Cours IFT6266,
Modèles statistiques récurrents

Quand on traite des données temporelles (non-supervisées, on veut modéliser le processus des Y_t; ou supervisées, on veut modéliser le processus des Y_t étant donnés les X_t) on a souvent des modèles qui impliquent une **récurrence** dans le temps, soit au niveau des variables aléatoires elle-mêmes ou bien de variables intermédiaires qui définissent le modèle.
Le cas non-supervisé est souvent équivalent au cas non-supervisé parce que

$$P(Y_1, \ldots, Y_T) = \prod_t P(Y_t | Y_{t-1}, \ldots Y_1)$$

où l'on peut considérer $X_t = (Y_{t-1}, \ldots, Y_1)$ et un problème supervisé.
Dans la plupart de ces cas on peut utiliser le **truc du calcul efficace dans un graphe de flot** pour obtenir le gradient de la log-vraisemblance totale (ou autre critère pertinent) par rapport aux paramètres.

Modèle ARMA(p,q)

Processus observé: $X_1, \ldots, X_T$.
Processus non-observé (innovations): $\epsilon_1, \ldots, \epsilon_T$.
Paramètres: $\theta = (\sigma^2, a_1, \ldots, a_p, b_1, \ldots, b_q)$.
Hypothèse distributionnelle:

$$X_t = \epsilon_t + \sum_{i=1}^{p} a_i X_{t-i} + \sum_{j=1}^{q} b_j \epsilon_{t-j}$$

avec ϵ_t i.i.d. $\sim \mathcal{N}(0, \sigma^2)$.
On peut écrire la vraisemblance de manière récursive, par exemple en utilisant la technique de factorisation somme-produit (ici les sommes sont des intégrales, qui se font analytiquement).
Cela donne donc une formule récursive mais analytique qui relie les paramètres à la log-vraisemblance d'apprentissage.
Si on a plusieurs séquences, on additionne les log-vraisemblances de ces séquences (on suppose les séquences tirées indépendemment).

Modèle probabiliste graphique avec état aléatoire caché

Exemple: filtre de Kalman, brièvement introduit dans les notes de Aaron Courville, un modèle similaire au Hidden Markov Model à discuter plus tard.
On a une v.a. non-observée qu'on appelle **l'état**. Ici elle suit le processus suivant:

$$S_t = AS_{t-1} + \epsilon_t$$

avec les ϵ_t i.i.d. normaux, et

$$X_t = BS_t + \gamma_t$$

avec les γ_t i.i.d. normaux aussi.
On peut encore utiliser la factorisation somme-produit pour calculer la vraisemblance. On peut aussi appliquer l'algorithme EM, avec le processus des S_t la variable cachée.

Classe de fonctions de système dynamique: état déterministe caché

On construit une fonction qui produit une séquence $\hat{y}_t$ (e.g. de décisions, de probabilités, ou de densité) qui dépend d'une série d'entrées $x_1, \dots x_T$. Avec une série de sorties $y_1, \dots, y_T$ on calcule un coût C, par exemple la log-vraisemblance de la séquence $(y_1, \dots y_T)$ étant donnée la séquence $(x_1, \dots, x_T)$, ou bien le ratio de Sharpe (rendement moyen relatif divisé par écart type des rendements) pour une fonction de décision qui alloue des actifs dans un portfefeuille selon des poids $(\hat{y}_t)$ choisis à chaque jour t, étant donné des intrants $x_1 \dots x_t$.
Une structure typique pour la dynamique des $\hat{y}_t$ est la suivante:

$$\begin{aligned} h_t &= H(h_{t-1}, x_t; \theta) \\ \hat{y}_t &= F(h_t; \theta) \end{aligned}$$

où les fonctions H et F sont paramétrées par les θ. *Remarquez que les mêmes θ servent pour tous les t.*
On peut relier le coût C aux paramètres θ à travers un graphe de flot, et on peut donc l'optimiser par une optimisation à base du gradient.

Réseau de neurones récurrent

Voici un exemple où F a une structure "neuronale" et H est simplement affine. Dans ce cas on peut montrer que le système est un approximateur universel dans l'espace des séquences, et en particulier peut théoriquement implanter une machine de Turing (avec un ruban de taille finie), donc n'importe quel programme. Évidemment on n'a pas, par contre, que l'on puisse bien optimiser un tel système (ce qui est aussi le cas d'un réseau de neurones ordinaire avec unités cachées).
Considérons la dynamique suivante. Sortie affine:

$$\hat{y}_{t,i} = a_i + \sum_{j=1}^{N_h} b_{ij}\, h_{t,j}$$

où t représente le temps (de $t = 1$ à T) et i varie de 1 à M, N_h est le nombre d'unités cachées, et

$$h_{t,j} = \tanh(c_j + \sum_{k=1}^{N_h} d_{jk}\, h_{t-1,k} + \sum_{l=1}^{N} e_{jl}\, x_{t,l})$$

avec N le nombre d'entrées à chaque instant t, et M le nombre de sorties à chaque instant t. On prend arbitrairement $h_{0,j} = 0$. Les paramètres sont $\theta = (a, b, c, d, e)$ et ne dépendent pas de t et T (on dit qu'ils sont *partagés* dans le temps), *ce qui permet*

d'applique le réseau, un fois entraîné, à des séquences d'entrées de n'importe quelle longueur T. Commencez par dériver les formules de calcul du gradient pour ce réseau (en utilisant le principe du graphe de flots).
Pour faciliter les calculs, on introduit des variables intermédiares $p_{t,j}$, ce qui nous donne les formules suivantes:

$$\begin{aligned}
\hat{y}_{t,i} &= a_i + \sum_{j=1}^{N_H} b_{ij} h_{t,j} \\
h_{t,j} &= \tanh(p_{t,j}) \\
p_{t,j} &= c_j + \sum_{k=1}^{N_H} d_{jk} h_{t-1,k} + \sum_{l=1}^{N} e_{jl} x_{t,l}
\end{aligned}$$

