Cours IFT6266,
Éléments pratiques de l'utilisation des réseaux multi-couches

- On discute ici des techniques permettant l'apprentissage réussi des réseaux de neurones multi-couche, avec généralement une connectivité complète entre une couche et la suivante. Pour plus de détails, voir le livre de Bishop (*Pattern Recognition and Machine Learning*), le tutorial de Yann Le Cun à NIPS'93 `http://www.iro.umontreal.ca/~bengioy/ift6266/mlp/YannNipsTutorial.ps`, et la thèse de Léon Bottou.

  Deux principes guident les choix discutés ici:

  - contrôle de la capacité et autres techniques pour améliorer la généralisation,
  - amélioration du processus d'optimisation.

- **Problème fondamentalement difficile d'optimisation**

  L'optimisation du critère d'apprentissage dans les réseaux de neurones multi-couche est difficile car il y a de nombreux minima locaux. On peut même démontrer que de trouver les poids optimaux est NP-dur. Cependant on se contente de trouver un bon minimum local, ou même simplement une valeur suffisamment basse du critère. Comme ce qui nous intéresse est la généralisation et non pas l'erreur d'apprentissage (ce qu'on minimise n'est pas ce qu'on voudrait vraiment minimiser), la différence entre "près d'un minimum" et "au minimum" est sans importance. Par ailleurs, comme il n'y a pas de solution analytique au problème de minimisation, on est forcé de faire cette optimisation de manière itérative.

- **Choix de l'architecture**
  En principe, une manière d'accélérer la descente de gradient est de faire des choix qui rendent la matrice Hessienne $\frac{\partial^2 C}{\partial \theta_i \partial \theta_j}$ mieux conditionnée. On peut montrer que le nombre d'itération d'un algorithme de descente de gradient sera proportionnel au ratio de la plus grande à la plus petite valeur propre de la matrice Hessienne (avec une approximation quadratique de la fonction de coût).

  - En théorie **une couche cachée** suffit. En pratique cela s'avère vrai dans la plupart des cas sauf les réseaux de neurones à convolution (qui peuvent avoir 5 ou 6 couches par exemple). Parfois on obtient de bien meilleurs résultats avec 2 couches cachées. *En fait on peut obtenir une*

bien meilleure généralisation avec encore plus de couches, mais une initialisation aléatoire ne fonctionne pas bien avec plus de deux couches (mais voir les travaux récents sur l'initialisation non-supervisée vorace).

- Pour la **régression** ou **cibles réelles** et non-bornées en général, il vaut généralement mieux utiliser des neurones **linéaires** à la couche de sortie. Pour la **classification**, il vaut généralement mieux utiliser des neurones avec non-linéralité (sigmoide ou softmax) à la couche de sortie.
- Dans certains cas une connexion directe entre l'entrée et la sortie peut être utile. Dans le cas de la régression, elle peut aussi être initialisée directement par le résultat d'une régression linéaire des sorties sur les entrées. Les neurones cachées servent alors seulement à apprendre la partie non-linéaire manquante.
- Une architecture avec poids partagés, ou bien le partage de certains éléments de l'architecture (e.g., la première couche) entre les réseaux associés à plusieurs tâches connexes, peuvent significativement améliorer la généralisation. Voir aussi la discussion à venir sur les réseaux à convolution.
- Il vaut mieux utiliser utiliser une non-linéarité symmétrique dans les couches cachées (comme la tanh, et non pas la sigmoide), afin d'améliorer le conditionnement du Hessien.

- **Normalisation des entrées**
  Il est impératif que les entrées soient de moyenne pas trop loin de zéro et de variance pas trop loin de 1. Les valeurs en entrées devraient aussi ne pas avoir une magnitude trop grande. On peut faire certaines transformations monotones non-linéaires qui réduisent les grandes valeurs. Si on a une entrée très grande, elle fait saturer plusieurs neurones et bloque l'apprentissage pour cet exemple. En fait, dans le cas linéaire, le conditionnement du Hessien est optimal quand les entrées sont normalisées (donc avec matrice de covariance = identité), ce qui peut se faire en les projetant dans l'espace des vecteurs propres de la matrice $X'X$, ou X est la matrice nombre d'exemples $\times$ nombre d'entrées.

