Update

$$\triangleright w = w - \epsilon \nabla_w C = (1 - \epsilon \lambda) w - \epsilon \nabla_w C_1$$

▶ La pénalisation est proportionnelle à w

▶ Penalisation L_1

▶ Coût

$$\triangleright C = C_1 + \lambda \sum_{j=1}^n |w_j|^1$$

▶ Gradient

$$\triangleright \nabla_w C = \lambda sign(w) + \nabla_w C_1$$

▶ $sign(w)$ est le signe de w appliqué à chaque composante de w

▶ Update

$$\triangleright w = w - \epsilon \nabla_w C = w - \epsilon \lambda sign(w) - \epsilon \nabla_w C_1$$

▶ La pénalisation est constante avec un signe $sign(w)$

Régularisation empirique pour les réseaux de neurones

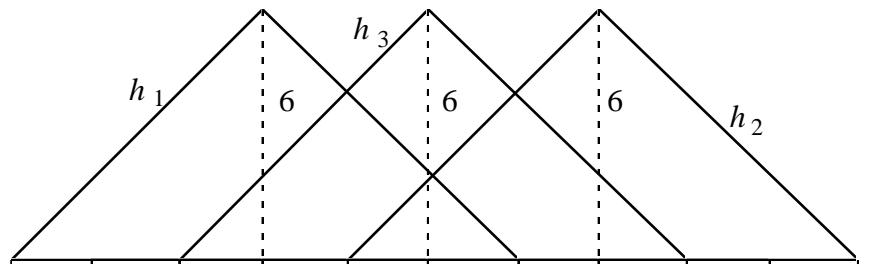
- ▶ $C_2 = \sum_{i=1}^n w_i^2$ → biaise la solution en diminuant les poids inutiles
 - ▶ $C_2 = \sum_{i=1}^n \frac{w_i^2}{1 + \frac{w_i^2}{c^2}}$ → 2 groupes de poids autour de c
 - ▶ $C_2 = \alpha \sum_{i=1}^n \frac{\frac{w_i^2}{c}}{1 + \frac{w_i^2}{c}} + (1 - \alpha) \sum_{i=1}^n \frac{\frac{h_i^2}{c}}{1 + \frac{h_i^2}{c}}$ → cellules cachées h + poids
-
- ▶ Utiliser des contraintes différentes suivant le rôle des poids
 - ▶ Problème : détermination des "hyper-paramètres"

Autres idées pour le problème de la généralisation dans les réseaux de neurones

- ▶ Arrêt de l'apprentissage
- ▶ Elagage : tuer les paramètres inutiles dans un réseau. Différentes mesures d'utilité ont été proposées
- ▶ Bruiter les entrées (Matsuoka 1992 ; Grandvallet et Canu 1994 ; Bishop 1994)
- ▶ Réseaux à convolution

Exemple (Cibas et al, 95, 96)

- ▶ Discriminer entre trois classes de "formes d'onde".
 - ▶ Les trois formes de base pour la génération des formes d'onde :



- ▶ 3 classes $C1^1, C2, C3^5$ engendrées respectivement par :

$$x = uh_1 + (1-u)h_2 + 2\epsilon$$

$$x = uh_1 + (1-u)h_3 + 10\epsilon$$

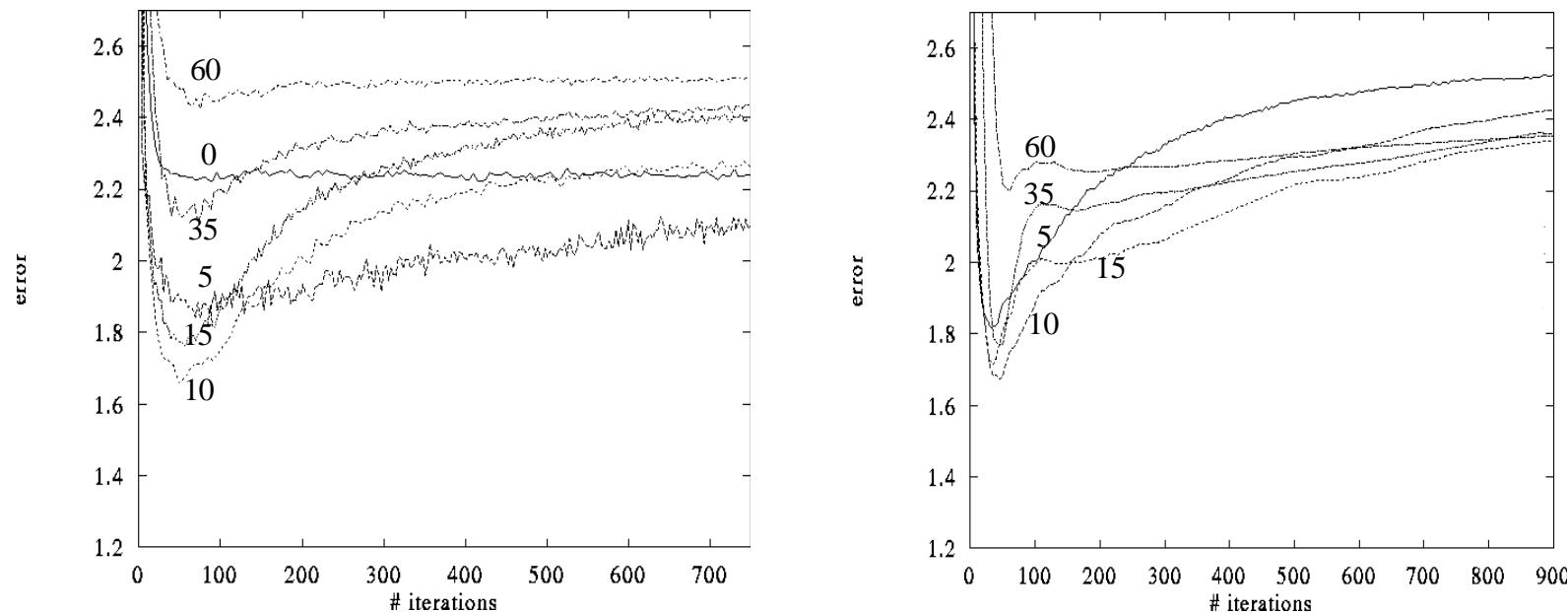
$$x = uh_2 + (1-u)h_3 + 26\epsilon$$

u v.a. de densité uniforme sur $[0,1]$, $\epsilon \sim N(0,1)$, Classes équiprobables

- ▶ Apprentissage = 10 ensembles disjoints, chacun de 300 exemples
- ▶ Test = 5000 exemples
- ▶ Algorithme : Rétropropagation

Evolution des performances pendant l'apprentissage

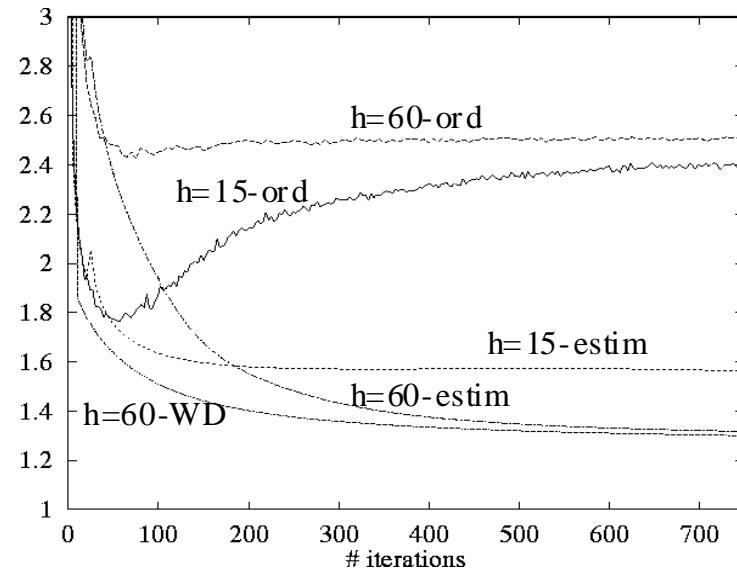
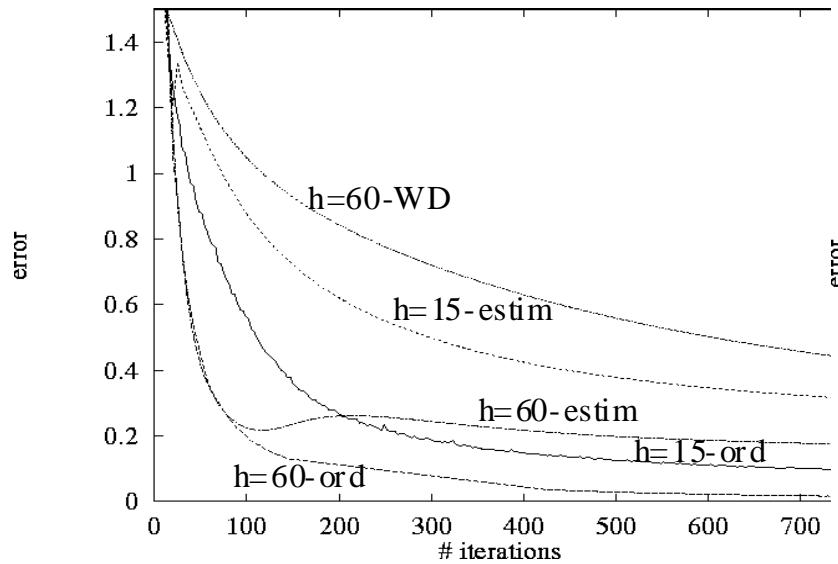
▶ Sans régularisation



□ Figure I a (left), b (right): evolution of the performances (mean square error) during training for MLPs with a varying number of hidden units. (a) corresponds to a stochastic gradient descent and (b) to a conjugate gradient. Each curve corresponds to a two weight layer MLP, the number on the curve gives the size of the hidden layer.

Evolution des performances pendant l'apprentissage

▶ Avec régularisation



- Comparaison de l'erreur en apprentissage (a) et en généralisation (b) pour les réseaux $h=15$ et $h=60$ en minimisant le coût ordinaire sans terme de régularisation (...-ord) et le coût avec la régularisation: avec détermination des paramètres à priori (...-WD) et en les estimant pendant l'apprentissage (...-estim)

Fonctions à Base Radiale

► Réseau à deux couches

► Notations

- $w_i.$ = poids vers la cellule i , x_i sortie de la cellule i , x entrée

Couche de sortie

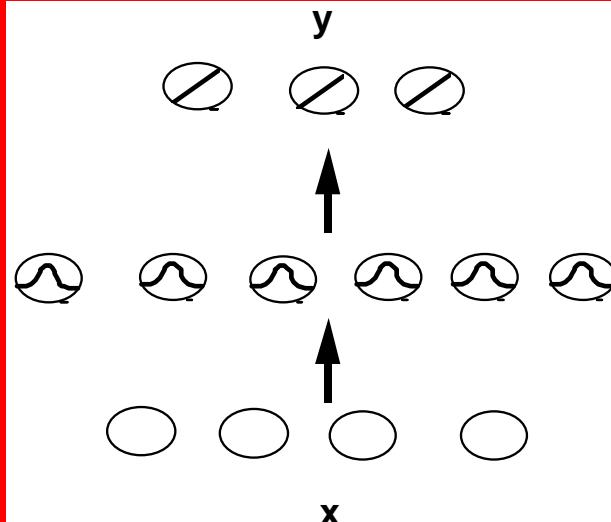
$$A = w_0 + \sum_j w_j x_j$$

$$g = \text{Id}$$

Couche intermédiaire

$$A = \|x - w\|^2$$

$$g(A) = e^{-\frac{A}{\sigma^2}}$$



- $y_i = G_i(x) = w_{i0} + \sum_j w_{ij} g\left(\|x - w_j\|^2\right)$
- Risque : moindres carrés

La fonction sigmoïde

► Distribution de la famille exponentielle :

$$p(x, \theta, \phi) = \exp((\theta^T x - b(\theta))/a(\phi) + c(x, \phi))$$

- ▶ q, f : paramètres de la loi, (q paramètre de position , f paramètre de dispersion).
- ▶ Ex. de distributions exponentielles : normale, gamma, binomiale, poisson, hypergéométrique ...
- ▶ Hypothèse : la distribution des données conditionnellement à chaque classe est de la famille exponentielle, avec un paramètre de dispersion identique pour toutes les classes i.e. :
- ▶ Alors

$$p(x/C_i) = \exp((\theta_i^T x - b(\theta_i))/a(\phi) + c(x, \phi))$$
$$P(C_i/x) = \frac{1}{1 + e^{-(w^T x + b)}}$$

Capacités d'approximation des PMC

- ▶ Résultats basés sur les théorèmes d'approximation de l'analyse fonctionnelle.
 - ▶ (Cybenko (1989))
 - ▶ Théorème 1 (regression): Soit f une fonction saturante continue, alors l'espace des fonctions de la forme $g(x) = \sum_{j=1}^n v_j f(\mathbf{w}_j \cdot \mathbf{x})$ est dense dans l'espace des fonctions continues sur le cube unité $C(I)$. i.e. $\forall h \in C(I)$ et $\forall \epsilon > 0$, $\exists g : |g(x) - h(x)| < \epsilon$ sur I
 - ▶ Théorème 2 (classification): Soit f une fonction saturante continue. Soit F une fonction de décision définissant une partition de I . Alors $\forall \epsilon > 0$, il existe une fonction de la forme $g(x) = \sum_{j=1}^n v_j f(\mathbf{w}_j \cdot \mathbf{x})$ et un ensemble $D \subset I$ tel que $\text{measure}(D) = 1 - \epsilon(D)$ $|g(x) - h(x)| < \epsilon$ sur D
 - ▶ .
 - ▶ (Hornik et al., 1989)
 - ▶ Théorème 3 : Pour toute fonction saturante croissante f , et toute mesure de probabilité m sur R^n , l'espace des fonctions de la forme $g(x) = \sum_{j=1}^n v_j f(\mathbf{w}_j \cdot \mathbf{x})$ est uniformément dense sur les compacts de $C(R^n)$ - espace des fonctions continues sur R^n

- ▶ *Fonctions radiales* (Park & Sandberg, 1993)
 - ▶ **Théorème 4 :** Si f , fonction réelle définie sur \mathbb{R}^n est intégrable, alors l'espace des fonctions de la forme :

$$g(x) = \sum_{j=1}^N v_j \cdot f\left(\frac{x - w_j}{\sigma_j}\right)$$

est dense dans $L^1(\mathbb{R}^n)$ ssi

$$\int_{\mathbb{R}^n} f(x) dx \neq 0$$

- ▶ Résultats basés sur le théorème de Kolmogorov
- ▶ Théorème sur la représentation (exacte) des fonctions réelles de Kolmogorov
 - ▶ Toute fonction h de $C(I)$ peut s'écrire sous la forme

$$h(x_1, \dots, x_n) = \sum_{q=1}^{2n+1} g_q \left(\sum_{p=1}^n f_{pq}(x_p) \right)$$

où les fonctions g et f sont des fonctions continues d'une variable.

- ▶ Théorème 6 (Kurkova 1992)
 - ▶ Soit h dans $C(I)$, $n \geq 2$ et $\varepsilon \in \mathbb{R}^+$, alors quelquesoient $m \in \mathbb{N}$ vérifiant

$$\begin{aligned} m &= 2n + 1 \\ n/(m-n) + v &< \varepsilon / \|h\| \\ \omega_h(1/m) &< v(m - n)/(2m - 3n) \\ v &> 0 \end{aligned}$$
 - ▶ h peut être approximée à une précision ε par un perceptron possédant deux couches cachées de fonctions saturantes et dont les sorties sont linéaires. La première couche comprend $n.m(m+1)$ unités et la seconde $m^2(m+1)n$. Les poids sont universels sauf ceux de la dernière couche, pour toutes les fonctions f vérifiant :

$$\omega_f(d) = \sup\{|f(x_1, \dots, x_n) - f(y_1, \dots, y_n)|, x, y \in I \text{ et } |x_p - y_p| < \delta \forall p\}.$$

Interprétation probabiliste des sorties

- ▶ Risque théorique $R = E_{x,d} [(d - h(x))^2]$
- ▶ Le min de R est obtenu pour $h^*(x) = E_d[d|x]$
- ▶ Le Risque théorique C pour la famille de fonction $F_w(x)$ se décompose de la façon suivante :
 - ▶ $C = E_{x,d}[(d - F_w(x))^2]$
 - ▶ $C = E_d[(d - E_d[d|x])^2] + E_{x,d}[(E_d[d|x] - F_w(x))^2]$
- ▶ Considérons $E_d[(d - E_d[d|x])^2]$
 - ▶ Ce terme est indépendant du modèle choisi et dépend uniquement des caractéristiques du problème (le bruit intrinsèque des données).
 - ▶ Il représente l'erreur minimum que l'on peut attendre sur ces données
 - ▶ $h^*(x) = E_d[d|x]$ est bien la solution optimale au problème $\text{Min}_h R = E_{x,d} [(d - h(x))^2]$
- ▶ Minimiser $E_{x,d} [(d - F_w(x))^2]$ est équivalent à minimiser $E_{x,d} [(E_d[d|x] - F_w(x))^2]$
 - ▶ La solution optimale $F_{w*}(x) = \text{argmin}_w E_{x,d} [(E_d[d|x] - F_w(x))^2]$ est la meilleure approximation au sens des moindres carrés de $E[d|x]$

Interprétation probabiliste des sorties

- ▶ Cas de la Classification
 - ▶ On considère la classification multi-classes avec un codage des sorties désirées «1 vs all »
 - ▶ i.e. $d = (0, \dots, 0, 1, 0, \dots, 0)$ avec un 1 en i ème position si la sortie désirée est la classe i et 0 partout ailleurs
 - ▶ $h_i^* = E_d[d|x] = 1 \cdot P(C_i|x) + 0 \cdot (1 - P(C_i|x)) = P(C_i|x)$
 - ▶ i.e. $F_{w^*}()$ est la meilleure approximation LMS de la fonction discriminante de Bayes (solution optimale pour la classification avec coûts 0/1).
 - ▶ de façon générale avec des sorties binaires
 - ▶ $h_i^* = P(d_i = 1|x)$
- ▶ L'importance de la précision sur les sorties dépend de l'utilisation
 - ▶ classification : la précision peut ne pas être importante
 - ▶ estimation de probabilité conditionnelle : elle l'est

Décomposition biais-variance

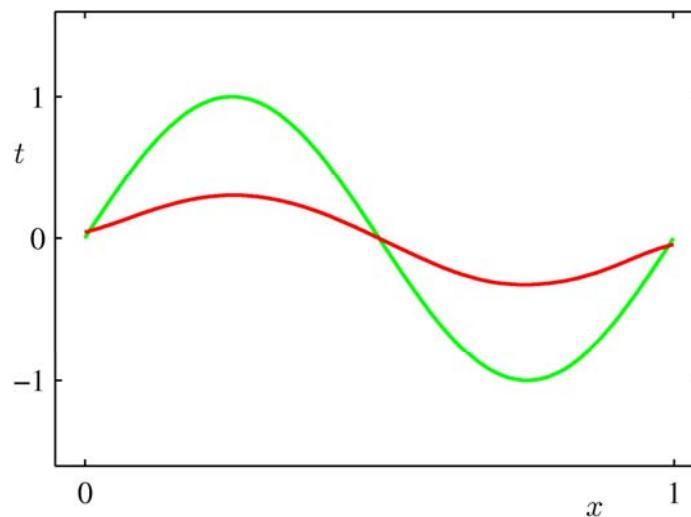
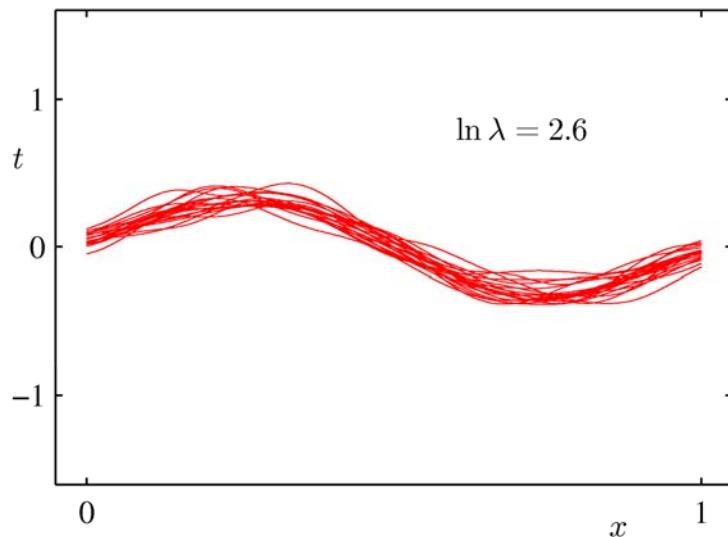
- ▶ Illustre la problématique du choix de modèle en mettant en évidence l'influence de la complexité du modèle
 - ▶ On rappelle la décomposition de l'espérance du coût quadratique
 - ▶ $E_{x,d} \left[(d - F_w(x))^2 \right] = E_d[(d - E_d[d|x])^2] + E_{x,d}[(E_d[d|x] - F_w(x))^2]$
 - ▶ On note $h^*(x) = E_d[d|x]$ la solution optimale à la minimisation de ce risque
 - ▶ En pratique on ne dispose pas d'une infinité de données permettant d'obtenir un bon estimateur de $E_d[d|x]$
 - ▶ L'estimateur obtenu dépendra de l'ensemble d'apprentissage D
 - ▶ L'incertitude sur l'estimateur liée au choix de l'ensemble d'apprentissage D peut être mesurée comme suit :
 - On tire une série d'ensembles d'apprentissage de taille N : D_1, D_2, \dots
 - apprend $F_w(x, D)$ sur chacun de ces ensembles
 - On mesure la moyenne des coût empiriques sur chacun de ces ensembles
 - ▶ Examinons l'intuition derrière cette procédure

Décomposition biais-variance

- ▶ On considère l'erreur quadratique $(F(x; D) - h^*(x))^2$ pour une donnée x et pour la solution $F_w(x; D)$ obtenue avec un ensemble d'apprentissage D
 - ▶ On note $E_D[F_w(x; D)]$ l'espérance sur D de ces solutions
- ▶ $(F(x; D) - h^*(x))^2$ se décompose en
 - ▶ $(F(x; D) - h^*(x))^2 = (F(x; D) - E_D[F_w(x; D)] + E_D[F_w(x; D)] - h^*(x))^2$
 - ▶ $(F(x; D) - h^*(x))^2 = (F(x; D) - E_D[F_w(x; D)])^2 + (E_D[F_w(x; D)] - h^*(x))^2 + 2(F(x; D) - E_D[F_w(x; D)])(E_D[F_w(x; D)] - h^*(x))$
- ▶ L'espérance de $(F(x; D) - h^*(x))^2$ par rapport à D se décompose en
 - ▶ $E_D[(F(x; D) - h^*(x))^2] = (E_D[F(x; D)] - h^*(x))^2 + E_D[F(x; D) - E_D[F_w(x; D)]]^2 = \text{biais}^2 + \text{variance}$
- ▶ Intuition
 - ▶ Choisir le bon modèle nécessite un compromis flexibilité – simplicité (cf régularisation)
 - ▶ Modèle flexible : faible biais – forte variance
 - ▶ Modèle simple : fort biais – faible variance

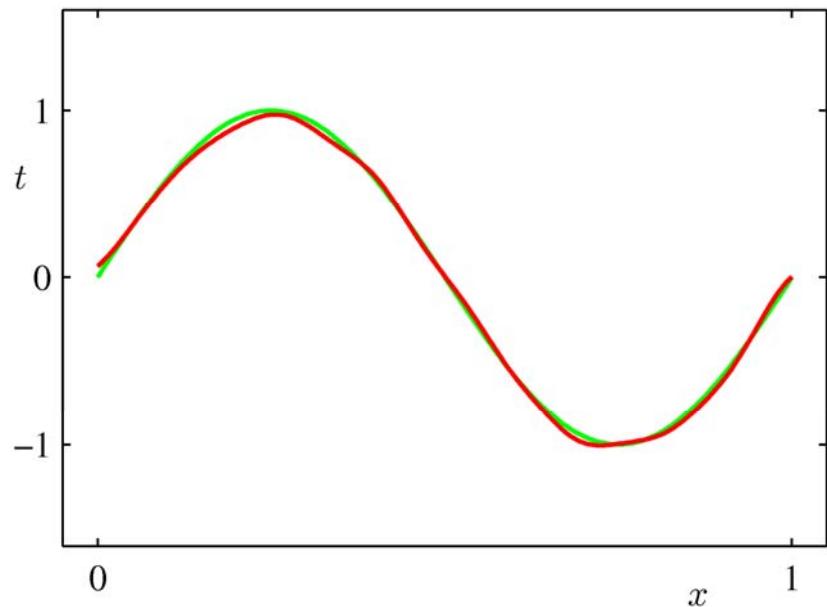
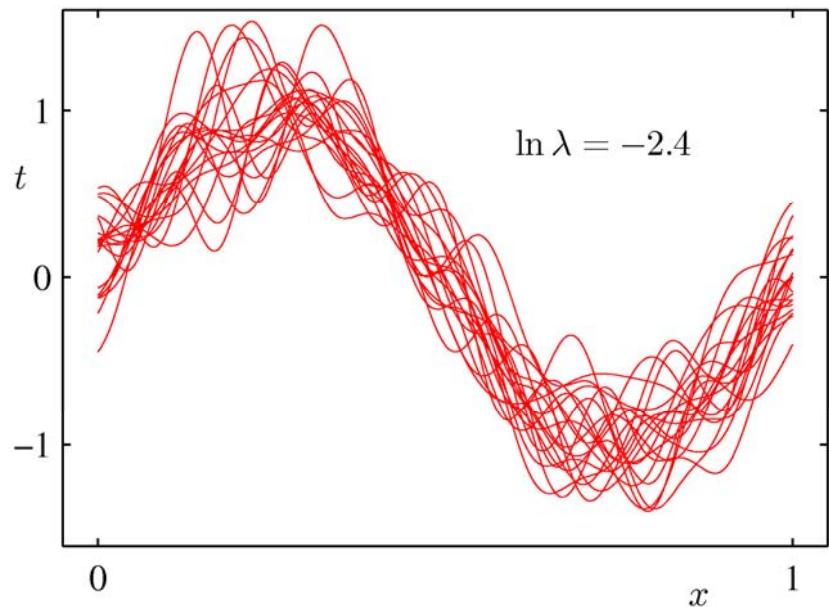
The Bias-Variance Decomposition (Bishop PRML)

- ▶ Example: 100 data sets from the sinusoidal, varying the degree of regularization
 - ▶ Model: gaussian basis function, Learning set size = 25, λ is the regularization parameter
 - ▶ Left 20 of the 100 models shown
 - ▶ Right : average of the 100 models (red), true sinusoid (green)



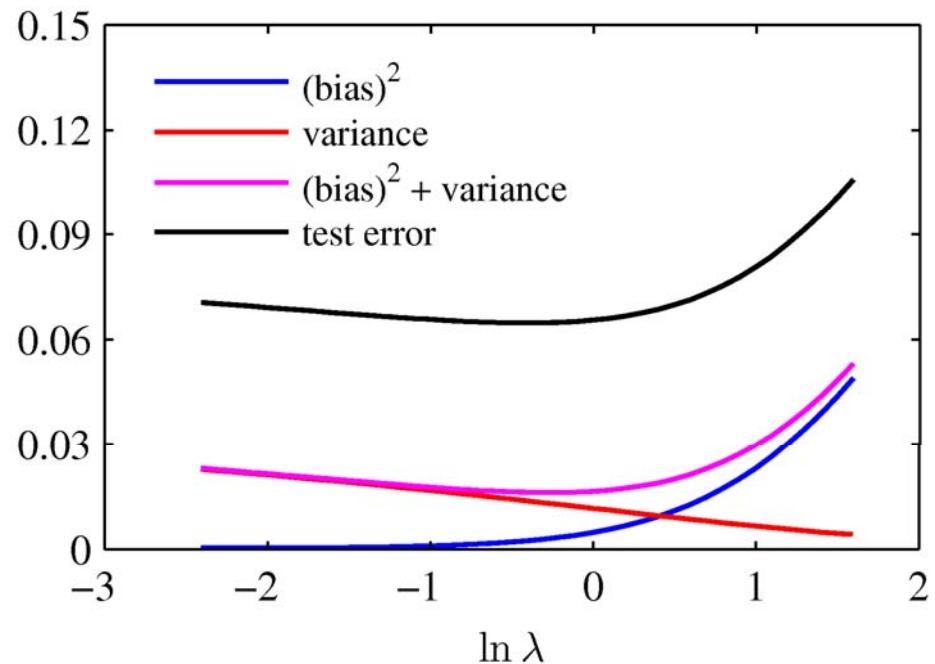
The Bias-Variance Decomposition (Bishop PRML)

- ▶ Example: 100 data sets from the sinusoidal, varying the degree of regularization



The Bias-Variance Decomposition (Bishop PRML)

- ▶ From these plots, we note that an over-regularized model (large λ) will have a high bias, while an under-regularized model (small λ) will have a high variance.