Avec la formule de coût

$$C(\theta, x, y) = \frac{1}{2} \sum_{t,i} (\hat{y}_{t,i} - y_{t,i})^2$$

IL FAUT NOTER QUE h_t INFLUENCE h_{t+1} ET $\hat{y}_t$. DONC LES ENFANTS DE $h_{t,j}$ DANS LE DIAGRAMME DE FLOT SONT LES NOEUDS POUR $h_{t+1,i}$ ET $\hat{y}_{t,i}$.
En appliquant le principe de propagation des gradients dans le sens inverse du flot des calculs de valeurs, dans le diagramme de flot, on obtient donc les formules suivantes pour calculer les gradients:

$$\begin{aligned}
\frac{\partial C}{\partial \hat{y}_{t,i}} &= \frac{\partial \frac{1}{2} \sum_{t,i} (\hat{y}_{t,i} - y_{t,i})^2}{\partial \hat{y}_{t,i}} = \hat{y}_{t,i} - y_{t,i} \\
\frac{\partial C}{\partial a_i} &= \sum_t \frac{\partial C}{\partial \hat{y}_{t,i}} \frac{\partial \hat{y}_{t,i}}{\partial a_i} = \sum_t \frac{\partial C}{\partial \hat{y}_{t,i}} \\
\frac{\partial C}{\partial b_{ij}} &= \sum_t \frac{\partial C}{\partial \hat{y}_{t,i}} \frac{\partial \hat{y}_{t,i}}{\partial b_{ij}} = \sum_t \frac{\partial C}{\partial \hat{y}_{t,i}} h_{t,j}
\end{aligned}$$

Les unités cachées alimentent à la fois les unités de sortie et les unités cachées de la période suivante:

$$\begin{aligned}
\frac{\partial C}{\partial h_{t,j}} &= \sum_{i=1}^{M} \frac{\partial C}{\partial \hat{y}_{t,i}} \frac{\partial \hat{y}_{t,i}}{\partial h_{t,j}} + \sum_{k=1}^{N_H} \frac{\partial C}{\partial p_{t+1,k}} \frac{\partial p_{t+1,k}}{\partial h_{t,j}} \\
&= \sum_{i=1}^{M} \frac{\partial C}{\partial \hat{y}_{t,i}} b_{ij} + \sum_{k=1}^{N_H} \frac{\partial C}{\partial p_{t+1,k}} d_{kj}
\end{aligned}$$

NOTEZ BIEN LA DEUXIEME SOMME, QUI PERMET DE TENIR COMPTE DE L'EFFET RECURRENT (QUI FAIT QUE LES SORTIES A $t = s$ DEPENDENT DE TOUTES LES ENTREES QUI PRECEDENT, $t \leq s$). Il faudra donc que les gradients soient calculés de façon "antichronologique" (ce qu'on appelle **rétropropagation dans le temps**), de telle sorte que les $\frac{\partial C}{\partial p_{t+1,k}}$ soient disponibles au moment de calculer les gradients de la période t. Pour les paramètres restants, on a:

$$\frac{\partial C}{\partial p_{t,j}} = \frac{\partial C}{\partial h_{t,j}} \frac{\partial h_{t,j}}{\partial p_{t,j}} = \frac{\partial C}{\partial h_{t,j}} (1 - \tanh(p_{t,j})^2) = \frac{\partial C}{\partial h_{t,j}} (1 - h_{t,j}^2)$$

$$
\begin{aligned}
\frac{\partial C}{\partial c_j} &= \sum_t \frac{\partial C}{\partial p_{t,j}} \frac{\partial p_{t,j}}{\partial c_j} = \sum_t \frac{\partial C}{\partial p_{t,j}} \\
\frac{\partial C}{\partial d_{jk}} &= \sum_t \frac{\partial C}{\partial p_{t,j}} \frac{\partial p_{t,j}}{\partial d_{jk}} = \sum_t \frac{\partial C}{\partial p_{t,j}} h_{t-1,k} \\
\frac{\partial C}{\partial e_{jl}} &= \sum_t \frac{\partial C}{\partial p_{t,j}} \frac{\partial p_{t,j}}{\partial e_{jl}} = \sum_t \frac{\partial C}{\partial p_{t,j}} x_{t,l}
\end{aligned}
$$

Un programme qui implémente ces calculs doit donc mémoriser non seulement tous les vecteurs d'entrée et de sortie (les $x_{t,i}$, les $y_{t,j}$ et les $\hat{y}_{t,j}$), mais également les valeurs intermédiaires $p_{t,j}$ et $h_{t,j}$.
Il existe une solution qui n'est pas basée sur le diagramme de flot et qui permet de calculer les gradients dans le sens "chronologique", mais elle est beaucoup moins efficace (requiert $O(N_h^4)$ calculs et $O(N_h^3)$ mémoire, alors que l'algorithme ci-haut requiert seulement $O(N_h^2)$ calculs et mémoire, i.e., proportionnel au nombre de paramètres).