- **Traitement des sorties désirées**
  Dans le cas d'apprentissage par minimisation d'un coût quadratique, on doit s'assurer que les sorties désirées

  - sont toujours dans l'intervalle des valeurs que la non-linéarité de la couche de sortie peut produire (et sont à peu près normales N(0,1) dans le cas linéaire),

- ne sont pas trop proches des valeurs limites de la non-linéarité de la couche de sortie: pour la classification, une valeur optimale est près d'un des deux points d'inflexion (i.e., les points de courbure (dérivée seconde) maximale, environ -0.6 et 0.6 pour tanh, 0.2 et 0.8 pour la sigmoide).
- Il vaut mieux utiliser le critère d'entropie croisée (maximum de vraisemblance conditionnelle) pour la classification probabiliste, ou bien le critère de marge (comme pour le perceptron mais en pénalisant les écarts à la surface de décision au-delà d'une marge): $\sum_i (-1^{1_{y=i}} f_i(x) + 1)_+$, où $x_+ = x 1_{x>0}$ et $f_i(x)$ est la sortie (sans non-linéarité) pour la classe i.

- **Codage des sorties désirées et des entrées discrètes**
  Le codage classique pour la classification utilise un neurone de sortie par classe, avec une valeur désirée haute pour le neurone de la classe correcte, et une valeur désirée faible pour les autres classes. De la même manière, des entrées discrètes avec des valeurs non-ordinales devraient être codées avec autant de neurones que de valeur possibles, selon un schema similaire. Dans ce cas, une solution souvent adoptée est d'utiliser 0 pour la valeur faible et 1 pour la valeur haute. D'autres codages sont possibles et même désirables si plus d'information sur la structure de ces symboles est connue (par exemple plusieurs "traits" binaires caractérisant chaque classe).

- **Algorithme d'optimisation**
  Quand le nombre d'exemples est grand (plusieurs milliers) la descente de gradient stochastique est souvent le meilleur choix (surtout pour la classification), en terme de vitesse et en terme de contrôle de la capacité (il est plus difficile d'avoir de l'overfitting avec la descente de gradient stochastique). La descente de gradient stochastique ne tombe pas dans les minima très pointus (qui ne généralisent pas bien, car une légére perturbation des données déplaçant la surface d'erreur donnerait une très mauvaise performance), à cause du bruit induit par le pas de gradient et le gradient "bruité". Ce gradient bruité aide aussi à sortir de certains minima locaux. Quand la descente de gradient stochastique est utilisé, il est IMPÉRATIF que les exemples soient "bien mélangés": par exemple si on a beaucoup d'exemples consécutifs de la même classe, la convergence sera très lente. Il suffit de permuter aléatoirement les exemples une fois pour toute, pour éliminer toute dépendence entre les exemples successifs. En principe le pas de gradient devrait être graduellement réduit pendant l'apprentissage. Pour certains problèmes (surtout de classification) cela ne semble pas absolument nécessaire. Une cédule de descente raisonnable est par exemple $\epsilon_t = \frac{\epsilon_0}{\alpha t+1}$. Si

on pouvait le calculer, le pas de gradient optimal serait $\frac{1}{\lambda_{\max}}$, i.e., l'inverse de la valeur propre la plus grande de la matrice Hessienne, et le pas de gradient maximal (avant divergence) est deux fois plus grand. Le Cun propose une méthode pour estimer efficacemenet $\lambda_{\max}$ (voir son tutorial sur le sujet).

Quand le nombre d'exemples (et donc de paramètres) est plus petit, et surtout pour la régression, les techniques du second degré (surtout la technique des gradients conjugués) permettent une convergence beaucoup plus rapide. Ces techniques sont "batch" (modification des paramètres après calcul de l'erreur et gradient sur tous les exemples). Ces techniques sont généralement plus facile à "tuner" que la descente de gradient stochastique, mais la généralisation est parfois moins bonne à cause de la facilité de tomber dans des minima pointus.