Deep Neural Networks and Representation Learning



Brief history

- ▶ Deep learning was popularized by Hinton 05 with a model called Restricted Boltzmann Machines
- ▶ Idea
 - ▶ Train several successive layers in an unsupervised way, stack them, stick a classifier at the last layer and do back propagation
- ▶ Developed by different actors Hinton 06, Bengio 06, LeCun 06, Ng 07
 - ▶ And many others now
- ▶ Success story
 - ▶ Very active domain, technology adopted by big actors (Google, Facebook, Msoft ..)
 - ▶ Success in many different academic benchmarks for a large series of problems
 - ▶ Image / scene labeling
 - ▶ Speech recognition
 - ▶ Natural language processing
 - ▶ Translation
 - ▶ etc

Motivations

- ▶ Learning representations
 - ▶ Handcrafted versus learned representation
 - ▶ Very often complex to define what are good representations
 - ▶ General methods that can be used for
 - ▶ different application domains
 - ▶ Multimodal data
 - ▶ Multi-task learning
 - ▶ Learning the latent factors behind the data generation
 - ▶ Unsupervised feature learning
 - ▶ Useful for learning data/ signal representations
- ▶ Several families of techniques
 - ▶ Matrix factorization, dictionary learning, compressed sensing
 - ▶ Latent probabilistic models
 - ▶ (Deep) Neural networks are the focus of this course

Complement to the NN course

More units

- ▶ In addition to the logistic or tanh units, already seen in the NN course, other forms may be used – some of the popular forms are:
 - ▶ Rectifier linear units
 - $g(x) = \max(0, b + \mathbf{w} \cdot \mathbf{x})$
 - Rectifier units allow to draw activations to 0 (used for sparse representations)
 - ▶ Maxout
 - $g(x) = \max_i(b_i + \mathbf{w}_i \cdot \mathbf{x})$
 - Generalizes the rectifier unit
 - There are multiple weight vectors for each unit
 - ▶ Softmax
 - ▶ Used for classification with a 1 out of p coding (p classes)
 - Ensures that the sum of predicted outputs sums to 1
 - $$g(x) = softmax(\mathbf{b} + \mathbf{W}\mathbf{x}) = \frac{e^{b_i + (\mathbf{W}\mathbf{x})_i}}{\sum_{j=1}^p e^{b_j + (\mathbf{W}\mathbf{x})_j}}$$

Mean square error and maximum likelihood

- ▶ Remember the following result concerning minimization of MSE
- ▶ Mean square loss $R = E[(d - h(x))^2]$
- ▶ Min of R is obtained for $h^*(x) = E_d[d|x]$
- ▶ MSE for a family of function $F_w(x)$
 - ▶ $C = E_{x,d}[(d - F_w(x))^2]$
 - ▶ $C = E_{x,d}[(d - E[d|x])^2] + E_{x,d}[(E[d|x] - F_w(x))^2]$
 - ⇒ $h^*(x) = E[d|x]$ is the optimal solution to $\text{Min}_w C$
 - ▶ $w^* / R(w^*) = \text{Min}_w C$ minimizes simultaneously:
 - ▶ $E_{x,d}[(d - F_w(x))^2]$ LMSE
 - ▶ $E_{x,d}[(E_d[d|x] - F_w(x))^2]$ best LMS approximation of $E[d|x]$

Mean Square error maximum likelihood

► Statistical model of the regression

- $d = F_w(x) + \epsilon$, where ϵ is a random variable modeling the error (variability)

► Let ϵ be a i.i.d. Gaussian random variable

- $\epsilon \sim N(0, \sigma^2)$, $p(\epsilon) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{\epsilon^2}{2\sigma^2})$

- The posterior distribution of d is $p(d | x; w) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{(d - F(x))^2}{2\sigma^2})$

► Likelihood

- $L(w) = \prod_{i=1}^N p(d^i | x^i; w)$

► Maximum likelihood

- Choose the parameter w maximizing $L(w)$ or alternatively any increasing function of $L(w)$ such as

► log – likelihood $l(w) = \log L(w)$

- $$l(w) = N \log \left(\frac{1}{\sqrt{2\pi}\sigma} \right) - \frac{1}{2\sigma^2} \sum_{i=1}^N (d^i - F(x^i))^2$$

► Under normal hypothesis on $p(d | x; w)$, maximizing the likelihood is equivalent to minimizing the MSE loss

Cross-entropy loss – 2 classes

► 2 classes classification problem

- ▶ Target is $d = 1$ if $x \in C_1, 0$ if $x \in C_2$
- ▶ We consider a discriminant function $y = F(x)$
 - ▶ Here we take, $F(x) = \frac{1}{1+\exp(-a)}$ with $a = w \cdot x + w_0$, i.e. **logistic function**
 - ▶ The posterior probability of a class is a Bernoulli distribution and can be written
 - ▶ $p(d|x) = y^d(1-y)^{1-d}$
 - d target output, y computed output
 - ▶ Likelihood
 - ▶ $\prod_{i=1}^N (y^i)^{d^i} (1-y^i)^{1-d^i}$
 - ▶ Loss: negative log-likelihood
 - ▶ $L = -\sum_{i=1}^N (d^i \log y^i + (1-d^i) \log(1-y^i)) = -\sum_{i=1}^N \log(p_\theta(d|x))$ for the Bernoulli distribution
 $y = p_\theta(d|x)$
 - ▶ This is the cross entropy between the target and computed posterior class distributions
 - ▶ The minimum is obtained for $y^i = d^i \quad \forall i = 1..N$
 - ▶ This cross entropy loss could be used also when the targets are not binary but in the range $[0,1]$
 - ▶ It can be shown that the optimal solution to the minimization of L is $F(x) = p(d=1|x)$ this means that (with the logistic function F) the learned y^i will approximate this posterior probability

Cross-entropy loss – multiple classes

- ▶ p mutually exclusive classes

- ▶ One considers a 1 out of p target coding scheme

- ▶ i.e. the target has a 1 at the position corresponding to the class and 0 elsewhere

- e.g. $d = (0,1,0, \dots, 0)$ if $x \in \text{Class 2}$

- $$p(d^i|x) = \prod_{k=1}^p (y_k^i)^{d_k^i}$$

- $$\log(p(d^i|x)) = \sum_{k=1}^p d_k^i y_k^i$$
 only 1 term is non 0 in this sum

- Negative log likelihood

- $$L = - \sum_{i=1}^N \sum_{k=1}^p d_k^i \log y_k^i$$

- ▶ Which is equivalent to $L = - \sum_{i=1}^N \log y_*^i$ with y_*^i the output corresponding to $d_*^i = 1$

- Minimum when $y_k^i = d_k^i \quad \forall i = 1 \dots N$

- Activation function: **softmax**

- ▶ The outputs must be in $[0,1]$ and sum to 1

- $$y_k = \frac{\exp(a_k)}{\sum_{k'=1}^p \exp(a_{k'})}$$
 with $a_k = \mathbf{w}_k \cdot \mathbf{x} + w_{k,0}$

- ▶ Again this formula is also suitable for targets in $[0,1]$ that sum to unity

- ▶ This is called the multinoulli distribution

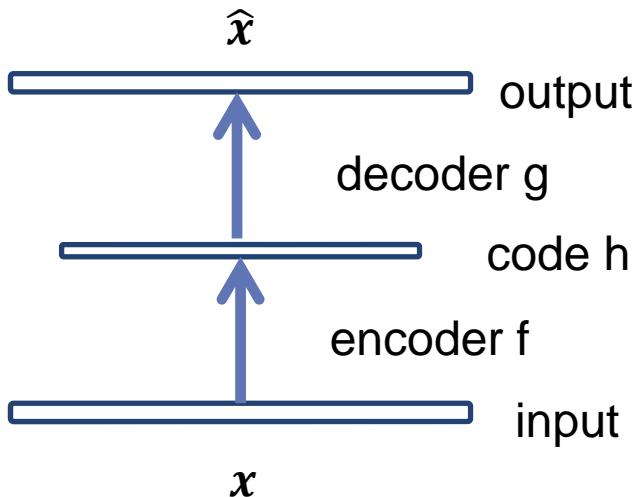
- ▶ y_k can be interpreted as the posterior probability of class k

- ▶ More generally, a probabilistic interpretation of NNs (and other parametric models) can be obtained by considering for the loss function a negative log-likelihood
 - ▶ $L(F(\mathbf{x}), d) = -\log(p(d|\mathbf{x}))$
 - ▶ If d is a continuous variable and we make the hypothesis that it is conditionally gaussian with mean $F(x)$, we get the MSE loss
 - ▶ If d is binary and we make the hypothesis that it is conditionally Bernoulli with probability $F(x) = p(d = 1|\mathbf{x})$ we get the cross entropy loss
- ▶ For multiple outputs
 - ▶ One has to specify the form of the joint distribution of the output variables conditionally on \mathbf{x}
 - ▶ e.g. conditional independence
 - ▶ $p(d_1, \dots, d_p | \mathbf{x}) = \prod_{i=1}^k p(d_i | \mathbf{x})$

Auto-encoders

Auto-encoders

- ▶ This is a neural network (MLP) trained to reproduce the inputs at the outputs
- ▶ Early attempts date back to the 80s with the auto-associative memories (Kohonen)
- ▶ Renewed interest since 2006 for learning representations and for deep learning
- ▶ Basic scheme



- ▶ Neurons in the hidden and output layers could be of any type
 - ▶ In the following one will assume hat units are sigmoid (but any differentiable function could be used)
 - ▶ Note that when using saturating functions like sigmoids, the x values shall be coded in a fixed interval $[0, 1]$ for sigmoids
 - ▶ Otherwise use linear ouput units

Auto-Encoders - Training

- ▶ Training set $\{x^1, \dots, x^N\}$
 - ▶ this is unsupervised learning
- ▶ The transition function of the AE is $F = g \circ f$
 - ▶ f is called the encoder, g is the decoder, $h = f(x)$ is the code of x
- ▶ Loss function
 - ▶ different functions can be used such as Mean Square Error or cross Entropy for example
- ▶ Algorithm
 - ▶ Auto-Encoders can be trained with back propagation, using the xs both as inputs and outputs
 - ▶ The auto-encoder will try to reproduce the input, i.e. to learn the identity function
- ▶ In order to avoid this trivial solution, one will impose constraints on the hidden layer
 - ▶ Undercomplete representations
 - ▶ One possible constraint is to limit the number of hidden units
 - ▶ The A-E will learn a compressed representation of the data by exploiting the correlation among the features, this is called **undercomplete** representation
 - ▶ rq: if all units are linear, AE trained with MSE performs a transformation f similar to PCA, i.e. it will learn the projection that maximizes the variance of the data (without the orthogonal property, of PCA but this can be easily solved)
 - ▶ Overcomplete representations
 - ▶ More recently, other constraints have been used with a hidden layer that may be even larger than the input layer
 - ▶ When the representation is larger than the input space, it is called **overcomplete** representation

Learning overcomplete representations

- ▶ In order to learn overcomplete representations different constraints have been proposed either on the form of the learned F function or on the hidden units
 - ▶ **Sparse auto-encoders**
 - ▶ A sparse representation is a representation such that for a given x , only a small number of the values in \mathbf{h} are non zero
 - ▶ Sparse representations have some desirable properties
 - e.g. it allows the mapping F to respond differently to different regions in the input space, i.e. hidden units specialize on some input regions (see manifold interpretation)
 - This corresponds to learning local mappings
 - This is usually implemented through constraints operating on the hidden units
 - ▶ **Contractive auto-encoders**
 - ▶ this encourages the derivative $\frac{\partial h}{\partial x}$ to be close to 0, so that \mathbf{h} is not sensible to variations in \mathbf{x} except in the directions where this is meaningful
 - ▶ **Prior on the distribution of hidden units**
 - ▶ Under a probabilistic interpretation of the NN, priors can play the role of constraints
 - ▶ **Dropout**
 - ▶ Specific heuristic effective for large NN

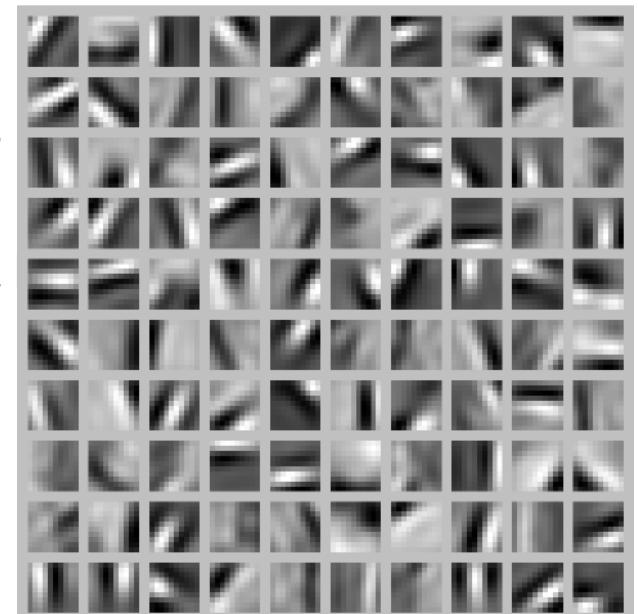
Sparsity for hidden units representation

- ▶ Sparsity is often imposed using a regularization term in the loss function
 - ▶ $C = C_1 + \lambda C_2(F)$
 - ▶ With C_2 constraining the activation of \mathbf{h} towards 0
- ▶ Different penalization terms may be used to enforce sparsity
 - ▶ L_1 norm $C_2 = |\mathbf{h}|_1$
 - ▶ Kullback Leibler divergence penalty
$$C_2 = -\sum_{i=1}^{\text{card}(\mathbf{h})} s \log(\hat{h}_i) + (1-s) \log(1 - \hat{h}_i)$$
 - ▶ The summation is over the hidden units
 - ▶ the hidden units are assumed sigmoid (values in $[0,1]$) and h_i is the activation of hidden unit i
 - ▶ \hat{h}_i is the mean value of h_i over the training examples
 - ▶ s is a **sparsity parameter**, typically set at a low value e.g. 0.05
 - ▶ Kullback Leibler divergence between 2 discrete distributions P and Q is a dissimilarity measure between these 2 distributions
 - $$KL(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$
 - ▶ $KL(s||h) = s \log(\frac{s}{h}) + (1-s) \log(\frac{1-s}{1-h})$ is KL divergence between 2 Bernoulli distributions of means s et h
 - ▶ Kullback Leibler divergence penalty forces the mean value of the hidden units to be close to s and thus to have a low mean value

- ▶ Derivatives for back propagation
 - ▶ Note that these sparsity terms are defined on the activations of the training data
 - ▶ Computing them thus requires a forward pass through the training set in order to compute the required average values
 - ▶ Then these mean values can be used for computing the gradient of these loss function

Visualizing hidden units activities

- ▶ One way to understand what a hidden unit (or any unit) is doing is to look at its weight vector
- ▶ We consider here unit activations based on dot products like sigmoid or tanh units
- ▶ Example (image from A. Ng cite)
 - ▶ Auto-encoder trained on a small set of 10x10 images
 - ▶ 100 hidden units
 - ▶ Each square represents the (normalized) weights of a
 - ▶ Grey value indicates the weight value
 - ▶ Each hidden unit has learned a feature detector
 - ▶ Here mostly image edges (gradient) detectors at different angles and positions

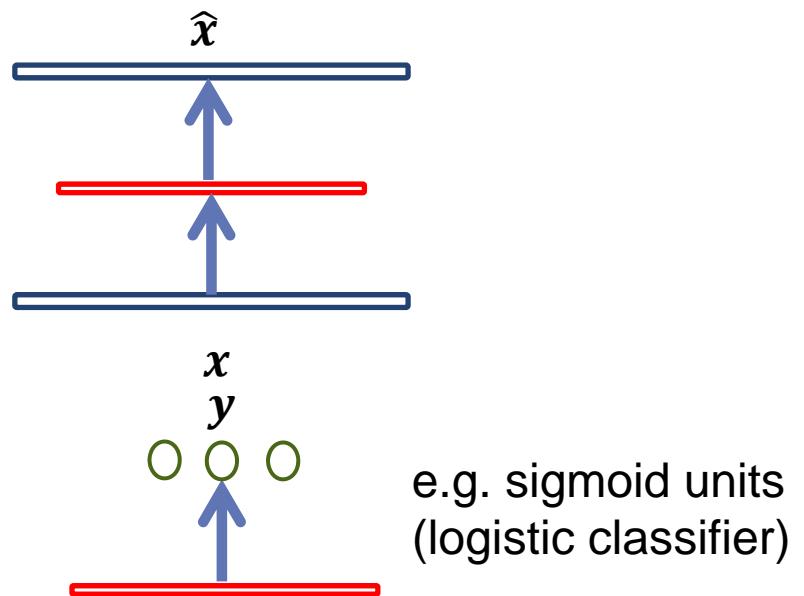


Probabilistic interpretation of auto-encoders

- ▶ The loss criteria can be given a probabilistic interpretation
- ▶ If we make the hypothesis of the existence of a conditional output distribution, the training objective is
 - ▶ $L = -\log p(x|F(x))$
 - ▶ If the conditional distribution is gaussian we get MSE, if it is a Bernoulli, we get the cross entropy loss
- ▶ For sparse auto-encoders
 - ▶ A prior distribution on \mathbf{h} the hidden unit vector, corresponds to a regularization term for the loss
 - ▶ e.g. Laplace prior on \mathbf{h} corresponds to the L_1 regularization on the hidden units activity
 - ▶ $L = -\log p(x|F(x)) + \lambda \sum_{i=1}^{n_h} |h_i| = -\log p(x|g(h)) + \lambda \sum_{i=1}^{n_h} |h_i|$
 - ▶ Note that all penalty terms do not need to have a prior interpretation (e.g. cross entropy on \mathbf{h})

Combining auto-encoders with a classifier

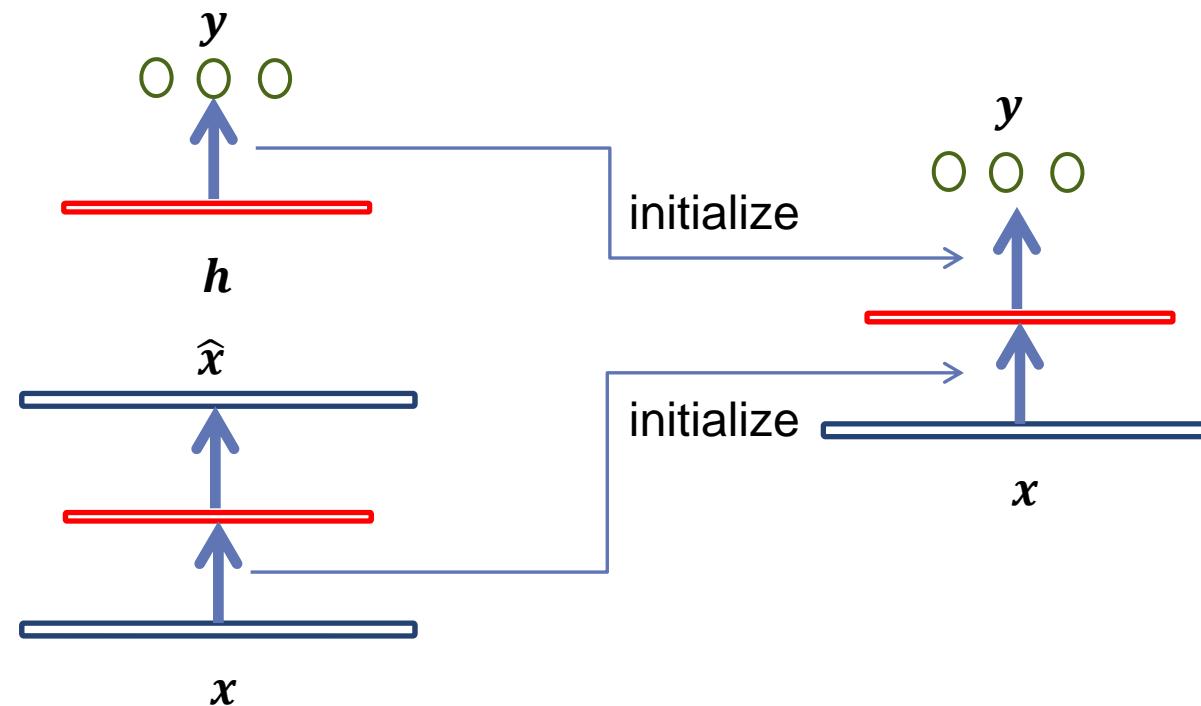
- ▶ The features learned can be used as inputs to any other system
- ▶ Instead of using the input features x use $\hat{h} = f(x)$
- ▶ Example for classification
 - ▶ 1. train an autoencoder
 - ▶ $\{x^1, \dots, x^N\} \rightarrow \{h^1, \dots, h^N\}$
 - ▶ 2. use the codes to train a classifier
 - ▶ $\{(h^1, d^1), \dots, (h^N, d^N)\}$
 - ▶ Assuming here a 3 class problem
- ▶ Both NN (encoder and classifier) can be trained separately using stochastic gradient descent



Join training of the auto-encoder and the classifier

► Fine tuning

- Separate training of each layer can be used as initialization
- Then stack the coding and classification layers one upon the other and fine tune the new NN using back propagation as in a classical MLP

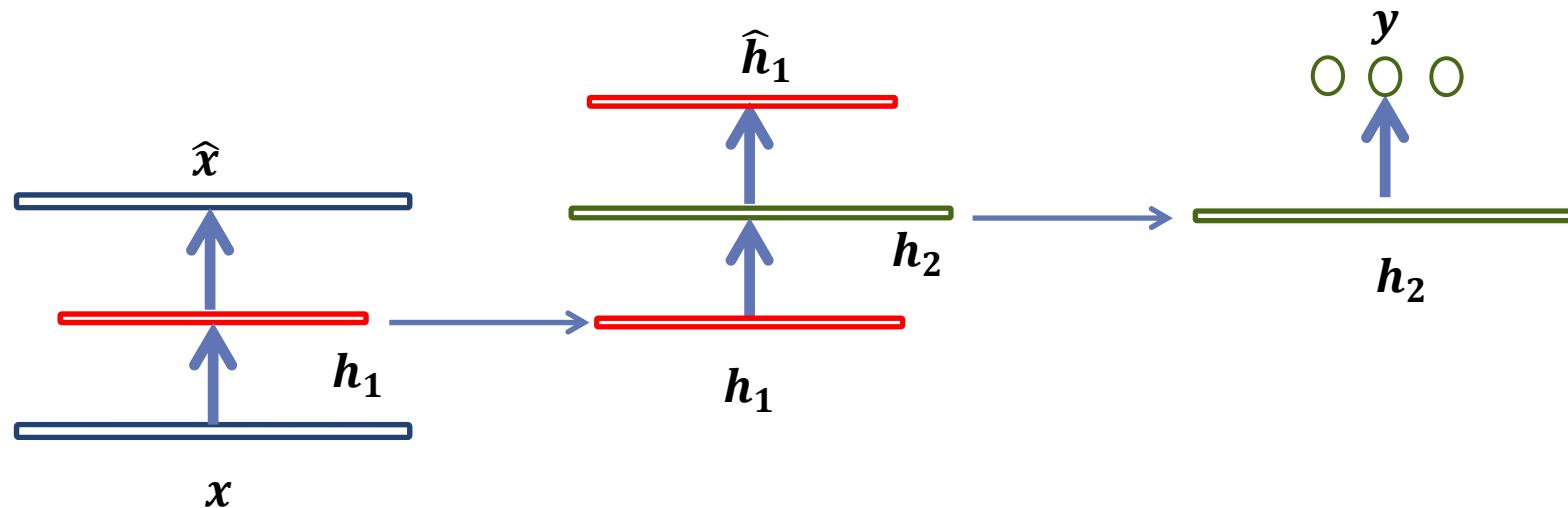


Deep auto-associators

- ▶ Idea
 - ▶ train successive layers of auto-associators in the hope of learning more complex features of the inputs
- ▶ Strategy (greedy layer wise training)
 - ▶ Each auto-associator will be trained using the representation layer of a previous auto-associator
 - ▶ The encoding layer of the different networks will then be stacked
 - ▶ After that fine tuning can be performed as before for a classification task
- ▶ Why not learning all the layers at once
 - ▶ Vanishing gradient
 - The gradients propagated backwards diminish rapidly in magnitude so that the first layers will not learn much
 - Non linear form of the loss function: many local minima

Deep architectures

- ▶ autoassociators can be stacked
 - ▶ Plus fine tuning using the last hidden layer if this is relevant for the task

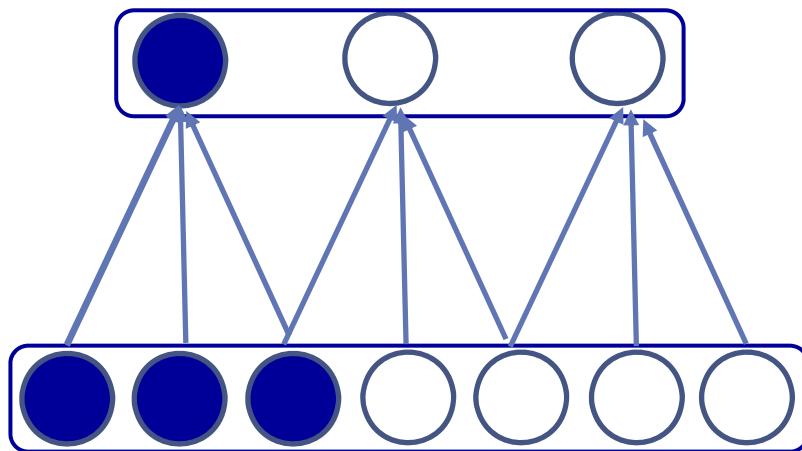


- ▶ Why using auto-association
 - ▶ Unsupervised learning
 - ▶ Often easy to get very large quantities of unlabeled data
 - ▶ Allows the extraction of « meaningful » features
 - ▶ These features can be used as preprocessing for other learning machines (e.g. classification) and for other datasets (same type of data but with different characteristics or another density distribution)
 - ▶ e.g extract meaningful image feature extractors on a large dataset that can be used for other image datasets
 - ▶ Supervised fine tuning
 - ▶ Can be used with other datasets when labels are available for the latter
 - ▶ Today, there exists procedures for training deep NN architectures from scratch

CNN: Convolutional Neural Nets

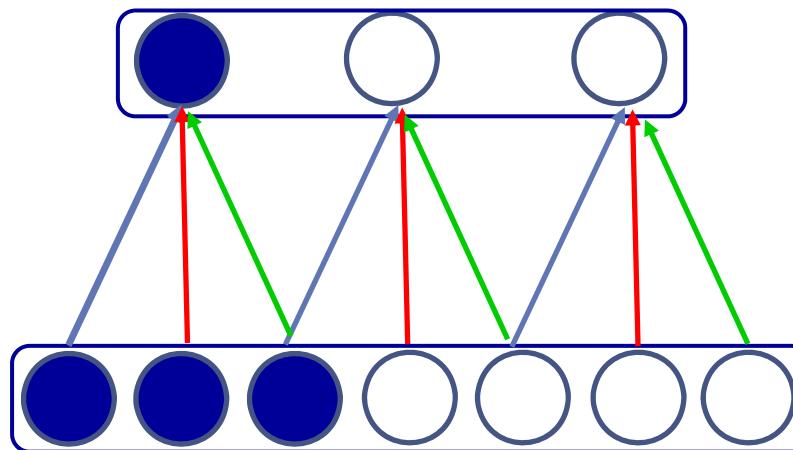
Local connections

- ▶ In many cases, data exhibit local properties, i.e. short term or space dependencies between characteristics
 - ▶ e.g. images, speech signal
- ▶ Instead of computing global statistics via fully connected layers, one may use local connections
 - ▶ This also allows the reduction of the number of parameters



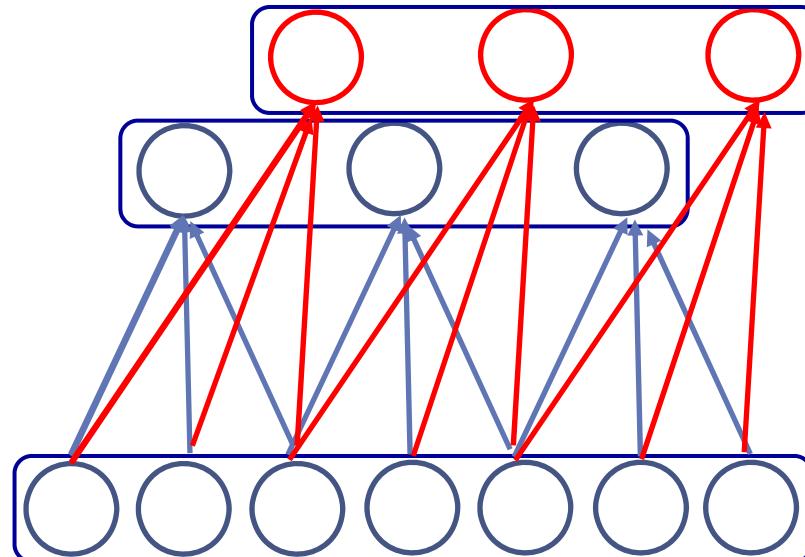
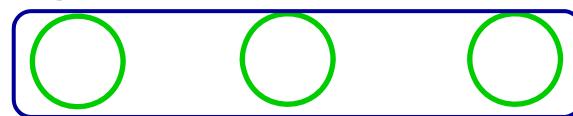
Shared connections: convolutions

- ▶ Local connections may be constrained to have the same weight vector
 - ▶ This is called shared weights
 - ▶ This is equivalent to convolve a unique weight vector with the input signal
 - ▶ Think of a local edge detector for images
 - ▶ This reduces even more the number of free parameters
 - ▶ In the figure, colors indicate a weight value, here the 3 hidden units share the same weight vector



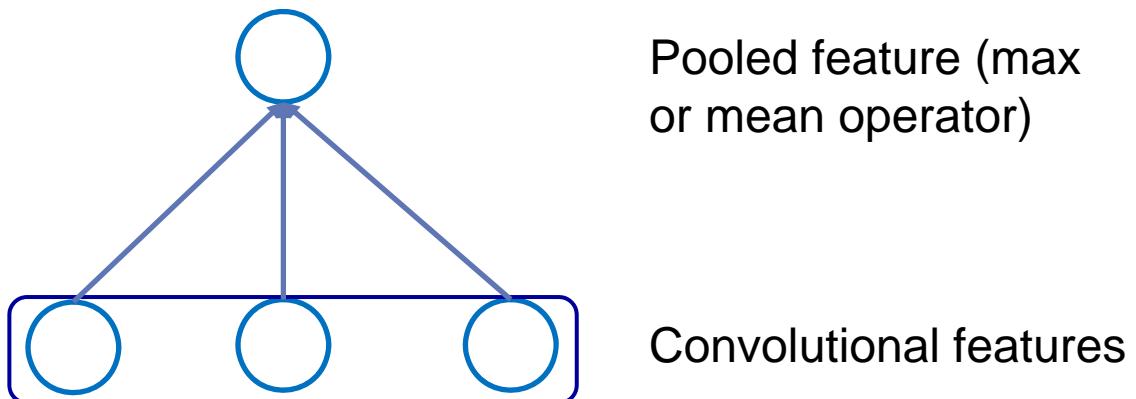
Shared connections: convolutions

- ▶ Several convolution filters can be learned simultaneously
- ▶ This corresponds to applying a set of local filters on the input signal
 - ▶ e.g edge detectors at different angles for an image
 - ▶ Here colors indicate similar weight vectors



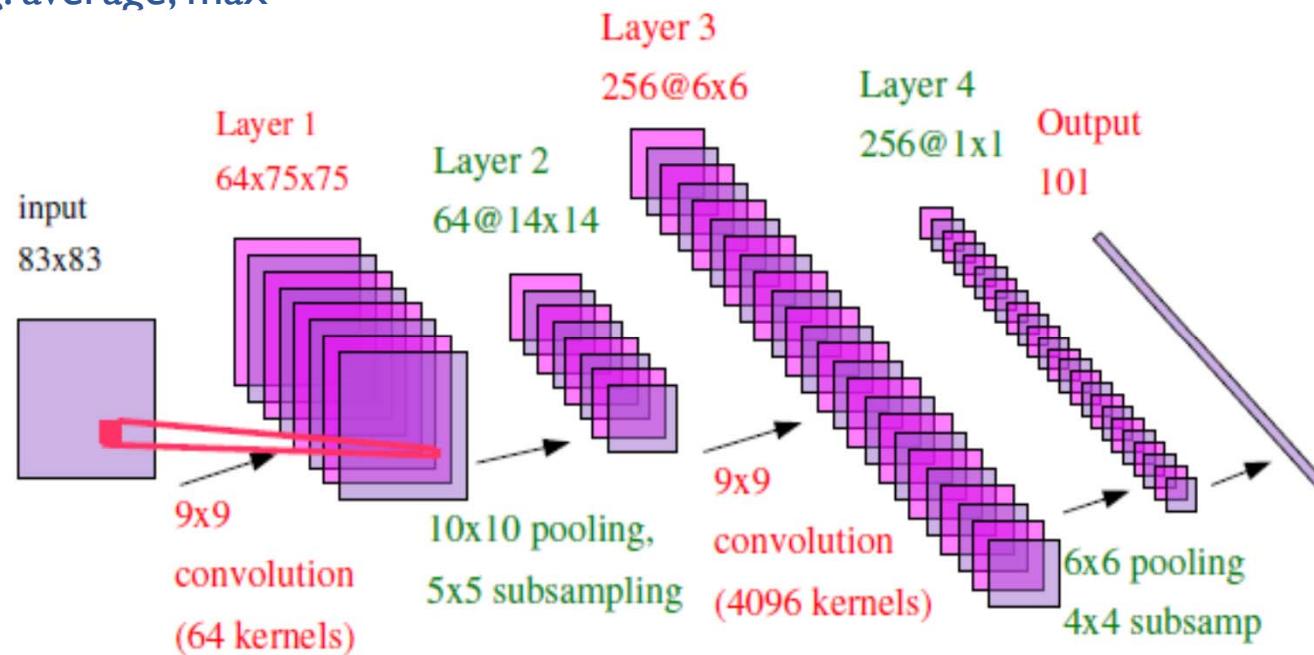
Pooling

- ▶ The number of extracted features on the hidden layer can be very large if one uses several filters
- ▶ In order to reduce this number, one may use pooling
 - ▶ This consists in aggregating statistics for these features over a region of the signal
 - ▶ Pooling operator : max, mean of the input units
 - ▶ For an image or a spatial signal, pooling of features generated by the same convolutional filter brings some invariance to small translations of the input signal



Convolutional nets

- ▶ ConvNet architecture (Y. LeCun since 1988)
 - ▶ Deployed e.g. at Bell Labs in 1989-90
 - ▶ Character recognition
 - ▶ Convolution: non linear embedding in high dimension
 - ▶ Pooling: average, max

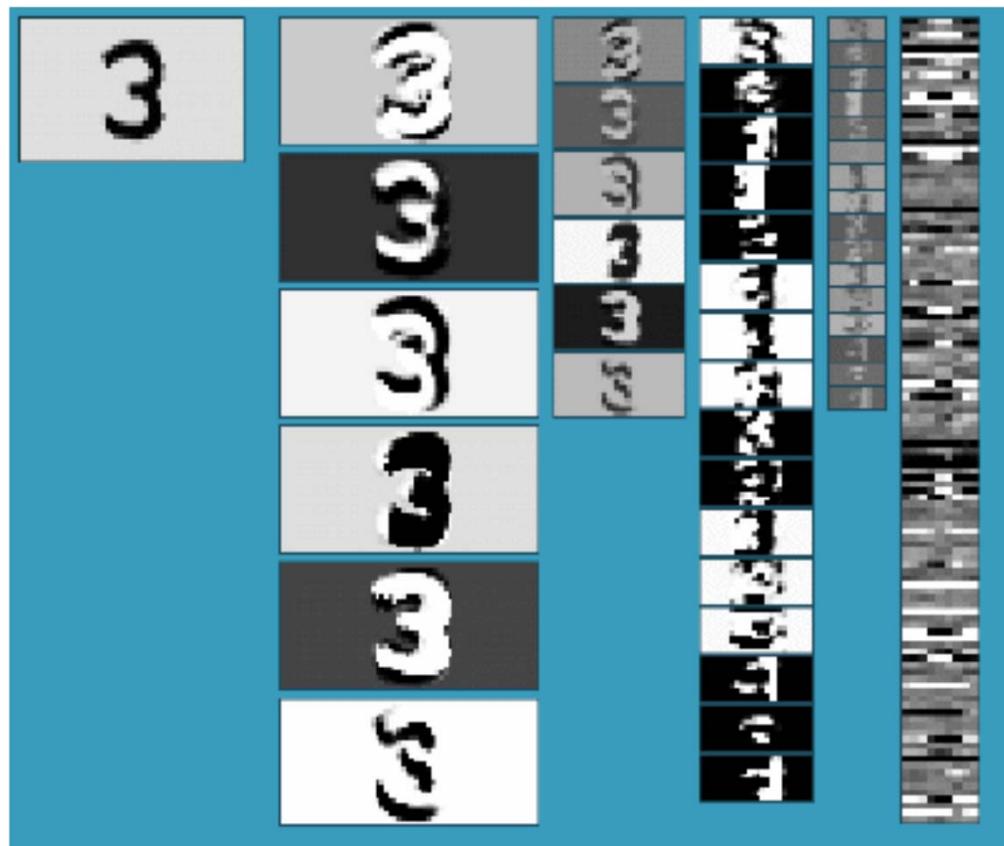


▶ In Convnet

- ▶ The first hidden layer consists in 64 different convolution kernels over the initial input, resulting in 64 different mapping of the input
- ▶ The second hidden layer is a sub-sampling layer with 1 pooling transformation applied to each matrix representation of the first hidden layer
- ▶ etc
- ▶ Last layer is a classification layer

Convolutional nets - visualization

- ▶ Hand writing recognition (Y. LeCun Bell labs 1989)



Convolutional nets (Krizhevsky et al. 2012)

- ▶ A landmark in object recognition
- ▶ Imagenet competition
 - ▶ Large Scale Visual Recognition Challenge
 - ▶ 1000 categories, 1.5 Million labeled training samples
 - ▶ Method: large convolutional net
 - ▶ 650K neurons, 630M synapses, 60M parameters

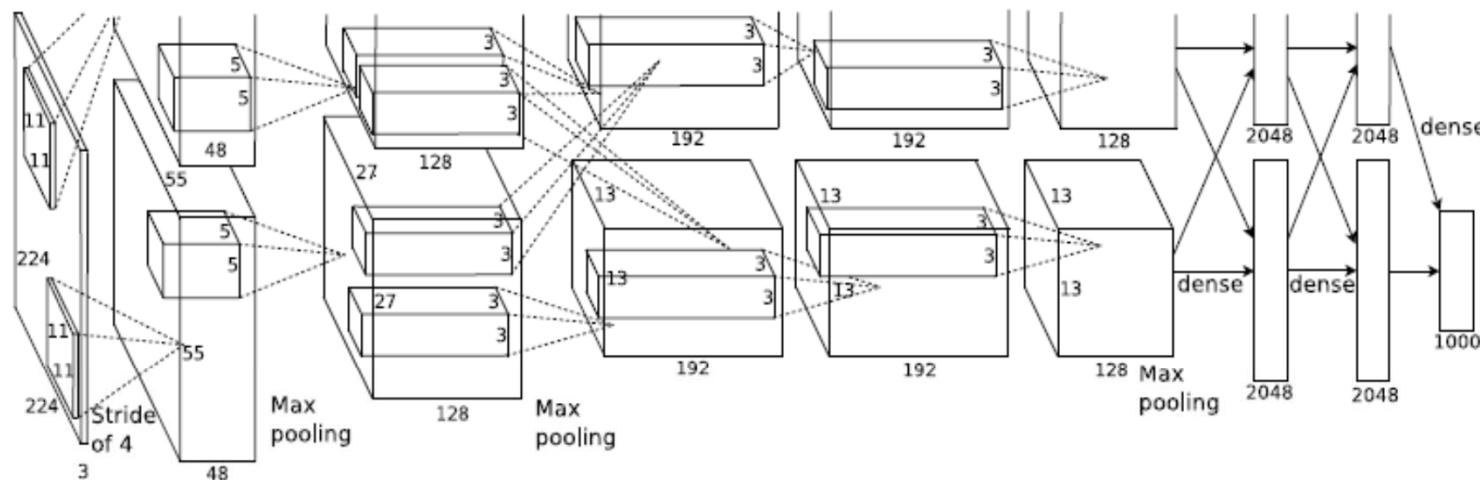


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Convolutional nets (Krizhevsky et al. 2012)

► Details of learned weights

► Labels

- True label under the image
- Predicted label probability for the 5 most probable labels

Lecun's demo

<https://youtu.be/ZJMtDRbqH40>

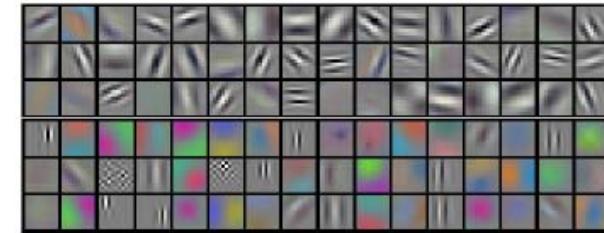
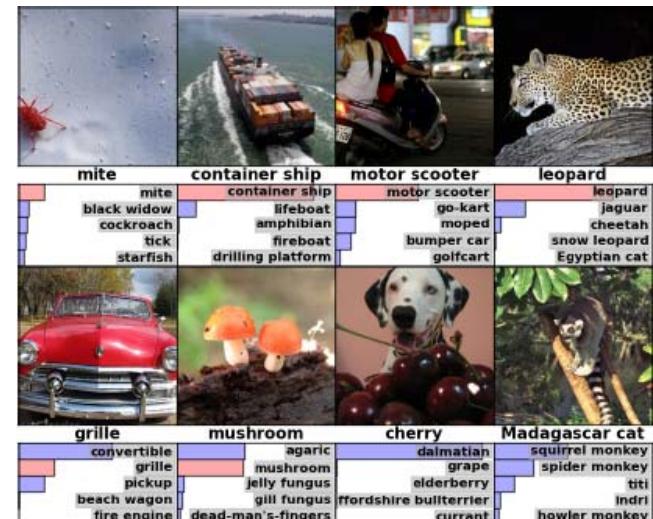


Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU



Learning word vectors (Collobert et al. JMLR 2011)

Learning word vector representations

(Mikolov et al. 2013a, 2013b)

► Goal

- ▶ Learn robust vector representation of words that can be used in different Natural Language Processing or Information retrieval tasks
- ▶ Learn word representations in phrase contexts
- ▶ Learn using **very** large text corpora
- ▶ Learn efficient, low complexity transformations

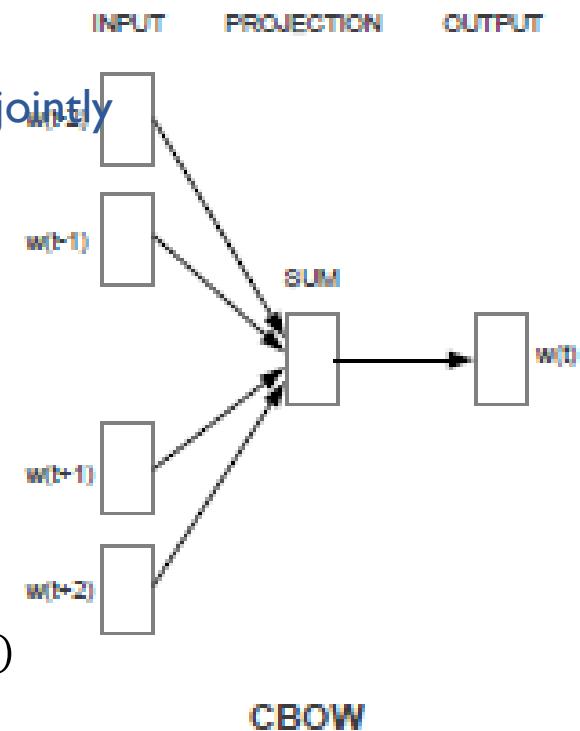
Learning word vector representations

(Mikolov et al. 2013a, 2013b)

▶ CBOW model

▶ Task

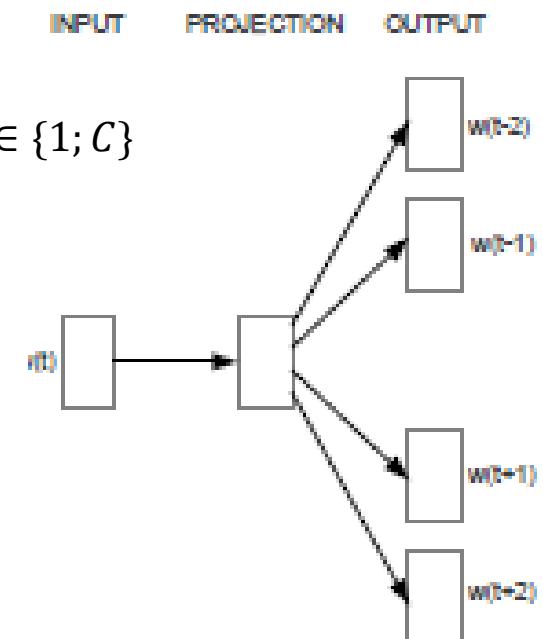
- ▶ Predict the middle word of a sequence of words
- ▶ Input and output word representations are learned jointly
 - ▶ (random initialization)
- ▶ The projection layer is linear followed by a sigmoid
- ▶ Word weight vectors in the projection layer are shared (all the weight vectors are the same)
- ▶ The output layer computes a hierarchical softmax
 - ▶ See later
 - ▶ This allows computing the output in $O(\log_2(\text{dictionary size}))$ instead of $O(\text{dictionary size})$
- ▶ The context is typically 4 words before and 4 after



Learning word vector representations (Mikolov et al. 2013a, 2013b)

▶ Skip Gram model

- ▶ Similar to the CBOW model, except that the context is predicted from the central word instead of the reverse
- ▶ Input and outputs have different representations for the same word
- ▶ The output is computed using a hierarchical softmax classifier
- ▶ Output words are sampled less frequently if they are far from the input word
 - ▶ i.e. if the context is $C = 5$ words each side, one selects $R \in \{1; C\}$ and use R words for the output context



Learning word vector representations (Mikolov et al. 2013a, 2013b)

▶ Skip gram model

- ▶ Loss average log probability
- ▶ $L = \frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t)$
 - ▶ Where T is the number of words in the whole sequence used for training (roughly number of words in the corpus) and c is the context size
- ▶ $p(w_{out} | w_{in}) = \frac{\exp(\nu_{w_{out}} \cdot \nu_{w_{in}})}{\sum_{w=1}^V \exp(\nu_w \cdot \nu_{w_{in}})}$
 - ▶ Where ν_w is the learned representation of the w vector (the hidden layer), $\nu_{w_{out}} \cdot \nu_{w_{in}}$ is a dot product and V is the vocabulary size
 - ▶ Note that computing this softmax function is impractical since it is proportional to the size of the vocabulary
 - ▶ In practice, this can be reduced to a complexity proportional to $\log_2 V$ using a binary tree structure for computing the softmax
 - Other alternatives are possible to compute the softmax in a reasonable time