Jusqu'à quelques dizaines de milliers d'exemples, la descente de gradient conjugués reste une des meilleures techniques pour l'optimisation des réseaux de neurones. Au-delà il vaut généralement mieux s'en tenir au gradient stochastique.

- **Initialisation des paramètres**
  On ne peut initialiser tous les poids à zéro sans quoi tous les neurones cachés sont condamnés à toujours faire la même chose (qu'on peut voir par simple symmétrie). On veut aussi éviter la saturation des neurones (sortie près des limites de la non-linéarité $\rightarrow$ gradient presque 0), mais ne pas être trop près initialement d'une fonction linéaire. Quand les paramètres sont tous près de 0, le réseau multicouche calcule une transformation affine (linéaire), donc sa capacité effective par sortie est égale au nombre d'entrées plus 1. En se basant sur ces considérations, le point idéal d'opération du neurone est le point d'inflexion de la non-linéarité (entre la partie linéaire près de l'origine et la partie "saturation"). Pour atteindre cet objectif, on peut argumenter que les poids initiaux devraient être initialisés de manière uniforme dans un intervalle $[-K/\sqrt{n}, K/\sqrt{n}]$, où n est le *fan-in*, i.e., le nombre d'entrées du neurone, et K est proche de l'unité. Voir la thèse de Léon Bottou (en français, disponible sur le site web du cours).

- **Contrôle de la saturation**
  Un des problèmes fréquents pendant l'apprentissage est la saturation des neurones, souvent dûe à une mauvaise normalisation des entrées ou des sorties désirées. On peut contrôler cela on observant la distribution des sorties des neurones (en particulier, la moyenne des valeurs absolues de la somme pondérée est un bon indice). Quand les neurones saturent fréquemment, l'apprentissage est bloqué sur un plateau de la fonction de coût dû à de très

petits gradients sur certains paramètres (donc un très mauvais conditionnement du Hessien).

- **Contrôle de la capacité effective**
  La théorie du *structural risk minimization* de Vapnik nous dit qu'il existe une capacité optimale autour de laquelle l'erreur de généralisation augmente (c'est un minimum global et unique). Les techniques de contrôle de la capacité effective visent donc à chercher ce minimum (evidemment de façon approximative).

  - **early stopping**: il s'agit d'une des techniques les plus populaires et les plus efficaces, mais elle ne marche pas bien quand le nombre d'exemples disponible est très petit. L'idée est très simple: on utilise un ensemble d'exemples *de validation* non-utilisés pour l'apprentissage par descente de gradient pour estimer l'erreur de généralisation au fur et à mesure que l'apprentissage itératif progresse (normalement, après chaque époque). On garde les paramètres correspondant au minimum de cette courbe d'erreur de généralisation estimée (et on peut s'arrêter quand cette erreur commence à remonter sérieusement ou qu'un minimum a été atteint depuis un certain nombre d'époques). Cela a l'avantage de répondre à une des questions difficile de l'optimisation, qui est: quand arrêter?
  - **contrôle du nombre d'unites cachées**: ce nombre influence directement la capacité. Dans ce cas il faut malheureusement faire plusieurs d'expériences d'apprentissage, à moins d'utiliser un algorithme d'apprentissage **constructif** (qui rajoute des ressources au fur et à mesure), voir l'algorithme de cascade-correlation (Fahlman, 1990). On peut utiliser un ensemble de validation ou la validation croisée pour estimer l'erreur de généralisation. Il faut faire attention au fait que cet estimé est bruité (d'autant plus qu'il y a peu d'exemples de validation).
  - **weight decay**: il s'agit de pénaliser les poids forts, en effet, on peut montrer que la capacité est bornée par la magnitude des poids du réseau de neurones. On rajoute la pénalité

    $$\lambda \sum_i w_i^2$$

    à la fonction de coût. Comme dans le cas précédent, il faut faire plusieurs expériences d'apprentissage et choisir le facteur de pénalité λ (un **hyper-paramètre**) qui minimise l'erreur de généralisation estimée. On l'estime avec un ensemble de validation ou bien par *validation croisée* (voir feuillet sur ce sujet).