Learning word vector representations

(Mikolov et al. 2013a, 2013b)

► Properties

- ▶ « analogical reasoning »
- ▶ This model learns analogical relationships between terms in the representation space
 - ▶ i.e. term pairs that share similar relations are share a similar geometric transformation in the representation space
 - ▶ Example for the relation « capital of »
 - ▶ In the vector space
 - Paris – France + Italy = Rome
 - At least approximatively
 - i.e. Rome is the nearest vector to Paris – France + Italy

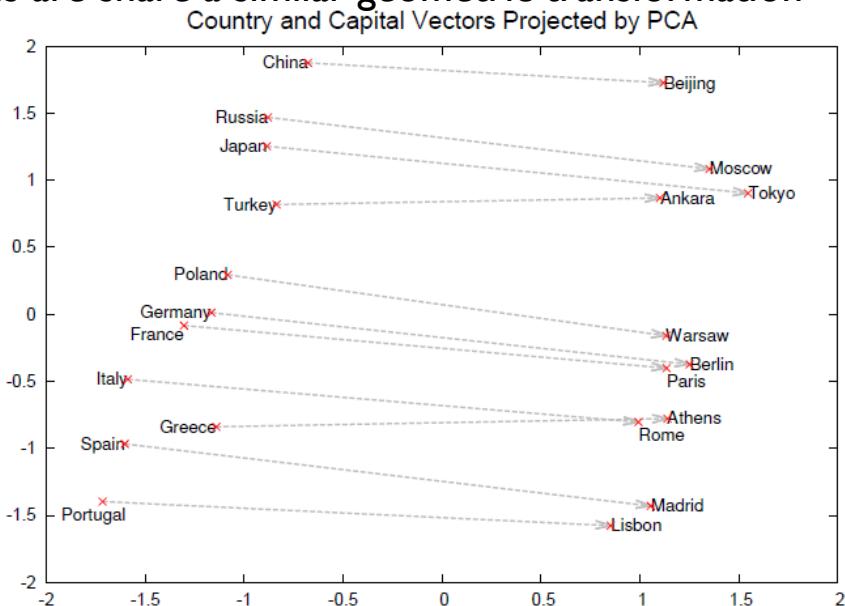


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Learning word vector representations

(Mikolov et al. 2013a, 2013b)

- ▶ Paris – France + Italy = Rome

Table 8: Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Image labeling at Google

DeViSE: A Deep Visual-Semantic Embedding Model (Frome et al,
NIPS 2013)

- ▶ Goal
 - ▶ Learn to label images
- ▶ Approach
 - ▶ Learn a language model from text
 - ▶ With the skip gram model of Mikolov et al.
 - ▶ This models maps each word of a vocabulary onto a fixed vector space representation
 - This is called *word representation*
 - ▶ Vectors are normalized to 1
 - ▶ Learn a visual model for image classification
 - ▶ Similar to the convolutional model of Krizhevsky et al. 2012, trained with 1000 target classes
 - ▶ The last layer of this network computes a representation of each image
 - This is called *image representation*
 - ▶ Visual-semantic embedding
 - ▶ The language model is used to map the label terms onto the fixed word vector space representation
 - ▶ A mapping is learned from the visual representation to the target words, word representation
 - This is a linear mapping with a ranking criterion
 - This is called *transformation mapping*
 - ▶ Test time
 - ▶ An image is
 - mapped onto its image representation
 - This image representation is then mapped via the transformation mapping
 - A nearest neighbor search produces a ranked list of potential labels

Image labeling at Google

DeViSE: A Deep Visual-Semantic Embedding Model (Frome et al, NIPS 2013)

▶ (image from Frome et al, NIPS 2013)

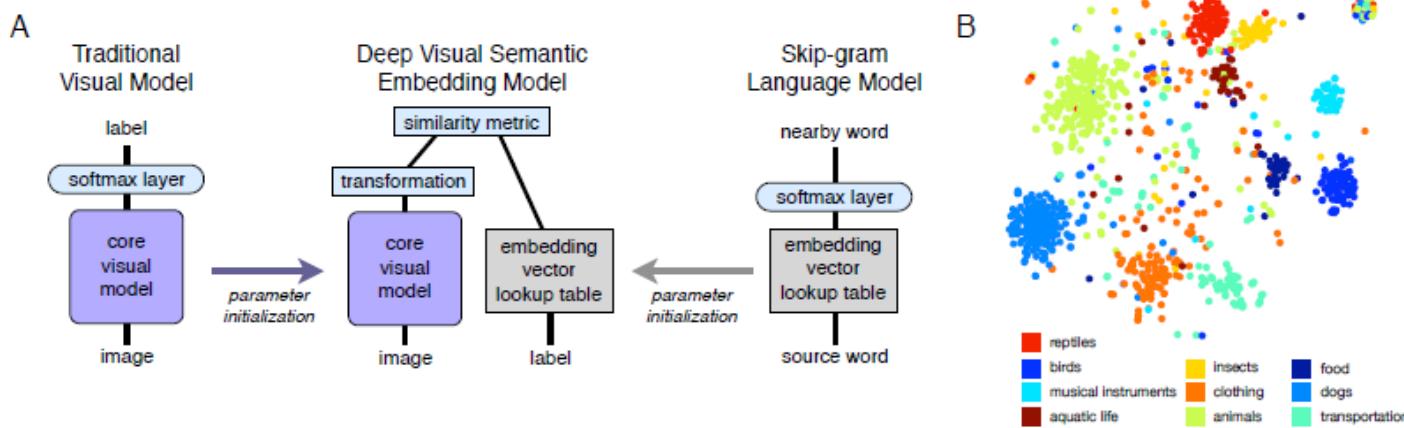


Figure 1: (a) Left: a traditional visual object categorization network with a softmax output layer. Right: a skip-gram language model, which learns word representations that allow the prediction of nearby words in a document. Center: our joint model, which is initialized with parameters pre-trained at the lower layers of the other two models. (b) t-SNE visualization [16] of the skip-gram language model embedding space across a subset of image recognition labels.

$$loss(image, label) = \sum_{j \neq label} \max[0, margin - \vec{t}_{label} M \vec{v}(image) + \vec{t}_j M \vec{v}(image)]$$

Image labeling at Google

DeViSE: A Deep Visual-Semantic Embedding Model (Frome et al, NIPS 2013)

- ▶ Loss function
 - ▶ Ranking criterion
 - ▶ $loss(image, textlabel) = \sum_{j \neq textlabel} \max(0, margin - t^{textlabel} M v^{image} + t^j M v^{image})$
 - ▶ Where
 - ▶ $t^{textlabel}$ is the row vector of the text label representation (normalized to unit norm)
 - ▶ M is the matrix of the linear transformation
 - ▶ v^{image} is the column vector of the image representation
 - ▶ t^j is the row vector of another term representation (another text label also normalized to unit norm)
 - ▶ $t^{textlabel} M v^{image}$ is a dot product (similarity) between $t^{textlabel}$ and $M v^{image}$
 - ▶ This loss function was more efficient than an L_2 loss in the experiments
- ▶ Training algorithm
 - ▶ Stochastic gradient descent
- ▶ Interesting properties
 - ▶ Generalization to images with new labels (0 shot learning)

Image labeling at Google

DeViSE: A Deep Visual-Semantic Embedding Model (Frome et al, NIPS 2013)

	Our model	Softmax over ImageNet 1K
A	eyepiece, ocular Polaroid compound lens telephoto lens, zoom lens rangefinder, range finder	typewriter keyboard tape player reflex camera CD player space bar
B	oboe, hautboy, hautbois bassoon English horn, cor anglais hook and eye hand	reel punching bag, punch bag, ... whistle bassoon letter opener, paper knife, ...
C	barbet patas, hussar monkey, ... babbler, cackler titmouse, tit bowerbird, catbird	patas, hussar monkey, ... proboscis monkey, Nasalis ... macaque titi, titi monkey guenon, guenon monkey
D	fruit pineapple pineapple plant, Ananas ... sweet orange sweet orange tree, ...	pineapple, ananas coral fungus artichoke, globe artichoke sea anemone, anemone cardoon
E	comestible, edible, ... dressing, salad dressing Sicilian pizza vegetable, veggie, veg fruit	pot, flowerpot cauliflower guacamole cucumber, cuke broccoli
F	dune buggy, beach buggy searcher beetle, ... seeker, searcher, quester Tragelaphus eurycerus, ... bongo, bongo drum	warplane, military plane missile projectile, missile sports car, sport car submarine, pigboat, sub, ...

Figure 2: Zero-shot predictions of the joint semantic visual model and a vision model trained on ImageNet 2012 1K. Predictions ordered by decreasing score. Correct predictions labeled in green. Ground truth: (a) telephoto lens, zoom lens; (b) English horn, cor anglais; (c) babbler, cackler; (d) pineapple, pineapple plant, Ananas comosus; (e) salad bar; (f) spacecraft, ballistic capsule, space vehicle.

Modeling relational data (Bordes et al. 2013)

▶ Objective

- ▶ Learn triplets (Subject, Predicate, Object) from multi-relational data
- ▶ Examples
 - ▶ Knowledge bases
 - $(S, P, O) = (\text{Obama}, \text{president of}, \text{USA})$
 - e.g. Freebase 1.2 billion triplets, 80 million entities, thousands of relations
 - ▶ Social networks
 - $(A, \text{friend of}, B)$
- ▶ Infer new relations
 - ▶ $(S, P, ?)$ infer all objects in relation p with s
 - ▶ $(S, ?, O)$ infer all relations between s and o
- ▶ Any other inference task on the triplets

Modeling relational data (Bordes et al. 2013)

▶ Method

- ▶ Learn representations for all occurrences of S, P and O so that (S, P, ?) and (S, ?, O) questions can be answered
- ▶ Use the observations made in Word2Vec
 - ▶ When learning word representations from word sequences, some relations emerge as translations between s and o
 - ▶ Here the model will force the learning of translations in the representation space
 - ▶ Let $s, p, o \in R^k$ the representations to be learned for S, P, O
 - One wants $s + p \approx o$ if (S, P, O) holds
- ▶ Representations are learned by optimizing a loss function representing this goal

Modeling relational data (Bordes et al. 2013)

- ▶ The loss function is a margin based ranking function

- ▶ $L =$

$$\sum_{(S,P,O) \in KB} \sum_{(S',P,O'), \text{noisy version of } (S,P,O)} \max(0, \text{margin} + d(s+p,o) - d(s',p,o'))$$

- ▶ Where

- ▶ $d(s+p,o)$ is a disimilarity measure (e.g. distance)
 - ▶ (S',P,O') is a corrupted version of (S,P,O) , where either S has been replaced by a random subject S' from the knowledge base or O has been replaced by O'
 - ▶ Margin is a positive scalar

- ▶ This loss is a ranking based criterion

- ▶ i.e. it forces $\text{margin} + d(s+p,o) \leq d(s',p,o')$

- ▶ Training

- It amounts at learning the representations s, p, o by optimizing the loss
 - Optimization is performed by stochastic gradient
 - With the constraint $\|s\| = \|o\| = 1$

Modeling relational data (Bordes et al. 2013)

► Examples (from Bordes et al. 2013)

Table 5: Example predictions on the FB15k test set using TransE. **Bold** indicates the test triplet's true tail and *italics* other true tails present in the training set.

INPUT (HEAD AND LABEL)	PREDICTED TAILS
J. K. Rowling influenced by	<i>G. K. Chesterton, J. R. R. Tolkien, C. S. Lewis, Lloyd Alexander,</i> Terry Pratchett, Roald Dahl, Jorge Luis Borges, <i>Stephen King</i> , Ian Fleming
Anthony LaPaglia performed in	<i>Lantana, Summer of Sam, Happy Feet, The House of Mirth,</i> Unfaithful, Legend of the Guardians, Naked Lunch, X-Men, The Namesake
Camden County adjoins	Burlington County, Atlantic County, Gloucester County, Union County, Essex County, New Jersey, Passaic County, Ocean County, Bucks County
The 40-Year-Old Virgin nominated for	<i>MTV Movie Award for Best Comedic Performance,</i> <i>BFCA Critics' Choice Award for Best Comedy,</i> <i>MTV Movie Award for Best On-Screen Duo,</i> MTV Movie Award for Best Breakthrough Performance, MTV Movie Award for Best Movie, MTV Movie Award for Best Kiss, D. F. Zanuck Producer of the Year Award in Theatrical Motion Pictures, Screen Actors Guild Award for Best Actor - Motion Picture
Costa Rica football team has position	<i>Forward, Defender, Midfielder, Goalkeepers,</i> Pitchers, Infielder, Outfielder, Center, Defenseman
Lil Wayne born in	New Orleans, Atlanta, Austin, St. Louis, Toronto, New York City, Wellington, Dallas, Puerto Rico
WALL-E has the genre	Animations, Computer Animation, <i>Comedy film,</i> <i>Adventure film, Science Fiction, Fantasy, Stop motion, Satire, Drama</i>

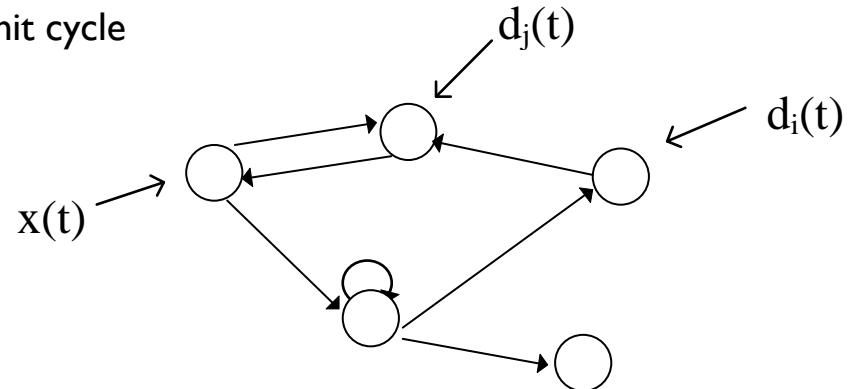
Recursive neural networks (Socher et al. 2013)

- ▶ Exemple bengio slide Canadian AI 2014
- ▶ Layer wise was an idea from 2006. Now possible to learn without pre-training better initialization and alternative non linearities (e.g. rectifier)
- ▶ Compositionality
 - ▶ Recursive NNs à donner en exemple socher
 - ▶ Autres papiers ?
 - ▶ Distributed representations : mutually exclusive features (disentangle)
- ▶ Exemples
 - ▶ Collobert example
 - ▶ Bordes
 - ▶ S. Bengio NIPS 2013, S. Bengio, Weston, Usunier
- ▶ Sparse representations
- ▶ Dropout

Recurrent networks

Recurrent networks - Introduction

- ▶ Recurrent Neural Networks
 - ▶ They are NN with feedback loops
 - ▶ They are dynamical systems
- ▶ Dynamics
 - ▶ Time limited
 - Stop the dynamic after a # time steps
 - ▶ unlimited
 - Wait for convergence: stable state or limit cycle
- ▶ Input x :
 - ▶ Fixed size input
 - $x(t_0), x(t) = 0, \forall t \neq t_0$
 - $x(t) = x(t_0), \forall t$
 - ▶ Sequential input
 - $x(t)$: sequence
- ▶ Supervision
 - ▶ Defined in the time x units space $\{1, \dots, T\} \times \{\text{units}\}$
 - ▶ e.g. at each step, at some prespecified steps x units, at the end of the sequence, ...
- ▶ Recurrent neural networks are dynamic, non linear and continuous state models



Recurrent networks - Introduction

▶ Two types of formulation

- ▶ Input – output
- ▶ State models

$$\begin{aligned}y(t) &= f(x(t)) \\ \begin{cases} c(t) = f(c(t-1), x(t-1)) \\ y(t) = g(c(t)) \end{cases}\end{aligned}$$

□ State models include the input-output formulation as a special case

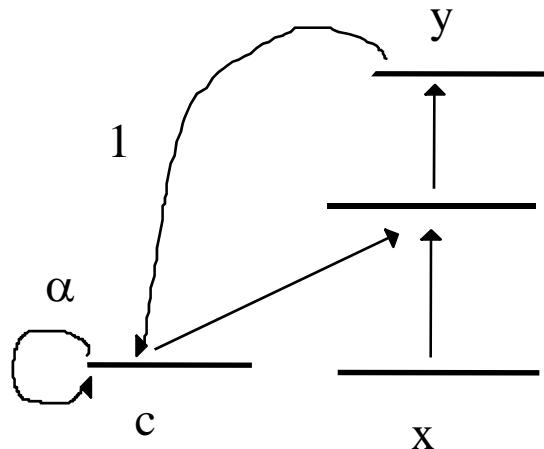
▶ Algorithms

- ▶ Deux main familles:
 - Local recurrences
 - Global recurrences

- ▶ Several formulations of RNN where proposed in the late 80s, early 90s
 - ▶ They faced several limitations and were not successful for applications
 - Recurrent NN are difficult to train
 - They have a limited memory capacity
- ▶ Recently (2012), new models where proposed which alleviate some of these limitations
- ▶ In this course
 - ▶ We briefly introduce key ideas of RNN
 - ▶ We survey some of the developments from the 90s
 - ▶ We introduce some recent developments

Local recurrences (1)

- ▶ Fixed recurrent connexions
 - ▶ Probably the simplest architecture
 - ▶ Recurrent connections are fixed by hand
 - ▶ Only the forward connections are learned like in classical back propagation

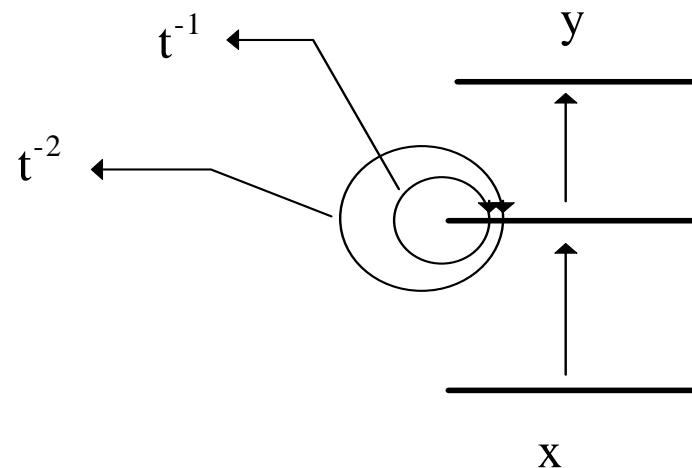


$$\begin{cases} c(t+1) = f(c(t), x(t)) \\ y(t) = g(c(t), x(t)) \end{cases}$$

- ▶ Can be used in two modes
 - ▶ x fixed \rightarrow sequence generation
 - E.g. x is the first term of a sequence to be generated by the network
 - ▶ x sequence $x(1), \dots, x(T) \rightarrow$ classification

Local recurrences (2)

- ▶ Here the local connections may be learned are learned
- ▶ The figure illustrates the simple case of self connections in the hidden layer (i.e. the connection matrix is diagonal)
- ▶ Extensions
 - ▶ Several time delays (1, 2, ...)
 - ▶ Non diagonal recurrent matrix
- ▶ e.g. if delay = 1, the hidden state at time t is:
 - ▶ $h_i(t) = f(w_{ii}h_i(t-1) + \sum_j \text{input cell } w_{ij}x_j(t))$
 - ▶ Where h_i is the state of hidden cell i
- ▶ Training: back-propagation for sequences
 - ▶ Loss:
$$Q = \sum_{t \in T} \sum_{i=1..p} \delta_t (y^i(t) - d^i(t))^2$$
 - ▶ Where t are the supervision steps and $\delta_t = 1$ if t is a supervision step and 0 otherwise



► Properties:

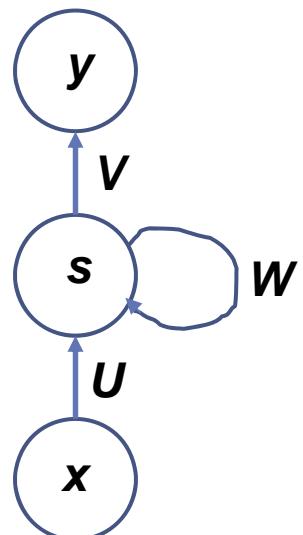
- ▶ Forgetting behavior: gradient vanishing:
 - ▶ Can only memorise short term dependencies

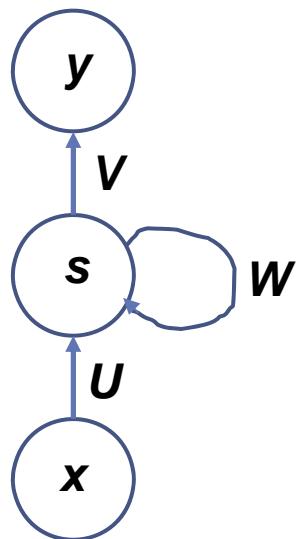
$$\frac{\partial x_i(t)}{\partial x_j(t-q)} \xrightarrow{q \rightarrow \infty} 0$$

- ▶ Can learn fixed size sequences up to the desired precision
- ▶ For unknown size sequences, there exists several limitations, e.g. cannot count the number of occurrences in a sequence of unknown length

Dynamics of RNN

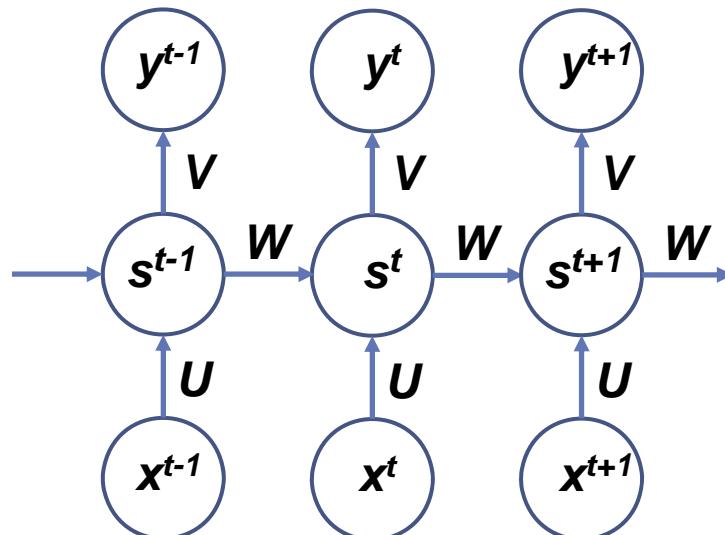
- ▶ We consider different tasks corresponding to different dynamics
 - ▶ They are illustrated for a simple RNN with loops on the hidden units
 - ▶ This can be extended to more complex architectures
- ▶ Basic architecture





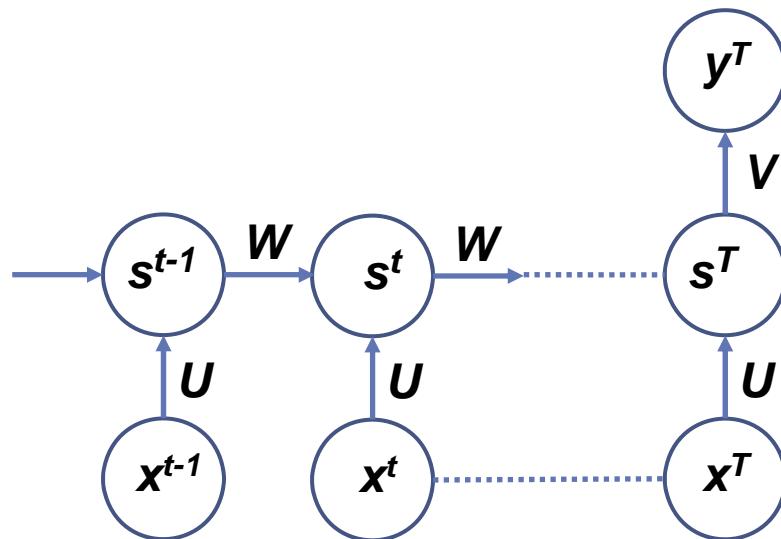
Dynamics of RNN

- ▶ Input and output are sequences of the same length



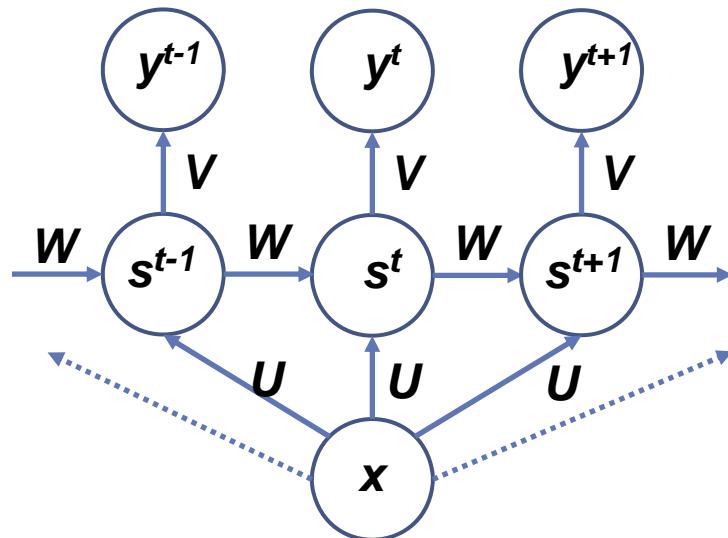
Dynamics of RNN

- ▶ Input is a sequence, output is a real value
 - ▶ Used for classifying single sequences of length T



Dynamics of RNN

- ▶ Input is a fixed vector, output is a sequence



Dynamics of RNN

Fig from Cho et al. EMNLP 2014

- ▶ Input and output are sequences of different length

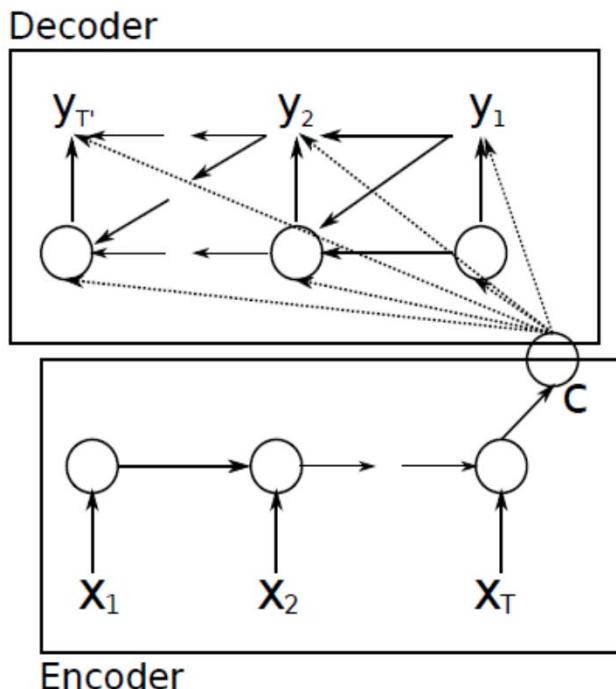
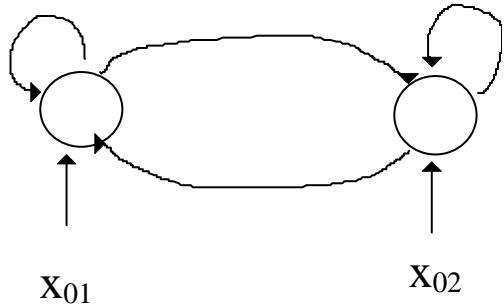


Figure 1: An illustration of the proposed RNN Encoder–Decoder.

Global recurrences

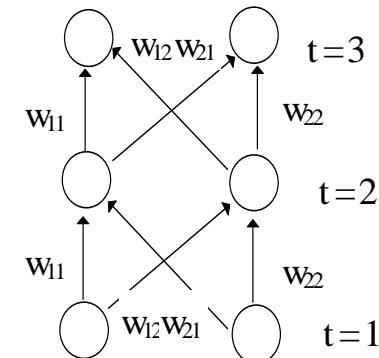


Dynamics

$$x_i(t+1) = f \left(\sum_j w_{ij} x_j(t) + x_{0i}(t) \right)$$

- ▶ Training
 - ▶ Targets are provided at specific steps
- ▶ For sequences of limited size
 - ▶ Network unfolding
 - ▶ Algorithm
 - ▶ Back propagation through time (BPTT)
- ▶ For general sequences
 - ▶ Algorithm is $O(n^4)$ if n units

Unfolded NN:
note that weights are shared



Example : trajectory learning

(Pearlmutter, 1995, IEEE Trans. on Neural Networks – nice review paper on RNN)

- ▶ Globally recurrent net: 2 output units

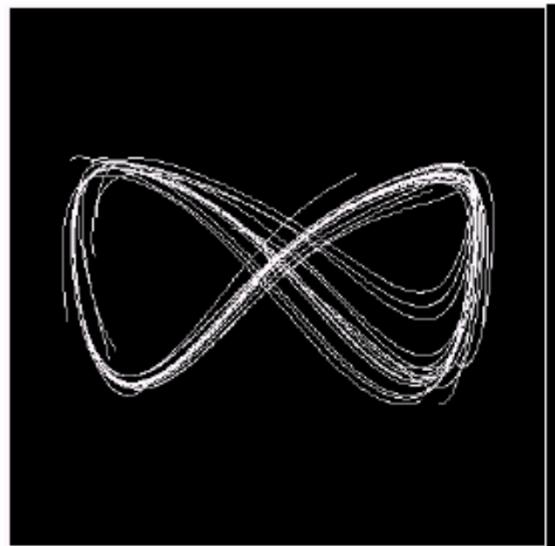


Fig. 9. The output states y_1 and y_2 plotted against each other for a 1000 time unit run, with all the units in the network perturbed by a random amount about every 40 units of time. The perturbations in the circle network (left) were uniform in ± 0.1 , and in the figure eight network (right) in ± 0.05 .

Example – Global RNN: trajectory learning

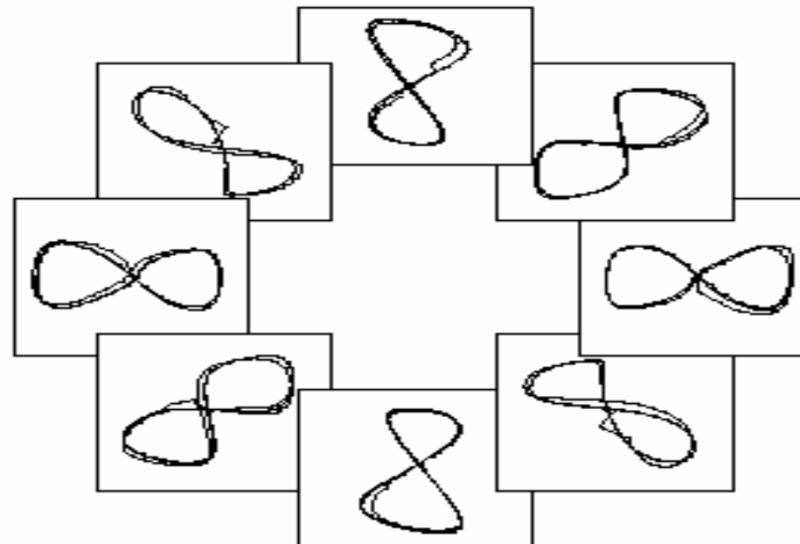
(Pearlmutter, 1995, IEEE Trans. on Neural Networks)

- ▶ Recurrent net with 2 input units ($\sin q, \cos q$), 30 hidden units and 2 output units (x, y) :

$$\begin{pmatrix} x \\ y \end{pmatrix} = 0.4 \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} \sin \pi t/16 \\ \cos \pi t/16 \end{pmatrix} + \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$$

- ▶ Goal: learn a trajectory that depends on the inputs

- ▶



Recurrent neural networks with memory

- ▶ Several attempts in order to limit or bypass the vanishing gradient problem
- ▶ Here we present two families of models
 - ▶ Reservoir computing
 - ▶ Gated recurrent units
 - ▶ Add a memory to the recurrent cell, that enable to remember past states
 - ▶ Add a possibility to forget the past of the sequence during its processing
 - ▶ Allows to model several successive sub-sequences as independent sequences
 - ▶ Other recent developments not covered here
 - ▶ Memory networks

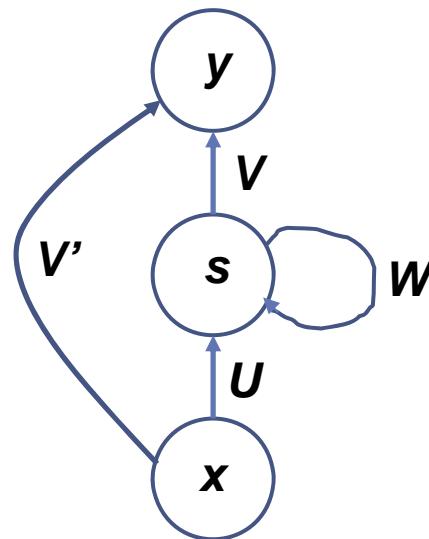
Reservoir computing

- ▶ Reservoir computing corresponds to different models based on a common idea
 - ▶ In a RNN the input and the recurrent weights are fixed and assigned to random values
 - ▶ Their role is double
 - ▶ Perform a non linear projection of the inputs
 - ▶ Encode a dynamic model
 - ▶ Only the output weights will be trained
 - ▶ This can be done using the classical optimization algorithms used for feed forward networks
 - ▶ We present one model family: Echo State Networks - ESN

Reservoir computing: Echo State Networks

Jaeger H. 2001

- ▶ ESN principle
 - ▶ ESN is a RNN
 - ▶ Where input (U) and recurrent (W) weights are generated randomly
 - ▶ Only the weights to the output (V and V') are learned



Reservoir computing: Echo State Networks

▶ ESN equations

- ▶ $\mathbf{s}^t = \tanh(\mathbf{Ux}^t + \mathbf{Ws}^{t-1})$ - tanh is applied elementwise
- ▶ $y^t = \mathbf{V}'\mathbf{x}^t + \mathbf{Vs}^t$

▶ The state s^t is called a reservoir

▶ Training

▶ Let

- ▶ $(\mathbf{x}^1, \dots, \mathbf{x}^T), (\mathbf{d}^1, \dots, \mathbf{d}^T)$ be an associated input – target sequence pair
- ▶ $(\mathbf{y}^1, \dots, \mathbf{y}^T)$ an output sequence computed by the ESN

▶ Loss function

- ▶
$$L(d, y) = \sum_{i=1}^T \|\mathbf{d}^i - \mathbf{y}^i\|^2$$

▶ Variant

▶ Leaky unit – to b defined

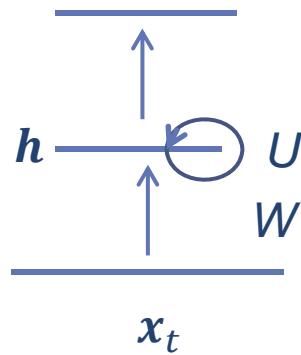
Reservoir computing: Echo State Networks

- ▶ Parameter setting
 - ▶ Size of the reservoir s
 - ▶ U and W are randomly generated
 - ▶ W is generated sparse using typically a uniform distribution of weights on non zero values
 - ▶ U and V , V' are generally dense
 - ▶ the spectral radius of W (largest absolute eigenvalue) governs the behavior of the memory – it is usually fixed around 1
 - ▶ In practice, W is generated, $\text{radius}(W)$ is computed and $\tilde{W} = W \cdot \frac{1}{\text{radius}(W)}$ is used

Gated recurrent unit – GRU (Cho et al. 2014)

- Let us start with a locally connected recurrent net with recurrent units on the hidden layer

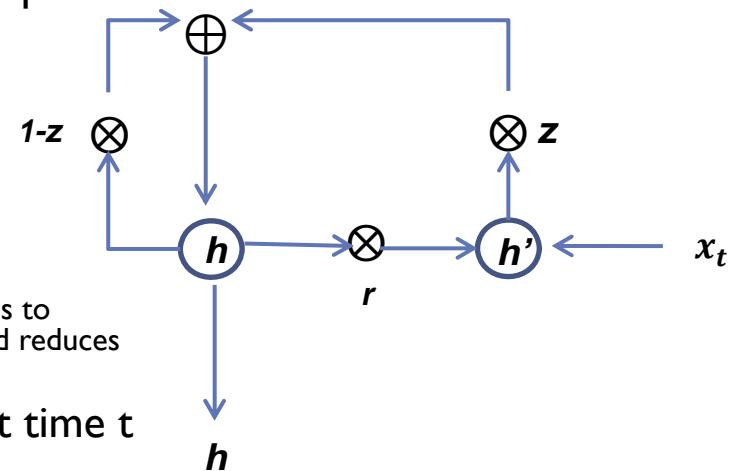
- $\mathbf{h}_t = g(W\mathbf{x}_t + U\mathbf{h}_{t-1})$



- The gated recurrent unit has been proposed to capture dependencies at different time scales
- The forward computation for a gated recurrent unit is governed by the following equations (explained on the next slide):
 - $h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j h_t'^j$
 - $z_t^j = \sigma(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1})^j$
 - $h_t'^j = \tanh(W \mathbf{x}_t + U(\mathbf{r}_t \odot \mathbf{h}_{t-1}))^j$
 - $r_t^j = \sigma(W_r \mathbf{x}_t + U_r \mathbf{h}_{t-1})^j$
 - In these equations
 - Bold characters ($\mathbf{x}_t, \mathbf{h}_{t-1}$) denote vectors
 - h_t^j is the value of hidden cell j at time t
 - \odot is the elementwise multiplication

Gated recurrent unit (Cho et al. 2014)

- ▶ The output h_t^j of cell j is a weighted sum of the cell output at time $t-1$ h_{t-1}^j and a new value of the cell h'_t
 - ▶ $h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j h'_t$
 - ▶ z is a gating function
 - ▶ If $z = 0$ h_t^j is a simple copy of h_{t-1}^j
 - ▶ If $z = 1$ it takes the new value h'_t
 - ▶ w.r.t the classical recurrent unit formulation, this new form allows us to remember the value of the hidden cell at a given time in the past and reduces the vanishing gradient phenomenon
- ▶ The gating function is a function of the current input at time t and the past value of the hidden cell \mathbf{h}_{t-1}
 - ▶ $z_t^j = \sigma(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1})$
- ▶ The new value h'_t is a classical recurrent unit where the values at time $t-1$ are gated by a reset unit r_t
 - ▶ $h'_t = \tanh(W \mathbf{x}_t + U(r_t \odot \mathbf{h}_{t-1}))$
- ▶ The reset unit r_t allows us to forget the previous hidden state and to start again a new modeling of the sequence
 - ▶ This is similar to a new state in a HMM (but it is soft)
 - ▶ $r_t^j = \sigma(W_r \mathbf{x}_t + U_r \mathbf{h}_{t-1})$



Gated recurrent unit (Cho et al. 2014)

- ▶ There are two main novelties in this unit
 - ▶ The z gating function and the $+$ form of the cell value which acts for reducing the vanishing gradient effect
 - ▶ The r gating function which acts for forgetting the previous state and starting again a new subsequence modeling with no memory
- ▶ Each unit adapts its specific parameters, i.e. each may adapt its own time scale and memory size
- ▶ Training
 - ▶ is performed using some adaptation of backpropagation for recurrent nets
 - ▶ All the functions – unit states and gating functions are learned from the data

Long short term memory - LSTM

- ▶ This was initially proposed in 1997 (Hochreiter et al.) and revised latter.
- ▶ State of the art on several sequence prediction problems
 - ▶ Speech, handwriting recognition, translation
 - ▶ Used in conjontions with other models e.g. HMMs or in standalone recurrent neural networks
 - ▶ The presentation here is based on (Graves 2012)

Long short term memory

- ▶ In the LSTM, there are 3 gating functions

- ▶ i: input gating
- ▶ o: output gating
- ▶ f: forget gating

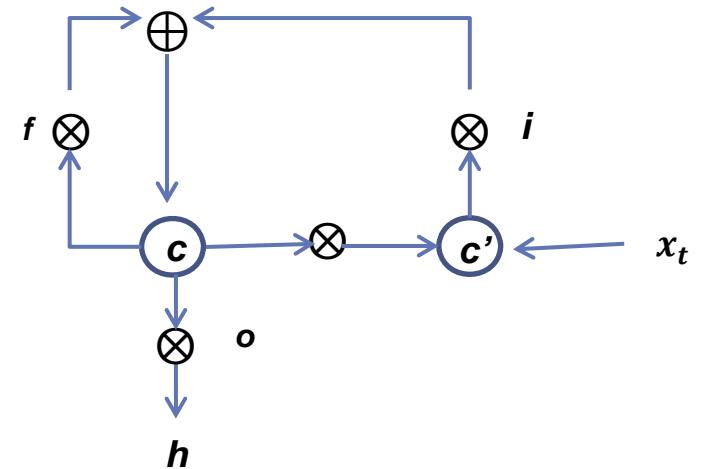
- ▶ Difference with the gated recurrent cell

- ▶ Similarities

- ▶ Both use an additive form for computing the hidden cell state (c) here.
 - This additive component reduces the vanishing gradient effect and allows us to keep in memory past state values.
- ▶ Both use a reset (called here forget (f)) gate
 - The reset permits to start from a new « state » a subsequence prediction

- ▶ Differences

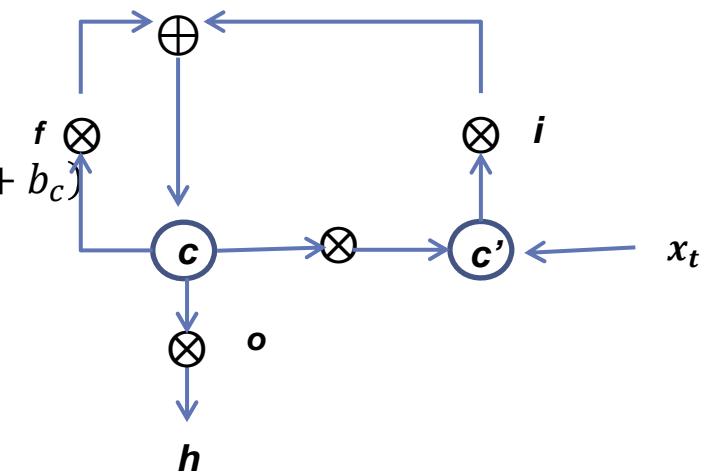
- ▶ No output gating in the GRU
- ▶ Reset does not play exactly the same role



Long short term memory

- ▶ For the forward pass, the different activations are computed as follows and the this order

- ▶ $i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i)$
- ▶ $f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f)$
- ▶ $c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$
- ▶ $o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o)$
- ▶ $h_t = o_t \tanh(c_t)$



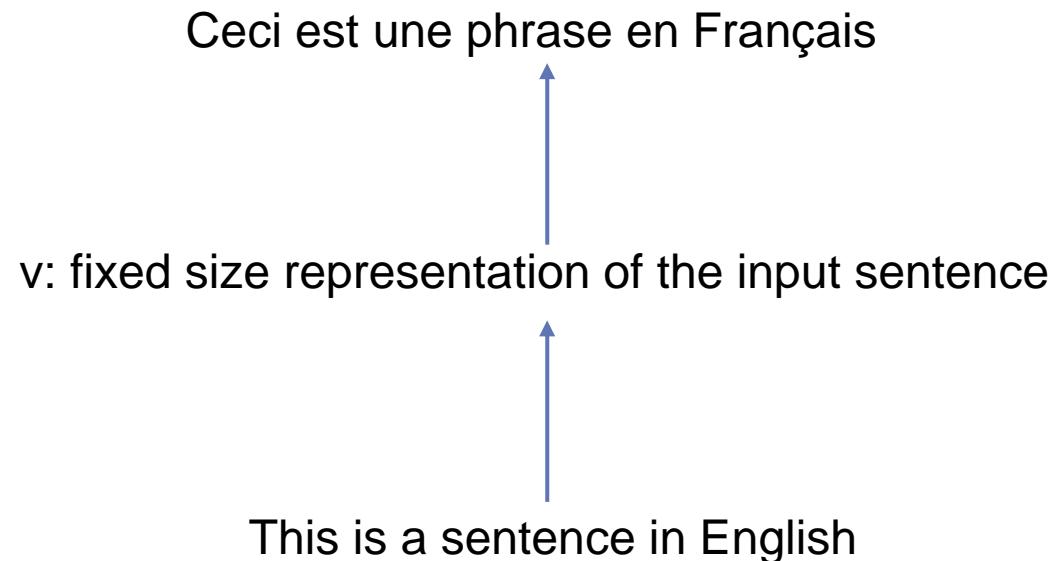
- ▶ c_t^i is a memory of cell i at time t , c_t is computed as for the GRU as a sum of c_{t-1} and of the new memory content $c'_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$
- ▶ o is an output gate
- ▶ σ is a logistic function
- ▶ W_{ci}, W_{cf}, W_{co} are diagonal matrices

Translation

- ▶ NN have been used for a long time in translation systems (as an additional component, e.g. for reranking or as language model)
- ▶ Recently translation systems have been proposed that are based on recurrent neural networks with GRU or LSTM units.
 - ▶ Sutskever et al. 2014, Cho et al. 2014
- ▶ General principle
 - ▶ Sentence to sentence translation
 - ▶ Use an encoder-decoder architecture
 - ▶ Encoding is performed using a RNN on the input sentence (e.g. english)
 - ▶ This transforms a variable length sequence into a fixed size vector which encodes the whole sentence
 - ▶ Starting with this encoding, another RNN generates the translated sentence (e.g. French)

Translation

- ▶ The encoder – decoder architecture



Translation

▶ Let

- ▶ x_1, \dots, x_T be an input sentence
- ▶ $y_1, \dots, y_{T'}$, be an output sentence
- ▶ Note that T and T' might be different and that the word order in the two sentences is also generally different

▶ Objective

- ▶ Learn $p(y_1, \dots, y_{T'} | x_1, \dots, x_T)$
- ▶ Encoder
 - ▶ Reads each symbol of the input sentence sequentially using a RNN
 - ▶ After each symbol the state of the RNN is changed according to $\mathbf{h}_t = f(x_t, \mathbf{h}_{t-1})$
 - ▶ After reading the sentence, the final state is $\mathbf{h}_T = \mathbf{v}$
- ▶ Decoder
 - ▶ Generates the output sequence by predicting the next symbol y_t given the hidden state \mathbf{h}_t , y_{t-1} and the vector \mathbf{v} :
 - $\mathbf{h}_t = f(y_{t-1}, \mathbf{h}_{t-1}, \mathbf{v})$
 - $p(y_t | y_{t-1}, \dots, y_1, \mathbf{v}) = g(y_{t-1}, \mathbf{h}_t, \mathbf{v})$

▶ Training

- ▶ $\max_{\theta} \frac{1}{N} \sum_{n=1}^N \log p_{\theta}(y_n | x_n)$

Translation

► Architecture

- ▶ RNN with 1000 hidden cells
- ▶ Word embeddings of dimension between 100 and 1000
- ▶ Softmax at the output for computing the word probabilities
- ▶ Of the order of 100 M parameters

Neural image caption generator (Vinyals et al. 2015)

▶ Objective

- ▶ Learn a textual description of an image
 - ▶ i.e. using an image as input, generate a sentence that describes the objects and their relation!

▶ Model

- ▶ Inspired by a translation approach but the input is an image
 - ▶ Use a RNN to generate the textual description, word by word, provided a learned description of an image via a deep CNN

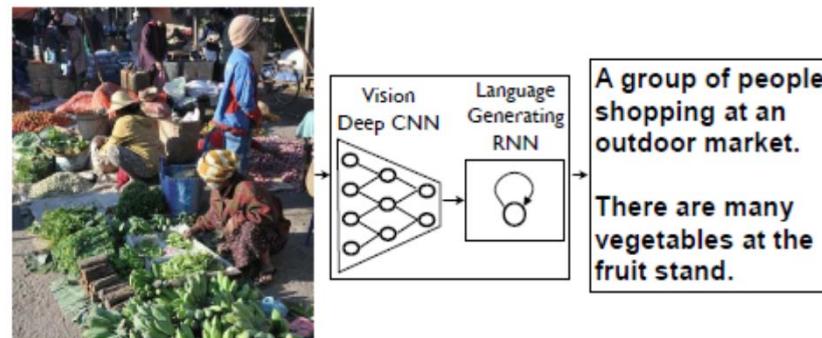


Figure 1. NIC, our model, is based end-to-end on a neural network consisting of a vision CNN followed by a language generating RNN. It generates complete sentences in natural language from an input image, as shown on the example above.

Neural image caption generator (Vinyals et al. 2015)

► Loss criterion

- ▶ $\max_{\theta} \sum_{I,S} \log p(S|I; \theta)$
 - ▶ Where (I, S) is an associated couple (Image, Sentence)
- ▶ $\log p(S|I; \theta) = \sum_{t=1}^N \log p(S_t|I, S_0, \dots, S_{t-1})$
- ▶ $p(S_t|I, S_0, \dots, S_{t-1})$ is modeled with a RNN with S_0, \dots, S_{t-1} encoded into the hidden state h_t of the RNN
- ▶ Here $h_{t+1} = f(h_t, x_t)$ is modelled using a RNN with LSTM
- ▶ For encoding the image, a CNN is used

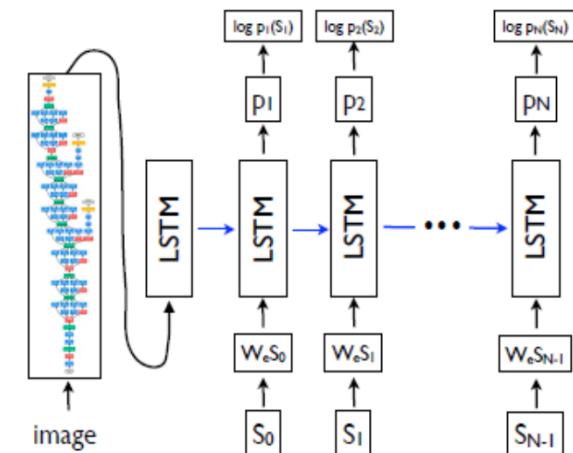
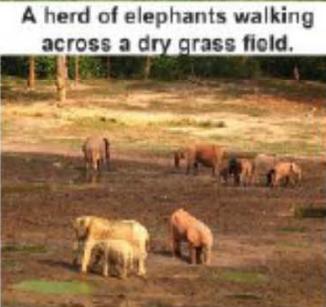


Figure 3. LSTM model combined with a CNN image embedder (as defined in [30]) and word embeddings. The unrolled connections between the LSTM memories are in blue and they correspond to the recurrent connections in Figure 2. All LSTMs share the same parameters.

Neural image caption generator (Vinyals et al. 2015)



Describes without errors

Describes with minor errors

Somewhat related to the image

Unrelated to the image

Figure 5. A selection of evaluation results, grouped by human rating.

► References

- ▶ Barak A. Pearlmutter, Gradient calculations for dynamic recurrent neural networks: a survey. *IEEE Transactions on Neural Networks* 6(5): 1212-1228 (1995)

Apprentissage supervisé

Machines à noyaux

Introduction

- ▶ Familles de machines d'apprentissage générales qui exploitent l'idée suivante :
 - ▶ Projeter les données dans un espace de grande dimension - éventuellement infini - où le problème sera facile à traiter
 - ▶ Utiliser des "projections" non linéaires permettant des calculs "efficaces"
- ▶ Exemples :
 - ▶ Machines à Vecteurs Support (généralisent : hyperplan optimal, cadre Vapnik)
 - ▶ Processus Gaussien (généralisent : régression logistique, cadre Bayesien)
- ▶ Cours
 - ▶ Machines à Vecteurs Support

Perceptron : formulation duale

hyp: 2 classes linéairement séparables, sorties désirées -1, 1

Perceptron primal

Initialiser $w(0) = 0$

Répéter (t)

choisir un exemple, $(x(t), d(t))$

si $d(t)w(t) \cdot x(t) \leq 0$

alors $w(t + 1) = w(t) + d(t)x(t)$

Jusqu'à convergence

Fonction de décision

$$F(x) = \operatorname{sgn}\left(\sum_{j=0}^n w_j x_j\right),$$

$$w = \sum_{i=1}^N \alpha_i d^i x^i$$

α_i : nombre de fois où l'algorithme a rencontré une erreur de classification sur x^i

Perceptron dual

Initialiser $\alpha = 0, \alpha \in R^N$

Répéter (t)

choisir un exemple, $(x(t), d(t))$

soit $k: x(t) = x^k$

si $d(t) \sum_{i=1}^N \alpha_i d^i x^i \cdot x(t) \leq 0$

alors $\alpha_k = \alpha_k + 1$

Jusqu'à convergence

Fonction de décision

$$F(x) = \operatorname{sgn}\left(\sum_{i=1}^N \alpha_i d^i x^i \cdot x\right)$$

Matrice de Gram G :

matrice $N \times N$ de terme i, j : $x^i \cdot x^j$

matrice de similarité entre données

Représentation Duale

- ▶ La plupart des machines à apprentissage linéaires ont une représentation duale
 - ▶ Exemples
 - ▶ Adaline, regression, regression ridge, etc
 - ▶ L'information sur les données est entièrement fournie par la matrice de Gram : $G = (x^i \cdot x^j)_{i,j=1 \dots N}$, qui joue un rôle central
 - ▶ La fonction de décision $F(x)$ s'exprime comme une combinaison linéaire de produits scalaires entre la donnée d'entrée x et les exemples d'apprentissage
 - ▶ Les machines à noyau généralisent ces idées
 - ▶ Une **fonction noyau K** est définie sur $R^n \times R^n$ (on suppose $x \in R^n$) par
$$K(x, z) = \langle \Phi(x), \Phi(z) \rangle$$
où Φ est une fonction de R^n dans un espace muni d'un produit scalaire

Produit Scalaire et Noyaux

- ▶ Projection non linéaire dans un espace de grande dimension H
 - ▶ (en général $\dim H \gg n$, et peut même être infinie)
 - ▶ $\Phi: R^n \rightarrow R^p$ avec $p \gg n$
- ▶ Machine linéaire dans H - Primal :
 - ▶ $F(x) = \sum_{j=1}^p w_j \Phi_j(x) + b$
- ▶ Machine linéaire dans H - Dual :
 - ▶ $w = \sum_{i=1}^N d^i \alpha_i \Phi(x^i), \quad F(x) = \sum_{i=1}^N d^i \alpha_i \Phi(x^i) \cdot \Phi(x) + b,$
- ▶ Calculer les produits scalaires dans l'espace initial : choisir F /
 - ▶ $\Phi(x) \cdot \Phi(x') = K(x, x')$
 - ▶ $F(x) = \sum_{i=1}^N d^i \alpha_i K(x^i, x) + b$
 - ▶ avec K : fonction noyau (i.e. symétrique)

- ▶ Généralise le produit scalaire dans l'espace initial
- ▶ Le calcul de F ne dépend pas directement de la taille de H : les calculs sont faits dans l'espace initial.
- ▶ La machine linéaire dans H peut être construite à partir d'une fonction K sans qu'il soit nécessaire de définir explicitement Φ : en pratique, on spécifiera directement K .
- ▶ Cette idée permet d'étendre de nombreuses techniques linéaires au non linéaire: il suffit de trouver des noyaux appropriés
 - ▶ Exemples
 - ▶ ACP, analyse discriminante, regression, etc

Caractérisation des noyaux

- ▶ Quand peut on utiliser cette idée ?
- ▶ Cas d'un espace fini
 - ▶ Soit $X = \{x^1, \dots, x^N\}$, $K(x, x')$ une fonction symétrique sur X , K est une fonction noyau ssi la matrice $N \times N$ dont l'élément (i, j) est $K(x^i, x^j)$ est positive semi-définie (valeurs propres ≥ 0 ou $x^T K x > 0 \forall x$, avec $K = \text{Matrice} (K(x^i, x^j)) ; i, j = 1..N$)
- ▶ Cas général : Conditions de Mercer (noyaux de Mercer)
 - ▶ Il existe une application Φ_∞ et un développement
$$K(x, x') = \sum_{i=1} \Phi(x)_i \cdot \Phi(x')_i$$
 - ▶ ssi $\forall g / \int g(x)^2 dx$ est fini, $\int K(x, x') g(x) g(x') dx dx' \geq 0$

Caractérisation des noyaux

Espace de Hilbert à noyau autoreproduisant

- ▶ Une fonction $K: X * X \rightarrow \mathbb{R}$ qui est soit continue soit définie sur un domaine fini peut s'écrire sous la forme d'un produit scalaire :

$$K(x, z) = \langle \phi(x), \phi(z) \rangle$$

avec $\phi: x \rightarrow \phi(x) \in F$ espace de Hilbert

ssi c'est une fonction symétrique et toutes les matrices formées par la restriction de K à un échantillon fini sur X sont semi-définies positives).

- ▶ Résultat à la base de la caractérisation effective des fonctions noyaux
- ▶ Il permet de caractériser K comme un noyau sans passer par ϕ
- ▶ C'est une formulation équivalente aux conditions de Mercer

- ▶ L'espace de Hilbert associé au noyau K :

$$F = \left\{ \sum_{i=1}^l \alpha_i K(x_i, \cdot) / l \in N, x_i \in X, \alpha_i \in R, i = 1..l \right\}$$

- ▶ Le produit scalaire défini sur cet espace :

$$\text{Soient } f(\cdot) = \sum_{i=1}^l \alpha_i K(x_i, \cdot), \quad g(\cdot) = \sum_{j=1}^n \beta_j K(x_j, \cdot)$$

$$\langle f, g \rangle = \sum_{i=1}^l \sum_{j=1}^n \alpha_i \beta_j K(x_i, z_j) = \sum_{i=1}^l \alpha_i g(x_i) = \sum_{j=1}^n \beta_j f(z_j)$$

▶ Noyau auto-reproduisant

► Si on prend $g(\cdot) = K(x, \cdot)$ alors $\langle f, K(x, \cdot) \rangle = \sum_{i=1}^l \alpha_i K(x_i, x) = f(x)$

Exemples de noyaux

$$K(x, z) = \langle x \cdot z \rangle^2$$

$$K(x, z) = \left(\sum_{i=1}^n x_i \cdot z_i \right)^2 = \sum_{i,j=1}^n (x_i \cdot x_j)(z_i \cdot z_j)$$

$$K(x, z) = \langle \phi(x) \cdot \phi(z) \rangle \text{ avec } \phi(x) / \phi(x)_{i,j} = (x_i \cdot x_j)_{i,j=1,n}$$

i.e. tous les monomes de $d^o 2 : \binom{n+1}{2}$

$$K(x, z) = (\langle x \cdot z \rangle + c)^2$$

$$K(x, z) = \langle \phi(x) \cdot \phi(z) \rangle \text{ avec } \phi(x) / \phi(x)_{i,j} = ((x_i \cdot x_j)_{i,j=1,n}, (\sqrt{2}cx_i)_{i=1,n}, c)$$

i.e. ss ensemble des polynomes de $d^o 2$

Exemples de noyaux (suite)

$$K(x, x^i) = \begin{cases} (x \cdot x^i + 1)^d & \text{Φ polynome d'ordre } d \\ \exp - \gamma \|x - x^i\|^2 & \text{Φ noyau gaussien} \\ \text{Sigmoïde}(vx \cdot x^i + c) \end{cases}$$

- ▶ En pratique, on utilise souvent des noyaux
 - ▶ Linéaires
 - ▶ Gaussiens
 - ▶ Polynomiaux

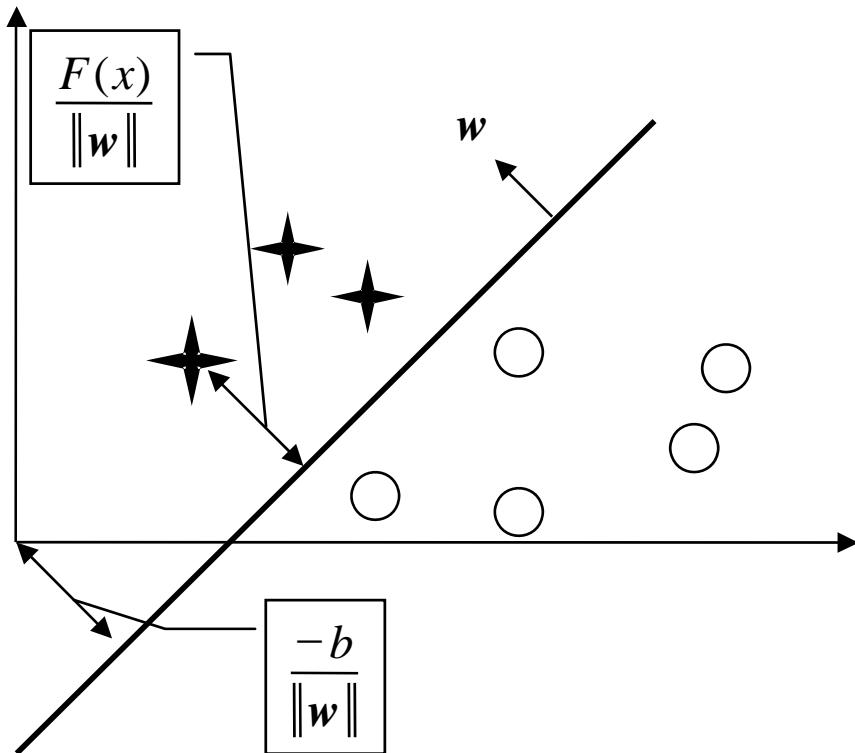
Construction des noyaux en pratique

- ▶ Les résultats de Mercer servent à prouver les propriétés des fonctions noyaux. En pratique, elles sont peu utiles
- ▶ Pour construire des noyaux, on procède par combinaison à partir de noyaux connus
- ▶ Si K_1 et K_2 sont des noyaux sur X^2 , K_3 défini sur F , les fonctions suivantes sont des noyaux :
 - ▶ $K(x, z) = K_1(x, z) + K_2(x, z)$
 - ▶ $K(x, z) = K_1(x, z) \cdot K_2(x, z)^*$
 - ▶ $K(x, z) = aK_1(x, z)$
 - ▶ $K(x, z) = K_3(\Phi(x), \Phi(z))$
 - ▶

Machines à vecteurs support

- ▶ Exposé du cours : discrimination 2 classes
- ▶ Cas général : discrimination multi-classes, régression, densité
- ▶ Idées
 - ▶ Projeter -non linéairement- les données dans un espace de "très" grande taille H
 - ▶ Faire une séparation linéaire de bonne qualité dans cet espace
 - ▶ Raisonner dans H , mais résoudre le problème d'optimisation dans l'espace de départ (noyaux)

▶ Notion de marge



▶ Hyperplan H

- ▶ $F(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b = 0$

▶ Marge géométrique pour x^i

- ▶ $M(\mathbf{x}^i) = d^i \left(\frac{\mathbf{w} \cdot \mathbf{x}^i}{\|\mathbf{w}\|} + \frac{b}{\|\mathbf{w}\|} \right) = d^i \frac{F(\mathbf{x}^i)}{\|\mathbf{w}\|}$

▶ Marge de w par rapport à un ensemble de données D

- ▶ $\text{Min}_i M(\mathbf{x}^i)$

▶ Hyperplan de marge maximale

- ▶ $\text{Max}_{\mathbf{w}} (\text{Min}_i M(\mathbf{x}^i))$

Marge géométrique vs marge fonctionnelle

- ▶ Marge géométrique
 - ▶ $d^i \frac{F(x^i)}{\|w\|}$
- ▶ Marge fonctionnelle
 - ▶ $d^i \cdot F(x^i)$
- ▶ Remplacer w par kw ne change pas la fonction de décision ou la marge géométrique, mais change la marge fonctionnelle.
- ▶ Pour les SVM, on fixera la marge fonctionnelle à 1 et on optimisera la marge géométrique.

Prémisses : Séparation linéaire à hyperplan optimal (1974)

- ▶ Hyp : D linéairement séparable

- ▶ Fonction de décision : $F(x) = w \cdot x + b$ $D = \{x^i, d^i\}_{i=1..N}$ avec $d^i = \pm 1$

- ▶ Pb apprentissage :

- ▶ trouver l'hyperplan optimal H^* qui sépare D i.e.
 - ▶ $d^i \cdot F(x^i) \geq 1, \forall i$

- ▶ avec une marge maximale $M =$

i.e. : Problème Primal :

$$\min_i \frac{d^i \cdot F(x^i)}{\|w\|} = \frac{1}{\|w\|}$$

$$\begin{cases} \text{Minimiser} & \|w\|^2 \\ S.C. & d^i \cdot F(x^i) \geq 1 \end{cases}$$

- ▶ Dans le cas de noyaux linéaires, on résout en général le problème primal
 - ▶ Il existe de nombreux algorithmes rapides
- ▶ Dans le cas de noyaux non linéaires, on va en général résoudre une version duale du problème
 - ▶ On introduit ensuite quelques notions d'optimisation sous contrainte pour décrire la formulation duale du problème

Intermède

Optimisation
Problèmes sous contraintes égalités, inégalités

Optimisation

Problèmes sous contraintes égalités, inégalités

► Soient

► $f, g_{i,i=1,\dots,k}, h_{j,j=1,\dots,m}$ des fonctions définies sur \mathbb{R}^n à valeur dans \mathbb{R}

► On considère le problème primal suivant (pb. 0) :

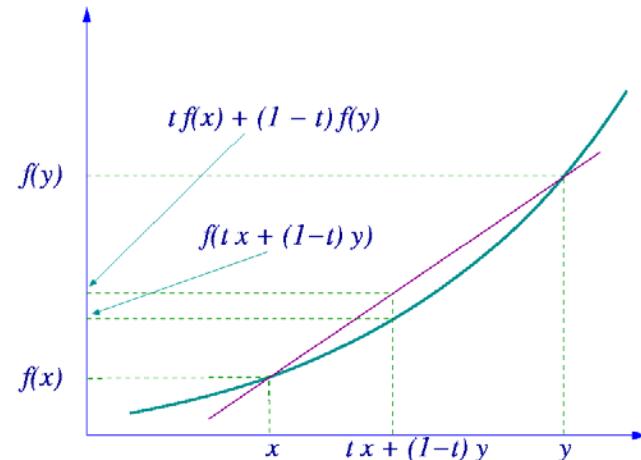
$$\text{Min } f(\mathbf{w}), \mathbf{w} \in \Omega \subset \mathbb{R}^n$$

Sous contraintes

$$\begin{array}{ll} g_i(\mathbf{w}) \leq 0, i = 1, \dots, k & \text{noté } \mathbf{g}(\mathbf{w}) \leq 0 \\ h_j(\mathbf{w}) = 0, j = 1, \dots, m & \text{noté } \mathbf{h}(\mathbf{w}) = 0 \end{array}$$

- Fonction objectif $f(w)$
- Région admissible $R = \{\mathbf{w} \in \Omega : \mathbf{g}(\mathbf{w}) \leq 0, \mathbf{h}(\mathbf{w}) = 0\}$, région de Ω où f est définie et les contraintes vérifiées
- \mathbf{w}^* est un **minimum global** si il n'existe pas d'autre point tel que $f(\mathbf{w}) < f(\mathbf{w}^*)$, c'est un optimum local si $\exists \epsilon > 0 : f(\mathbf{w}) \geq f(\mathbf{w}^*)$, sur la boule $\|\mathbf{w} - \mathbf{w}^*\| < \epsilon$
- Une contrainte $g_i(\mathbf{w}) \leq 0$ est dite **active** si la solution \mathbf{w}^* vérifie $\mathbf{g}(\mathbf{w}^*) = 0$ et inactive sinon
- La valeur optimale de la fonction objectif (solution du pb. 0 est appelée la **valeur du problème d'optimisation primal**)

- ▶ $f(w)$ est **convexe** pour $w \in \mathbb{R}^n$ si
 $\forall t \in [0,1], \forall w, v \in \mathbb{R}^n, \forall t \in [0,1], f(tw + (1-t)v) \leq tf(w) + (1-t)f(v)$



- ▶ Un **ensemble** $\Omega \subset \mathbb{R}^n$ est **convexe** si $\forall w, v \in \mathbb{R}^n, \forall t \in [0,1], tw + (1-t)v \in \Omega$
- ▶ Si une fonction est convexe, tout minimum local est un minimum global
- ▶ Un **problème d'optimisation** pour lequel Ω est convexe, la fonction objectif et les contraintes sont convexes est dit **convexe**

Optimisation non contrainte

► Th. Fermat

- Une C.N. pour que w^* soit un min. de $f(w)$, $f \in C^1$ est $\frac{\partial f(w^*)}{\partial w} = 0$
- Si f est convexe c'est une Condition Suffisante

Optimisation avec contraintes égalités Lagrangien

- ▶ Optimisation avec contraintes égalité (pb I):

$$\text{Min } f(\mathbf{w}), \mathbf{w} \in \Omega \subset \mathbb{R}^n$$

Sous les contraintes

$$h_j(\mathbf{w}) = 0, \quad j = 1, \dots, m \quad \text{noté } \mathbf{h}(\mathbf{w}) = 0$$

- ▶ On définit le Lagrangien $L(\mathbf{w}, \boldsymbol{\beta})$ associé à ce problème par

$$L(\mathbf{w}, \boldsymbol{\beta}) = f(\mathbf{w}) + \sum_{j=1}^m \beta_j h_j(\mathbf{w})$$

- ▶ les β_j sont les coefficients de Lagrange
- ▶ Rq
 - ▶ Si \mathbf{w}^* est une solution du problème d'optimisation sous contrainte, il est possible que $\frac{\partial f(\mathbf{w}^*)}{\partial \mathbf{w}} \neq 0$

Optimisation avec contraintes égalités

Th. Lagrange

► Th. Lagrange

- ▶ Une CN pour que w^* , soit solution de (pb. I), avec $f, h_i \in C^1$ est
 - $\frac{\partial L(w^*, \beta^*)}{\partial w} = 0$
 - $\frac{\partial L(w^*, \beta^*)}{\partial \beta} = 0$
- ▶ Si $L(w, \beta^*)$ est une fonction convexe de w , c'est une condition suffisante
- ▶ Rq
 - ▶ La première condition donne un nouveau système d'équations
 - ▶ La seconde donne les contraintes

Optimisation sous contraintes égalité + inégalités - Lagrangien augmenté

- ▶ De même, on définit le **Lagrangien augmenté** pour le pb. 0 :

$$L(\mathbf{w}, \boldsymbol{\beta}) = f(\mathbf{w}) + \sum_{i=1}^k \alpha_i g_i(\mathbf{w}) + \sum_{j=1}^m \beta_j h_j(\mathbf{w})$$

▶ Formulation duale du problème d'optimisation

- ▶ Le problème d'optimisation dual correspondant au problème primal pb 0 est :

$$\text{Maximiser}_{\alpha, \beta} \quad \theta(\alpha, \beta) = \min_{w \in \Omega} L(w, \alpha, \beta)$$

sous contrainte $\alpha \geq 0$

- ▶ Max $\theta(\alpha, \beta)$ est appelé la **valeur du dual**
- ▶ Rq : $\inf_{w \in \Omega} L(w, \alpha, \beta)$ est une fonction de α, β uniquement
- ▶ Propriété : la valeur du dual est bornée supérieurement par la valeur du primal

$$\max_{\alpha, \beta, \alpha \geq 0} \min_{w \in \Omega} L(w, \alpha, \beta) \leq \min_{w \in \Omega} \max_{\alpha, \beta, \alpha \geq 0} L(w, \alpha, \beta)$$

Dans certains cas, on a égalité, cf. dualité forte

▶ Théorème de dualité forte

▶ Etant donné un problème d'optimisation

$\text{Min } f(\mathbf{w}), \mathbf{w} \in \Omega \subset \mathbb{R}^n$ convexe et $f \in C^1$ convexe

Sous contraintes

$$g_i(\mathbf{w}) \leq 0, i = 1, \dots, k \quad \text{noté } \mathbf{g}(\mathbf{w}) \leq 0$$

$$h_j(\mathbf{w}) = 0, j = 1, \dots, m \quad \text{noté } \mathbf{h}(\mathbf{w}) = 0$$

où les g_i et les h_j sont affines ($h_j(\mathbf{w}) = A_j \mathbf{w} + b_j$)

- ▶ alors les valeurs du primal et du dual sont égales
- ▶ Les conditions d'existence d'un optimum sont données par le théorème de Kuhn et Tucker

Optimisation

Th. Kuhn et Tucker

- ▶ On considère (pb. 0) avec Ω convexe et $f \in C^1$ convexe, g_i, h_j affines ($h = A.w + b$)
- ▶ Une CNS pour que w^* soit un optimum est qu'il existe α^* et β^* :

$$\left\{ \begin{array}{l} \frac{\partial L(w^*, \alpha^*, \beta^*)}{\partial w} = 0 \\ \frac{\partial L(w^*, \alpha^*, \beta^*)}{\partial \beta} = 0 \\ \alpha_i^* g_i(w^*) = 0, i = 1..k \\ g_i(w^*) \leq 0, i = 1..k \\ \alpha_i^* \geq 0, i = 1..k \end{array} \right.$$

- ▶ La formulation duale est une alternative à la formulation primaire qui peut se révéler plus simple à traiter

- ▶ Rq
 - ▶ La 3^e condition dite condition complémentaire de Karush-Kuhn-Tucker implique que pour une contrainte active $\alpha_i^* \geq 0$ alors que pour une contrainte inactive $\alpha_i^* = 0$
 - ▶ Soit une contrainte est **active** ($\alpha_i^* \geq 0$ et $g_i(w^*) = 0$), w^* est un point frontière de la région admissible
 - ▶ Soit elle est **inactive** ($\alpha_i^* = 0$) et w^* est dans la région admissible
 - ▶ Si le point solution w^* est dans la région admissible (contrainte inactive) alors les conditions d'optimalité sont données par le th. de Fermat et $\alpha_i^* = 0$. Si il est sur la frontière (contrainte active), les conditions d'optimalité sont données par le th. de Lagrange avec $\alpha_i^* > 0$.
- ▶ Fin de l'intermède

SVM – formulations primale et duale

Cas d'un noyau linéaire

- ▶ SVM

- ▶ Ω, f , contraintes sont convexes, L est quadratique
- ▶ On étudie le cas, $D = \{(x^i, d^i)\}_{i=1..N}$ linéairement séparables,
 $d^i \in \{-1, 1\}$

- ▶ Pb. Primal

$$\text{Min}(w.w) \quad (\text{i.e. max la marge})$$

sous les contraintes

$$d^i(w.x^i + b) \geq 1, i = 1..N$$

- ▶ Lagrangien primal

$$L(w, b, \alpha) = \frac{1}{2} w.w - \sum_{i=1}^N \alpha_i (d^i (w.x^i + b) - 1)$$

- ▶ Lagrangien dual

$$L(w, b, \alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N d^i d^j \alpha_i \alpha_j (x^i . x^j)$$

- ▶ Avec $\alpha_i \geq 0$ dans les 2 cas

SVM – formulations primale et duale

▶ Pb. Dual

$$\text{Max } L(\alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N d^i d^j \alpha_i \alpha_j (x^i \cdot x^j)$$

sous les contraintes

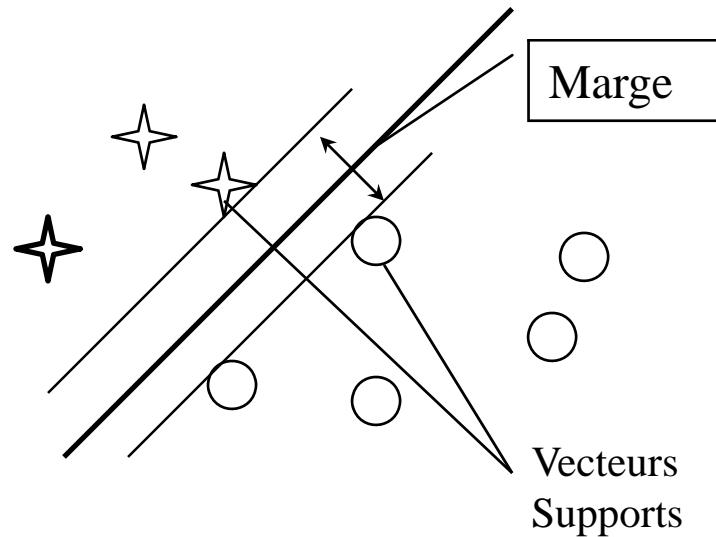
$$\begin{cases} \sum_{i=1}^N d^i \alpha_i = 0 \\ \alpha_i \geq 0, i = 1..N \end{cases}$$

▶ C'est un problème d'optimisation quadratique sous contrainte

- ▶ Solution : w^* dépend uniquement des points supports i.e. points sur la marge qui vérifient : $d^i F^*(x^i) = 1$
- ▶ la fonction de décision prend la forme

$$F(x, \alpha^*, \beta^*) = \sum_{i \text{ vecteur support}} d^i \alpha^*_i (x^i \cdot x) + b^*$$

- ▶ Rq: Quelque soit la dimension de l'espace, le nombre de degrés de liberté est "égal" au nombre de points de support
- ▶ F^* dépend uniquement du produit scalaire $x^i \cdot x$



Machines à vecteurs supports cas de noyaux non linéaires

- ▶ Faire une séparation à marge max. dans un espace défini par une fonction noyau.
- ▶ Tous les résultats sur le classifieur linéaire à marge max. se transposent en remplaçant $x^i.x$ par $K(x^i, x)$

$$\Phi : R^n \rightarrow R^p$$

$$W = \sum_{x^i \in V.S.} d^i \alpha_i \Phi(x^i) \quad F(x) = \sum_{x^i \in V.S.} d^i \alpha_i \Phi(x^i) \Phi(x) + b$$

$$\Phi(x).\Phi(x') = K(x, x')$$

$$F(x) = \sum_{x^i \in V.S.} d^i \alpha_i K(x, x^i) + b$$

▶ Apprentissage :

- ▶ On résout le problème d'optimisation dual :

$$\text{Maximiser} \quad L(\alpha) = \sum_i \alpha_i - \sum_{i,j} \alpha_i \alpha_j d^i d^j K(x^i, x^j)$$

$$S.C \quad \alpha_i \geq 0 \quad \text{et} \quad \sum_i \alpha_i d^i = 0$$

- ▶ Problème minimisation quadratique sous contraintes dans l'espace de départ
- ▶ Difficile en pratique : différents algorithmes.
- ▶ Dans la solution optimale $\alpha_i > 0$ uniquement pour les points support.
 - ▶ Seuls les produits scalaires K apparaissent, et pas les Φ .

Propriétés de généralisation - exemples

► Th 1

$$E [P(\text{erreur}(x))] \leq \frac{E [\# \text{vecteurs supports}]}{\# \text{exemples apprentissage} - 1}$$

- ▶ peu de points support → meilleure généralisation
- ▶ indépendant de la taille de l'espace de départ

► Th 2

- ▶ Si

$$\exists q / \forall i = 1..N, \|x^i\| \leq q$$

- ▶ l'hyperplan optimal passe par l'origine et a pour marge ρ

$$E \left[\frac{q}{2} \right]$$

- ▶ Alors

$$E [P(\text{erreur}(x))] \leq \frac{\rho}{N}$$

- Dans les 2 cas, $E[P()]$ est l'espérance sur tous les ensembles de taille $I-1$, et $E[\text{membre droit}]$ est l'espérance sur tous les ensembles d'apprentissage de taille I (leave one out).

Cas non linéairement séparable

- ▶ Marges molles

- ▶ L'algorithme est instable

- ▶ Dans les cas non linéairement séparables
 - ▶ Dans le cas de données réelles même linéairement séparables

- ▶ **Solution adoptée en pratique**

- autoriser des erreurs, i.e. prendre pour contraintes :

$$d^i (W \cdot \Phi(x^i) + b) \geq 1 - \eta^i$$
$$\eta^i \geq 0$$

- ▶ $\eta^i = 0$, x^i est correctement classifié et est du bon côté de la marge
- ▶ $0 < \eta^i \leq 1$, x^i est correctement classifié, est à l'intérieur de la marge
- ▶ $\eta^i > 1$, x^i est mal classé
- ▶ η^i : *slack variable*

- ▶ **But**
 - ▶ Maximiser la marge tout en pénalisant les points qui sont mal classés
- ▶ **Formalisation**
 - ▶ Plusieurs expressions possibles du problème
 - ▶ L'une des plus courantes :

$$Min(w.w) + C \sum_{i=1}^N \eta^i \quad (\text{i.e. max la marge})$$

S.C.

$$d^i(w.x^i + b) \geq 1 - \eta^i, i = 1..N$$

$$\eta^i \geq 0, i = 1..N$$

- ▶ **C fixé par validation croisée joue le rôle de paramètre de régularisation**

▶ Marges molles – formulation duale

Maximiser $L(\alpha) = \sum_i \alpha_i - \sum_{i,j} \alpha_i \alpha_j d^i d^j K(x^i, x^j)$

S.C $0 \leq \alpha_i \leq C$ et $\sum_i \alpha_i d^i = 0$

Algorithmes d'optimisation

- ▶ **Algorithmes d'optimisation standard pour la programmation quadratique sous contrainte**
 - ▶ e.g. **Sequential Minimal Optimization (SMO)**
- ▶ **Algorithmes stochastiques - SVM Results –(Bottou 2007)**
 - ▶ Task : Document classification - RCV1 documents belonging to the class CCAT (2 classes classification task)
 - ▶ Programs [SVMLight](#) and [SVMPerf](#) are well known SVM solvers written by [Thorsten Joachims](#). SVMLight is suitable for SVMs with arbitrary kernels. Similar results could be achieved using [Chih-Jen Lin's LibSVM](#) software. SVMPerf is a specialized solver for linear SVMs. It is considered to be one of the most efficient optimizer for this particular problem.

Algorithm (hinge loss)	Training Time	Primal cost	Test Error
SVMLight	23642 secs	0.2275	6.02%
SVMPerf	66 secs	0.2278	6.03%
Stochastic Gradient (svmsgd)	1.4 secs	0.2275	6.02%
Stochastic Gradient (svmsgd2)	1.4 secs	0.2275	6.01%

Apprentissage non supervisé

Algorithme EM et mélange de densités
Spectral clustering
Non Negative Matrix Factorization

Applications

- ▶ analyse des données quand il n'y a pas de connaissance sur la classe.
 - ▶ e.g. pas d'étiquetage des données (problème nouveau)
- ▶ trop de données ou étiquetage trop compliqué
 - ▶ e.g. traces utilisateur (web), documents web, parole, etc
- ▶ réduction de la quantité d'information
 - ▶ e.g. quantification
- ▶ découverte de régularités sur les données ou de similarités.

Apprentissage non supervisé

Algorithme Espérance Maximisation (EM)
Application aux mélanges de densités

Algorithme E. M. (Espérance Maximisation) - Introduction

- ▶ On dispose
 - ▶ de données $D = \{x^i; i = 1 \dots N\}$
 - ▶ On n'a pas d'étiquette d^i associée à x^i
 - ▶ d'un modèle génératif, de paramètres θ
- ▶ On veut trouver les paramètres du modèle qui expliquent au mieux la génération des données
- ▶ On se donne un critère
 - ▶ Ici on considère la vraisemblance des données qui est le critère le plus fréquent
 - ▶ $P(D; \theta) = P(x^1, \dots, x^N; \theta)$
 - ▶ D'autres critères sont également utilisés
- ▶ On va essayer de déterminer les paramètres θ de façon à maximiser la vraisemblance

Exemple : mélange de deux populations

- ▶ On recueille des données sur deux populations
 - ▶ e.g. taille d'individus $D = \{x^i; i = 1 \dots N\}$
 - ▶ Hypothèse : les données de chaque population sont gaussiennes 1 dimension
 - ▶ $N(\mu_1, \sigma_1), N(\mu_2, \sigma_2)$
- ▶ Problème
 - ▶ estimer les μ_i et les σ_i à partir des données
 - ▶ Si les d^i sont connus, i.e. $D = \{(x^i, d^i); i = 1 \dots N\}$ la solution est simple
 - ▶ On a deux population séparées (2 classes) C_1, C_2
 - ▶ On utilise les estimateurs classiques de la moyenne et de la variance
 - Pour la moyenne $\mu_j = \frac{1}{|C_j|} \sum_{x^i \in C_j} x^i$, idem pour la variance
 - ▶ Ces estimateurs usuels sont les estimateurs du maximum de vraisemblance
 - Cf. slide suivant

Mélange de deux populations

Cas où l'appartenance est connue

- ▶ Vraisemblance
 - ▶ $P(D|\theta) = \prod_{x^i \in C_1} p(x^i | \theta_1) \prod_{x^j \in C_2} p(x^j | \theta_2)$
- ▶ En pratique on maximise la log-vraisemblance
 - ▶ $L(\theta) = \log(P(D|\theta)) = \sum_{x^i \in C_1} \log p(x^i | \theta_1) + \sum_{x^j \in C_2} \log p(x^j | \theta_2)$
 - ▶ Cas des gaussiennes
 - ▶ $p(x|C_k) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu_k)^2}{2\sigma_k^2}\right)$
 - ▶ $\frac{\partial L}{\partial \mu_k} = 0 \Leftrightarrow \mu_k = \frac{1}{|C_k|} \sum_{x^i \in C_k} x^i$, idem pour la variance

Mélange de deux populations

Cas où la probabilité d'appartenance est connue

- ▶ On connaît les $p(C_k|x)$, $k = 1, 2$ pour tous les x de l'ensemble d'apprentissage
- ▶ $p(x^i|\theta) = p(x^i|C_1)p(C_1) + p(x^i|C_2)p(C_2)$
- ▶ C'est un modèle de mélange de deux densités
- ▶ Log-vraisemblance
 - ▶ $L(\theta) = \log(P(D|\theta)) = \sum_{x^i} \log(p(x^i|C_1)p(C_1) + p(x^i|C_2)p(C_2))$
 - ▶ Cas des gaussiennes
 - ▶ $\frac{\partial L}{\partial \mu_k} = 0 \Leftrightarrow \mu_k = \frac{\sum_{x^i} p(C_1|x^i).x^i}{\sum_{x^i} p(C_1|x^i)}$, idem pour la variance

Mélange de densités gaussiennes

La probabilité d'appartenance est inconnue

- ▶ On suppose que les données sont générées par le modèle suivant
 - ▶ Tirer $z^i \sim \text{Multinomiale}(\phi)$ $\phi_i \geq 0, \sum_{i=1}^k \phi_i = 1$
 - ▶ Tirer $x^i / x^i | z^i = j \sim N(\mu_j, \sigma_j)$
 - ▶ i.e. x^i est généré en choisissant d'abord $z^i \in \{1, \dots, k\}$ et ensuite en effectuant un tirage suivant une gaussienne $N(\mu_j, \sigma_j)$ si $z^i = j$
- ▶ C'est ce qu'on appelle un **mélange de gaussienne**
- ▶ la vraisemblance des données s'écrit
 - ▶ $l(\theta) = l(\mu, \sigma, \phi) = \prod_{i=1}^N p(x^i; \theta) = \prod_{i=1}^N \sum_{j=1}^k p(x^i, z^i = j; \theta) = \prod_{i=1}^N \sum_{j=1}^k p(x^i | z^i = j; \mu, \sigma) p(z^i | \phi)$
- ▶ La log vraisemblance
 - ▶ $l(\mu, \sigma, \phi) = \sum_{i=1}^N \log(\sum_{j=1}^k p(x^i | z^i = j; \mu, \sigma) p(z^i | \phi))$
- ▶ Les deux exemples précédents (appartenance connue et probabilité d'appartenance connue sont des cas particuliers de ce modèle de mélange)

Mélange de densités gaussiennes (suite)

- ▶ Quand la probabilité d'appartenance est inconnue, on ne peut pas trouver une forme analytique pour les estimateurs du maximum de vraisemblance
- ▶ L'algorithme E.M. apporte une solution à ce problème
- ▶ Il itère 2 étapes principales
 - ▶ Etape E (Espérance)
 - ▶ Calculer $p(z^i = j|x^i) \forall i, j$
 - ▶ Etape M (Maximisation) – estimation des paramètres des gaussiennes par MV
 - ▶ $\phi_j = \frac{1}{N} \sum_{i=1}^N p(z^i = j|x^i)$
 - ▶ $\mu_j = \frac{\sum_{i=1}^N p(z^i=j|x^i)x^i}{\sum_{i=1}^N p(z^i=j|x^i)}$
 - ▶ $\sigma_j = \frac{\sum_{i=1}^N p(z^i=j|x^i)(x^i - \mu_j)^2}{\sum_{i=1}^N p(z^i=j|x^i)}$

Modèle de mélange (suite)

- ▶ **Etape E**
 - ▶ Si on connaît les lois de mélange (les (μ, σ, ϕ)), on peut facilement calculer les $p(z^i = j|x^i)$ par $p(z^i = j|x^i) = \frac{p(x^i|z^i=j)p(z^i=j)}{p(x^i)}$
- ▶ **Etape M**
 - ▶ Si on connaît les probabilités d'appartenance, on peut facilement estimer les paramètres des lois de mélange comme vu dans l'exemple
- ▶ Il suffit de partir d'une valeur initiale – pour les $p(z^i = j|x^i)$ ou pour les paramètres des lois de mélange pour exécuter l'algorithme
 - ▶ Celui ci converge vers un maximum local de la vraisemblance
- ▶ Cet exemple est un instance d'un algorithme plus général appelé algorithme E.M.

Algorithme E.M.

- ▶ **Problème**
 - ▶ On a un problème d'estimation pour lequel on connaît un ensemble d'apprentissage $D = \{x^1, \dots, x^N\}$
 - ▶ On veut trouver les paramètres qui maximisent la (log) vraisemblance des données $L(\theta) = \log p(D; \theta)$
 - ▶ On ne connaît pas de formule analytique permettant d'estimer les paramètres θ
- ▶ **On postule**
 - ▶ l'existence de variables cachées z responsables de la génération des données
 - ▶ À chaque x^i , on associe sa classe cachée z^i
 $Z = \{z^i; i = 1..N\}$
 - ▶ l'existence d'une fonction densité jointe sur les données observées et cachées $p(x, z)$
 - ▶ $p(D, Z|\theta)$ sera appelé vraisemblance complète des données pour le modèle θ .
- ▶ **Remarque**
 - ▶ Les variables z sont inconnues et sont considérées comme des variables aléatoires
 - ▶ $P(D, Z|\theta)$ sera donc elle-même une variable aléatoire

Algorithme EM

- ▶ L'algorithme E.M. fournit une solution à notre problème
- ▶ C'est un algorithme itératif en 2 étapes
 - ▶ Etape E.
 - ▶ Construire une borne inférieure de $L(\theta)$
 - ▶ Etape M.
 - ▶ Déterminer les paramètres qui optimisent cette borne inférieure
- ▶ La borne inférieure est construite de façon à garantir une optimisation de $L(\theta)$

Algorithme EM – Etape E

- ▶ La borne inférieure est définie par une fonction auxiliaire Q
 - ▶ Q est l'espérance de la log-vraisemblance des données complètes $\log(p(D, Z; \theta))$, connaissant le modèle courant à l'étape t , $\theta(t)$
 - ▶ L'espérance est calculée par rapport à la distribution des variables cachées z connaissant le modèle courant $\theta(t)$: $p(z|x; \theta(t))$
 - ▶ $Q(\theta, \theta(t)) = E_Z[\log p(D, Z; \theta) | \theta(t)] = \sum_Z \log p(D, Z; \theta) p(Z|D; \theta(t))$
- ▶ Dans cette expression :
 - ▶ D et $\theta(t)$ sont des constantes
 - ▶ z est une variable aléatoire de densité $p(z|x; \theta(t))$, Z est l'ensemble des variables z
 - ▶ θ représente les paramètres que l'on veut estimer

Algorithme EM – Etape E

- ▶ On va montrer que $Q(\theta, \theta(t))$ est une borne inférieure de $L(\theta)$
- ▶ Pour cela on utilisera **l'inégalité de Jensen**
- ▶ Théorème
 - ▶ Soit f une fonction convexe définie sur R et x une variable aléatoire réelle, alors
 - ▶ $E[f(x)] \geq f(E[x])$
 - ▶ Si f est strictement convexe $E[f(x)] = f(E[x])$ si et seulement si x est une constante
- ▶ Rq
 - ▶ f est convexe si $f''(x) \geq 0 \forall x$
 - ▶ Pour une fonction concave, l'inégalité est vérifiée dans le sens opposé $E[f(x)] \leq f(E[x])$

Algorithme EM – Etape E

- ▶
$$\begin{aligned} \log p(D; \theta) &= \sum_{i=1}^N \log p(x^i; \theta) \\ &= \sum_{i=1}^N \log \sum_{z^i} p(x^i, z^i; \theta) \\ &= \sum_{i=1}^N \log \sum_{z^i} \frac{p(z^i|x^i; \theta(t))p(x^i, z^i; \theta)}{p(z^i|x^i; \theta(t))} \\ &= \sum_{i=1}^N \log E_{z^i \sim p(z|x; \theta(t))} \frac{p(x^i, z^i; \theta)}{p(z^i|x^i; \theta(t))} \\ &\geq \sum_{i=1}^N E_{z^i \sim p(z|x; \theta(t))} \log \frac{p(x^i, z^i; \theta)}{p(z^i|x^i; \theta(t))} \\ &= \sum_{i=1}^N \sum_{z^i} p(z^i|x^i; \theta(t)) \log \frac{p(x^i, z^i; \theta)}{p(z^i|x^i; \theta(t))} \end{aligned}$$
- ▶ On a utilisé la concavité de la fonction log pour l'inégalité
- ▶ Comme dans l'étape M, on maximise par rapport à θ , on peut éliminer le dénominateur du log dans l'expression précédente, on retrouve alors
- ▶
$$Q(\theta, \theta(t)) = \sum_{i=1}^N \sum_{z^i} p(z^i|x^i; \theta(t)) \log p(x^i, z^i; \theta)$$
- ▶ En maximisant Q , on maximise une borne inférieure de la vraisemblance

Algorithme EM

Initialiser $\theta(0)$

1. Etape E : *Espérance*

On calcule $p(Z|D, \theta(t))$

On en déduit $Q(\theta; \theta(t))$

L'espérance est calculée par rapport à la distribution de Z pour les paramètres courants $\theta(t)$

2. Etape M : *Maximisation*

Etant donnée la distribution courante sur Z , trouver les paramètres qui maximisent Q

$$\theta(t + 1) = \operatorname{argmax}_{\theta} E_{Z \sim p(z|x; \theta(t))} [\log p(D, Z; \theta)]$$

- ▶ L'algorithme converge vers un maximum local de la fonction Q et de $p(D; \theta)$

► Remarques

- ▶ Lors de l'étape E , on estime la distribution de $Z : P(Z|D; \theta(t))$ à partir des valeurs courantes des paramètres $\theta(t)$
- ▶ Au lieu d'essayer de maximiser directement, $P(D|\theta)$, on utilise la fonction auxiliaire Q .
- ▶ On peut montrer la convergence de l'algorithme par :

- ▶ $Q(\theta(t+1), \theta(t)) \geq Q(\theta(t), \theta(t)) \Rightarrow p(D; \theta(t+1) \geq p(D; \theta(t))$
- ▶ L'algorithme est utilisé pour
 - ▶ les algorithmes non supervisés, semi - supervisés
 - ▶ les données manquantes ou les composantes manquantes dans les données
 - ▶ les HMM ...

Mélange de densités – cas gaussien

- ▶ On suppose que le modèle génératif des données est un mélange de densités gaussiennes
 - ▶ On fixe a priori le nombre de composantes du mélange à k
 - ▶ Pour simplifier, on suppose que les données x sont unidimensionnelles
 - ▶ $p(x) = \sum_{l=1}^k p(l)p(x|l), \quad p(x|l) = \frac{1}{(2\pi\sigma_l^2)^{\frac{1}{2}}} \exp\left(-\frac{(x-\mu_l)^2}{2\sigma_l^2}\right)$
- ▶ Paramètres
 - ▶ Coefficients du mélange $\phi_l = p(l)$, moyennes μ_l et écarts types σ_l :

$$\theta = \{p(l), \mu_l, \sigma_l; l = 1 \dots k\}$$

▶ **Log-vraisemblance**

▶ $\log p(D; \theta) = \sum_{i=1}^N \log \sum_{l=1}^k p(l; \theta)p(x^i | l; \theta)$

▶ **Vraisemblance complète**

▶ $\log p(D, Z; \theta) = \sum_{i=1}^N \log(p(z^i; \theta)p(x^i | z^i; \theta)) = \sum_{i=1}^N \sum_{l=1}^k \delta_{l z^i} \log(p(z^i; \theta)p(x^i | l; \theta))$

▶ **Fonction auxiliaire**

▶ $Q(\theta, \theta(t)) = E_{z \sim p(z|x; \theta(t))} [\log p(D, Z; \theta)]$

▶ $= \sum_{z^1=1}^k \dots \sum_{z^N=1}^k \log p(D, Z; \theta) \prod_{i=1}^N p(z^i | x^i; \theta(t))$

▶ $= \sum_{i=1}^N \sum_{l=1}^k p(l | x^i; \theta(t)) \log(p(l; \theta)p(x^i | l; \theta))$

Mélange de densité – Etapes E et M

▶ Etape E

$$\triangleright p(z^i = j | x^i; \theta(t)) = \frac{p(x^i | z^i=j; \theta(t))p(z^i=j; \theta(t))}{\sum_{l=1}^k p(x^i | z^i=l; \theta(t))p(z^i=l; \theta(t))} \quad \forall i, j$$

▶ Etape M

- ▶ $\text{Min}_{\theta}(-Q(\theta, \theta(t)))$ sous la contrainte $\sum_{l=1}^k p(l; \theta) = 1$
- ▶ Équivalent à
- ▶ $\text{Min}_{\theta}(-Q(\theta, \theta(t)) + \lambda(\sum_{l=1}^k p(l; \theta) - 1)$
 - ▶ Avec λ le coefficient de Lagrange associé

Mélange de densités – Reestimation dans l'étape M

- ▶ à l'étape M à l'itération t , on obtient les formules de reestimation suivantes

$$\textcolor{blue}{\triangleright p(j) = \frac{1}{N} \sum_{i=1}^N p(z^i = j | x^i; \theta(t))}$$

$$\textcolor{blue}{\triangleright \mu_j = \frac{\sum_{i=1}^N p(z^i=j | x^i; \theta(t)) x^i}{\sum_{i=1}^N p(z^i=j | x^i)}}$$

$$\textcolor{blue}{\triangleright \sigma_j = \frac{\sum_{i=1}^N p(z^i=j | x^i; \theta(t)) (x^i - \mu_j)^2}{\sum_{i=1}^N p(z^i=j | x^i)}}$$

Apprentissage non supervisé

Mélange de densités

Apprentissage par échantillonnage de Gibbs

Les méthodes MCMC

Markov Chain Monte Carlo

- ▶ Méthodes de calcul intensif basées sur la simulation pour
 - ▶ Echantillonnage de variables aléatoires
 - ▶ $\{x^t\}_{t=1..T}$ qui suivent une certaine distribution $p(x)$
 - ▶ Calcul de l'espérance de fonctions suivant cette distribution
 - ▶ $E[f(x)]$ sera estimé par $\frac{1}{T} \sum_{t=1}^T f(x^t)$
 - ▶ e.g. moyenne, marginales, ...
 - ▶ Maximisation de fonctions
 - ▶ $\text{Argmax}_x p(x)$

Echantillonneur de Gibbs

- ▶ On veut estimer une densité $p(\mathbf{x})$, $\mathbf{x} \in R^n$, avec $\mathbf{x} = (x_1, \dots, x_n)$
- ▶ Hypothèse
 - ▶ On connaît les lois conditionnelles
 - ▶ $p(x_i | x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = p(x_i | x_{-i})$
- ▶ Algorithme
 - ▶ Initialiser $\{x_i, i = 1..n\}$
 - ▶ Itérer jusqu'à convergence (variable d'itération notée t)

Echantillonner $x_1^{t+1} \sim p(x_1 | x_{-1}^t)$

.....

Echantillonner $x_n^{t+1} \sim p(x_n | x_{-n}^t)$

▶ Propriétés

- ▶ Sous certaines conditions de régularité, la procédure converge vers la distribution cible $p(x)$
 - ▶ Les échantillons résultants sont des échantillons de la loi jointe $p(x)$
- ▶ On n'a pas besoin de connaître la forme analytique des $p(x_i|x_{-i})$ mais uniquement de pouvoir échantillonner à partir de ces distributions
 - ▶ Mais la forme analytique permet d'avoir de meilleurs estimés
- ▶ Avant de retenir les points échantillons, on autorise souvent une période de "burn-in" pendant laquelle on fait simplement tourner l'algorithme "à vide"
- ▶ Gibbs facile à implémenter, adapté aux modèles hiérarchiques (cf LDA)

Cas du mélange de deux lois gaussiennes

- ▶ Modèle : $p(x) = \sum_{l=1}^2 p_l p(x|l)$
 - ▶ On considère des gaussiennes à une dimension
- ▶ On va considérer un modèle augmenté en ajoutant une variable cachée h
 - ▶ Les données complètes sont les (x^i, h^i)
- ▶ Les paramètres à estimer sont : $W = \{p_l, \mu_l, \sigma_l; l = 1, 2\}$
- ▶ On va utiliser Gibbs en échantillonnant sur les densités conditionnelles
 - ▶ Pour simplifier on suppose dans l'exemple que les proportions p_l et les variances σ_l sont fixées, on estime juste les moyennes μ_1, μ_2
 - ▶ Pour cela, on va échantillonner suivant la distribution jointe (h, μ_1, μ_2)

Echantillonneur de Gibbs pour le modèle de mélange de deux gaussiennes

- ▶ Choisir des valeurs initiales $\mu_1(0), \mu_2(0)$
- ▶ Répéter (t) jusqu'à convergence
 - ▶ Pour $i = 1 \dots N$
 - Générer $h^i(t) \in \{0,1\}$ selon la distribution
$$p(h^i = 1) = \frac{p_1(t-1)p(x^i|\mu_1(t-1), \sigma_j)}{p_1(t-1)p(x^i|\mu_1(t-1), \sigma_1) + p_2(t-1)p(x^i|\mu_2(t-1), \sigma_2)}$$
 - ▶ Calculer
 - $\hat{\mu}_1 = \frac{\sum_{i=1}^N h^i(t)x^i}{\sum_{i=1}^N h^i(t)}$
 - $\hat{\mu}_2 = \frac{\sum_{i=1}^N (1-h^i(t))x^i}{\sum_{i=1}^N (1-h^i(t))}$
 - Générer $\mu_j(t) \sim N(\hat{\mu}_j, \sigma_j)$, $j = 1, 2$

Lien avec l'algorithme EM

- ▶ Les étapes pour cet exemple sont les mêmes que avec EM
- ▶ Différence
 - ▶ Au lieu de maximiser la vraisemblance, aux étapes 1 et 2, on échantillonne
 - ▶ Etape 1 : on simule les variables cachées z au lieu de calculer $E(z|W, D)$
 - ▶ Etape 2 : on simule à partir de $p(\mu_1, \mu_2; Z, D)$ au lieu de calculer le max. vraisemblance $p(\mu_1, \mu_2; D)$ dans EM

Apprentissage non supervisé

Spectral Clustering

Spectral Clustering (after Von Luxburg 2007)

▶ Intuition

- ▶ $\mathbf{x}_1, \dots, \mathbf{x}_n$ data points, w_{ij} similarity between \mathbf{x}_i and \mathbf{x}_j
- ▶ $G = (V, E)$ graph
 - ▶ vertex v_i corresponds to data point \mathbf{x}_i
 - ▶ Edges are weighted by w_{ij}
- ▶ Clustering amounts at finding a graph partition such that
 - ▶ Edges between clusters have low weights
 - ▶ Edges among points inside a cluster have high values

Spectral Clustering

► Graphs notations

- ▶ $G = (V, E)$ undirected graph
 - ▶ $V = \{v_1, \dots, v_n\}$
 - ▶ Edges are weighted, $W = (w_{ij})_{i,j=1 \dots n}$, $w_{ij} \geq 0$ is the weight matrix
- ▶ D : diagonal matrix with $d_i = \sum_{j=1}^n w_{ij}$

Spectral Clustering

- ▶ Building similarity graphs from data points
 - ▶ Different ways to build a similarity graph
 - ▶ Build a locally connected graphs: k-nearest neighbor graphs
 - ▶ Two vertices are connected if one of them is among the k-nearest neighbor of the other
 - ▶ Or two vertices are connected if both are in the k-neighborhood of the other
 - ▶ Edges are then weighted using the similarity of the vertices
 - ▶ Build a fully connected graphs
 - ▶ $w_{ij} = \exp(-\|x_i - x_j\|^2 / 2\sigma^2)$

Spectral Clustering

▶ Graph Laplacians

▶ Unnormalized graph Laplacian

$$\triangleright L = D - W$$

▶ Normalized graph Laplacians

$$\triangleright L_{sym} = D^{-\frac{1}{2}}LD^{-\frac{1}{2}} = I - D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$$

symmetric

$$\triangleright L_{rw} = D^{-1}L = I - D^{-1}W$$

interpretation : random
walk on the graph

Spectral Clustering

- ▶ Properties of the unnormalized graph Laplacian L
- ▶ Proposition I- L satisfies:
 - ▶ $\forall y \in R^n, y^T Ly = \frac{1}{2} \sum_{i,j=1}^n w_{ij} (y_i - y_j)^2$
 - ▶ L is symmetric, positive semi-definite
 - ▶ The smallest eigenvalue of L is 0, the corresponding eigenvector is $\mathbf{1}$ (vector with n 1s)
 - ▶ L has n non negative eigenvalues $0 = \lambda_1 \leq \dots \leq \lambda_n$
- ▶ Proof
 - ▶ $y^T Ly = y^T Dy - y^T Wy = \sum_{i=1}^n d_i f_i^2 - \sum_{i,j=1}^n f_i f_j w_{ij} = \frac{1}{2} (\sum_{i=1}^n d_i f_i^2 - 2 \sum_{i,j=1}^n f_i f_j w_{ij} + \sum_{j=1}^n d_j f_j^2) = \frac{1}{2} \sum_{i,j=1}^n w_{ij} (y_i - y_j)^2$
 - ▶ Symmetry follows from the symmetry of L and W , positive semi definiteness comes from $y^T Ly \geq 0$
 - ▶ Obvious
 - ▶ Direct consequence of properties I and 3

Spectral Clustering

- ▶ Proposition 2 (number of connected components and spectrum of L)
 - ▶ Let G be an undirected graph with non negative weights. The multiplicity k of the eigenvalue 0 of L equals the number of connected components in the graph. The eigenspace of eigenvalue 0 is spanned by the indicator vectors of these components.
 - ▶ This result provides intuition on spectral clustering algorithms
 - ▶ Connected graph
 - ▶ Let us consider first $k = 1$. If y is an eigenvector with eigenvalue 0 then $0 = y^T L y = \sum_{i=1}^n w_{ij} (y_i - y_j)^2$. Since $w_{ij} \geq 0$ this is only possible if $y_i = y_j \forall i, j$. In a graph with 1 connected component, $\mathbf{1}$ the constant vector filled with 1s is the eigenvector with eigenvalue 0. It is also the indicator vector of the connected component
 - ▶ Multiple components
 - ▶ If there are k connected components. The W matrix and hence L can be arranged in a block diagonal form. The above result holds for each connected component A_i . The corresponding eigenvectors are the $\mathbf{1}_{A_i}$ with 1s corresponding to the i^{th} block and 0s everywhere else.
 - ▶ If one can identify these eigenvectors, one will identify the clusters
 - This is only intuition, the situation is usually more complex (see Luxburg 2007)

Spectral Clustering

► Properties of the normalized graph Laplacians

- ▶ $\forall y \in R^n, \quad y^T L_{sym} y = \frac{1}{2} \sum_{i,j=1}^n w_{ij} \left(\frac{y_i}{\sqrt{d_i}} - \frac{y_j}{\sqrt{d_j}} \right)^2$
- ▶ L_{sym} and L_{rw} are positive semi-definite and have n non negative eigenvalues
 $0 = \lambda_1 \leq \dots \leq \lambda_n$
- ▶ λ is an eigenvalue of L_{rw} with eigenvector u iff λ is an eigenvalue of L_{sym} with eigenvector $D^{1/2}u$

Unnormalized spectral clustering algorithm

- ▶ Idea
 - ▶ Project data points $x_i \in R^m, i = 1 \dots n$, in a smaller dimensional space, say of dimension k where clustering is performed
- ▶ Input: n points $\mathbf{x}_1, \dots, \mathbf{x}_n$, similarity matrix S
- ▶ Output: clusters
 - ▶ Construct similarity graph and corresponding weight matrix W
 - ▶ Compute unnormalized Laplacian L
 - ▶ Compute first eigenvectors of L (corresponding to smallest eigenvalues): u_1, \dots, u_k
 - ▶ $U: n \times k$ matrix with columns u_1, \dots, u_k
 - ▶ For $i = 1 \dots n$, $y_i \in R^k$ i -th row of U
 - ▶ Cluster $y_i, i = 1 \dots n$ with k -means into clusters C_1, \dots, C_k
- ▶ k clusters in the initial space: $C'_1, \dots, C'_k / C'_i = \{\mathbf{x}_j / y_j \in C_i\}$
- ▶ Note: Similar algorithms with normalized Laplacians

Apprentissage non supervisé

Non Negative Matrix Factorization

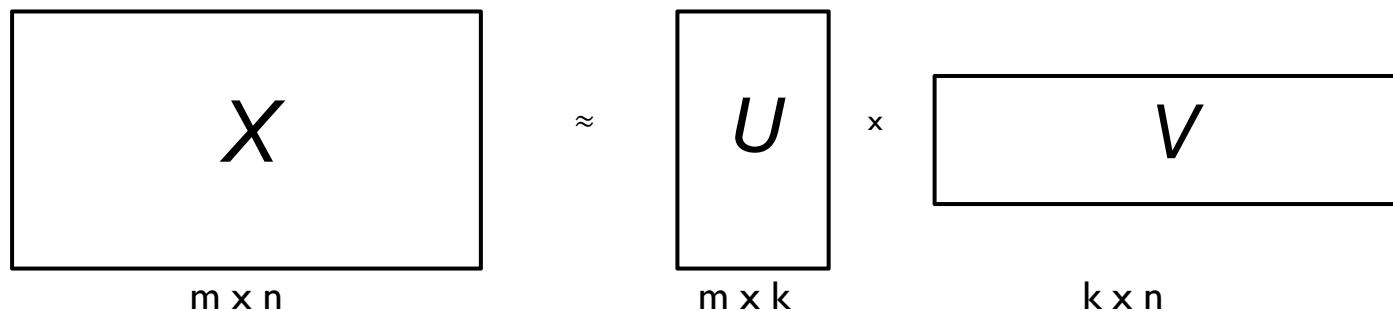
Matrix Factorization

▶ Idea

- ▶ Project data vectors in a latent space of dimension $k < m$ size of the original space
- ▶ Axis in this latent space represent a new basis for data representation
- ▶ Each original data vector will be approximated as a linear combination of k basis vectors in this new space
- ▶ Data are assigned to the nearest axis
- ▶ This provide a clustering of the data

Matrix factorization

- ▶ $X = \{x_1, \dots, x_n\}, x_i \in R^m$
- ▶ X $m \times n$ matrix with columns the x_i 's
- ▶ Low rank approximation of X
 - ▶ Find factors U, V , / $X \approx UV$
 - ▶ With U an $m \times k$ matrix, V a $k \times n$ matrix, $k < m, n$



- ▶ Many different decompositions
 - ▶ e.g. Singular Value Decomposition, Non Negative Matrix Factorization, Tri factorization, etc

▶ Applications

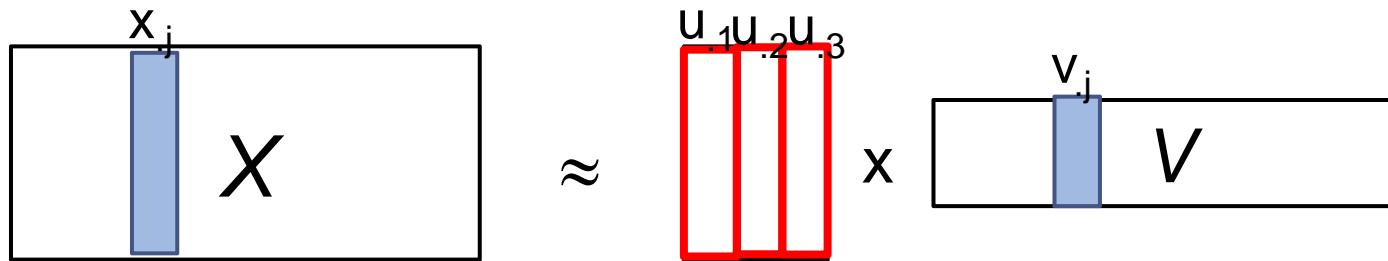
- ▶ Recommendation (User x Item matrix)
 - ▶ Matrix completion

	Star wars	Terminator	Titanic	Scarface
Bob	5	4	3	?
Alice	4	?	5	3
John	?	5	3	4
—				

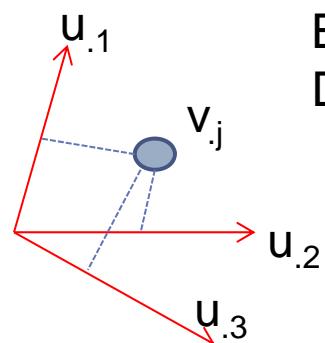
- ▶ Link prediction (Adjacency matrix)
- ▶ ...

► $X \approx UV$

- $x_{.j} = \sum_{j=1}^k u_{.i} v_{ij}$
- Columns of U , $u_{.j}$ are basis vectors, the v_{ij} are the coefficient of $x_{.j}$ in this basis



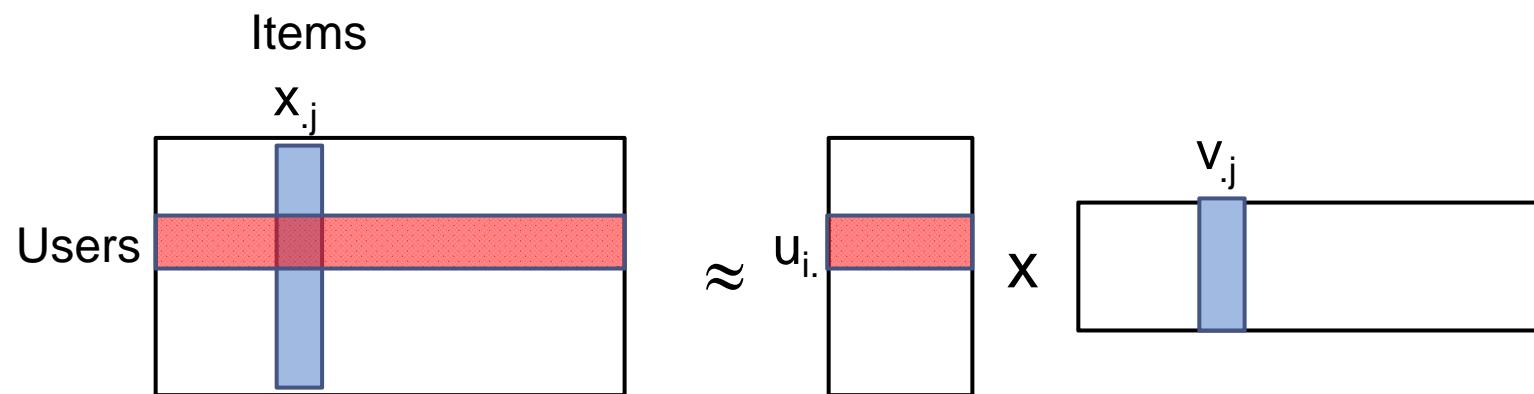
Original data



Basis vectors Representation
Dictionary

▶ Interpretation

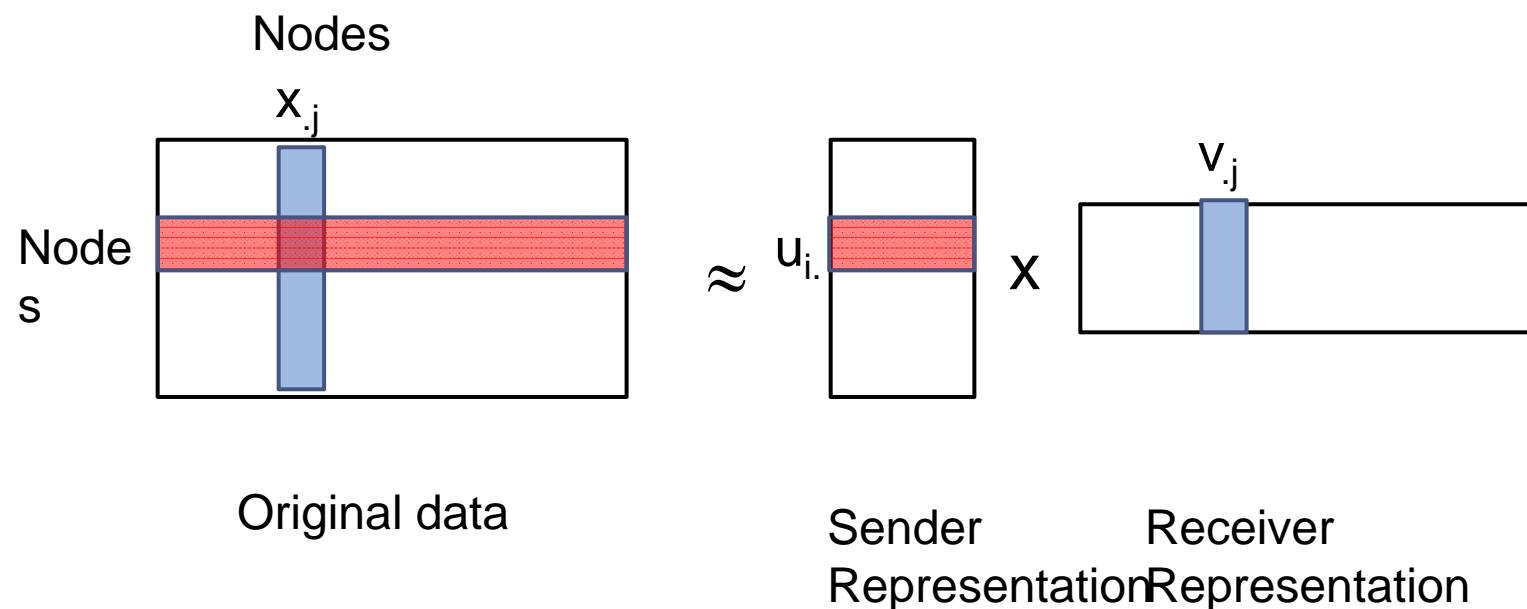
- ▶ If X is a User \times Item matrix



Original data

- ▶ Users and items are represented in a common ~~common~~ space of size k
- ▶ Their interaction is measured by a dot product in this space

- ▶ Interpretation
 - ▶ If X is a directed graph adjacency or weight matrix



- ▶ Loss function - example
- ▶ Minimize $C = \|X - UV\|^2 + c(U, V)$
 - ▶ constraints on U, V : $c(U, V)$ e.g.
 - ▶ Positivity (NMF)
 - ▶ Sparsity of representations, e.g. $\|V\|_1$, group sparsity, ...
 - ▶ Overcomplete dictionary U : $k > m$
 - ▶ Symmetry, e.g. trifactorisation
 - ▶ Bias on U and V
 - e.g. biases of users (low scores) or item (popularity) for recommendation
 - ▶ Multiple graphs
 - ▶ Any a priori knowledge on U and V

Non Negative Matrix Factorization

- ▶ Loss function

- ▶ Solve

$$\text{Min}_{\{U,V\}} \|X - UV\|^2$$

Under constraints $U, V \geq 0$

i.e. the factors are constrained to be non negative

- ▶ Convex loss function in U and in V, but not in both U and V

▶ Algorithm

- ▶ Constrained optimization problem
- ▶ Can be solved by a Lagrangian formulation
 - ▶ Iterative multiplicative algorithm (Xu et al. 2003)
 - U, V initialized at random values
 - Iterate until convergence
 - $u_{ij} \leftarrow u_{ij} \frac{(xV^T)_{ij}}{(UVV^T)_{ij}}$
 - $v_{ij} \leftarrow v_{ij} \frac{(x^TU)_{ij}}{(V^TU^TU)_{ij}}$
 - ▶ Or by projected gradient formulations
 - ▶ The solution U, V is not unique, if U, V is solution, then $UD, D^{-1}V$ for D diagonal positive is also solution

▶ Using NMF for Clustering

- ▶ Normalize U as a column stochastic matrix (each column vector is of norm 1)
 - ▶ $u_{ij} \leftarrow \frac{u_{ij}}{\sqrt{\sum_i u_{ij}^2}}$
 - ▶ $v_{ij} \leftarrow v_{ij} \sqrt{\sum_i u_{ij}^2}$
- ▶ Under the constraint “U normalized” the solution U,V is unique
- ▶ Associate x^i to cluster j if $j = argmax_j(v_{ji})$

- ▶ Note
 - ▶ many different versions and extensions of NMF
 - ▶ Different loss functions
 - ▶ e.g. different constraints on the decomposition
 - ▶ Different algorithms
- ▶ Applications
 - ▶ Clustering
 - ▶ Recommendation
 - ▶ Link prediction
 - ▶ Etc
- ▶ Specific forms of NMF can be shown equivalent to
 - ▶ PLSA
 - ▶ Spectral clustering

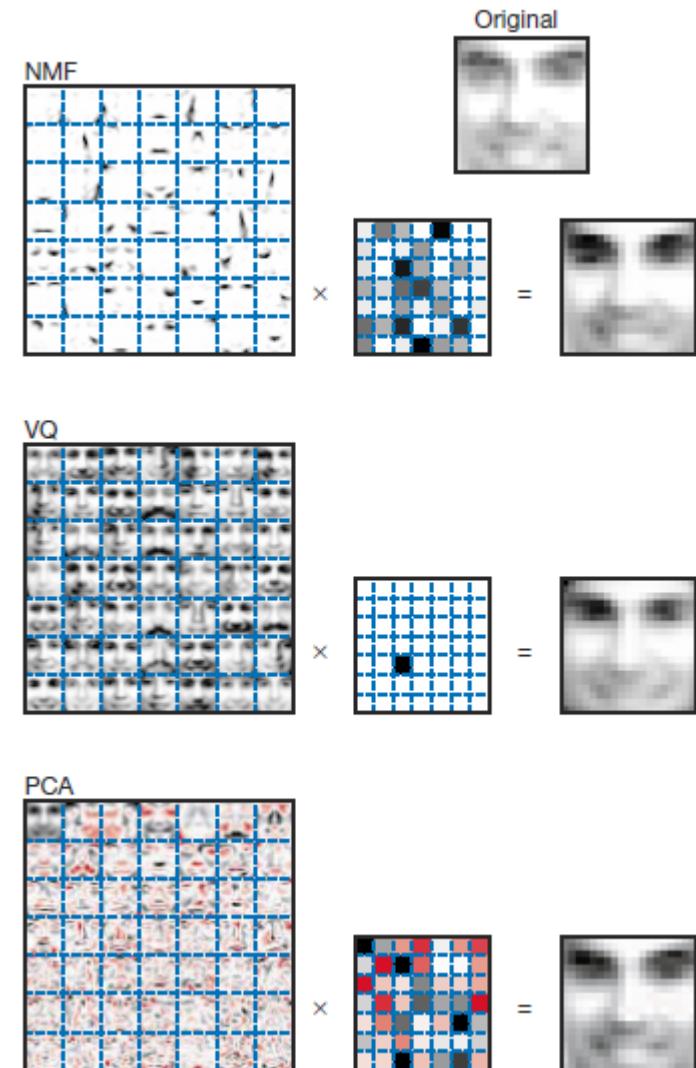
Illustration (Lee & Seung 1999)

- ▶ Basis images for

- ▶ NMF

- ▶ Vector Quantization

- ▶ Principal Component Analysis



▶ Citations

- ▶ Ulrike Luxburg. 2007. A tutorial on spectral clustering. *Statistics and Computing* 17, 4 (December 2007), 395-416.
- ▶ Daniel D. Lee and H. Sebastian Seung (1999). "Learning the parts of objects by non-negative matrix factorization". *Nature* 401 (6755): 788–791.

Some useful links

- ▶ Books
 - ▶ Closest to this course
 - Cornuéjols A and Miclet L.: Apprentissage Artificiel. Concepts et algorithmes (2nd ed.with revisions and additions - 2006 Eyrolles,
 - Christopher M. Bishop, Pattern Recognition and Machine Learning, Springer (2006)
 - ▶ Most useful
 - Hastie T, Tibshirani R, and Friedman J, (2009). The Elements of Statistical Learning (2nd edition), Springer-Verlag.
 - ▶ pdf version at <http://statweb.stanford.edu/~tibs/ElemStatLearn/>
 - David Barber, 2012, Bayesian Reasoning and Machine Learning, Cambridge Univ. Press.
- ▶ Courses on the web
 - ▶ Andrew Ng course on Coursera is one of the most popular
 - ▶ Caltech course: <https://work.caltech.edu/telecourse.html>
 - ▶ Oxford course: <https://www.cs.ox.ac.uk/people/nando.defreitas/machinelearning/>
 - ▶ And many others
- ▶ Software
 - ▶ General
 - ▶ Python libraries, e.g. Scikit-learn, <http://scikit-learn.org/> and other languages (Matlab, R)
 - ▶ Weka 3: Data Mining Software in Java
 - <http://www.cs.waikato.ac.nz/ml/weka/>
 - ▶ SVM
 - ▶ <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>
 - ▶ <http://svmlight.joachims.org/>
 - ▶ Neural networks
 - ▶ <http://torch.ch/>
 - ▶ <http://deeplearning.net/software/theano/> propose GPU codes
 - ▶ <http://caffe.berkeleyvision.org/> initially proposed for vision applications
- ▶ Test sets
 - ▶ UCI machine learning repository
 - ▶ Kaggle Site
 - ▶