Reconstruction d'état caché avec cartes auto-organisatrices récurrentes

Alain Dutech, Jérémy Fix, Hervé Frezza-Buet

Reconstruction d'état caché avec cartes auto-organisatrices récurrentes.

A. Dutech[1] J. Fix[2] H. Frezza-Buet[2]
[1] Université de Lorraine, CNRS, Inria, LORIA ; F-54000 Nancy, France
[2] Centrale-Supélec, LORIA ; F-57070 Metz, France
contact : alain.dutech@loria.fr

Mots Clef

Reconstruction d'état, Chaînes de Markov Cachées, Cartes auto-organisatrices

1 Motivations

Quand les états d'un processus ne sont pas Markoviens (POMDP par exemple), la convergence des algorithmes d'apprentissage par renforcement n'est pas garantie. Une solution est de reconstruire un processus Markovien en partant de la séquence des états. Dans ce but, nous explorons les capacités d'architectures récurrentes qui s'appuient sur des cartes neuronales auto-organisatrices pour apprendre à prédire des séquences d'observations issues de HMM.

Les algorithmes classiques d'apprentissage par renforcement [10] offrent des garanties de convergence quand ils sont appliqués à des problèmes qui peuvent se modéliser comme des Processus Décisionnels de Markov [8]. Or, dans de nombreux cas, la séquence d'information dont dispose l'agent pour apprendre n'est pas un processus markovien, l'agent n'a pas accès au *véritable* état du système (au sens de la physique ou de l'automatique), il n'en a qu'une observation partielle, bruitée, bien incomplète.

Si on se place dans le cadre formel des POMDP ([1, 2]), une manière de procéder est de considérer que cet état d'information est constitué des n dernières paires (o, a) d'observation et d'action. Dans le cas général, n doit être infini pour s'assurer que l'état d'information ainsi extrait est bien *complet*[1].

Nous voulons ici exploiter la puissance des réseaux de neurones récurrents pour extraire des états d'information les plus complets possibles dans le cadre de processus non-Markoviens. Mais contrairement à [6, 4, 7, 3] où ces états d'information sont appris indirectement, comme un moyen pour estimer la Q-fonction, nous voulons ici apprendre explicitement à extraire des états d'information. Pour cela, nous proposons une architecture neuronale récurrente qui s'appuie sur des ≪*Dynamic Self-Organizing Maps*≫ (DSOM) [9]. Les DSOM s'apparentent aux cartes auto-organisatrices de Kohonen qui sont connues pour leur bonnes propriétés dans le cadre de la quantification vectorielle [5], elles en diffèrent par une sensibilité réduite à la densité des échantillons d'apprentissage. Cette dernière propriété nous intéresse tout particulièrement dans le cadre général de l'apprentissage par renforcement. En effet, lors de l'apprentissage, il nous paraît pertinent d'accorder *a priori* autant d'importance aux régions de l'espace sensorimoteur visitées rarement qu'aux régions visitées souvent.

1. C'est-à-dire qu'il permet de contruire un Processus Décisionnel Markovien dont la solution est équivalente au POMDP original

2 Architecture

L'architecture neuronale récurrente que nous utilisons est décrite à la figure 1. En fonction de l'observation courante et de l'état actuel du réseau (le contexte), un neurone ≪vainqueur≫ est déterminé et l'état d'information est porté par ce neurone vainqueur, cela peut être son indice ou la valeur de son prototype.

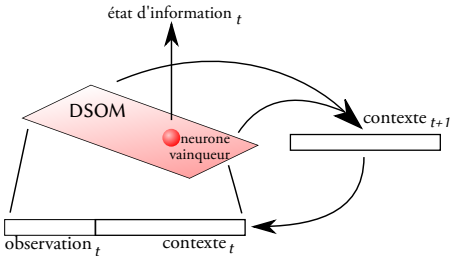

FIGURE 1 – **DSOM récurrent**.

3 Expérimentations

Dans un premier temps, nous avons vérifié que pour des HMM totalement observables (donc

Markoviens), l'architecture était convainquante. Par exemple, la figure 2 montre les états reconstruits pour un HMM qui produit la suite d'observation "`A-B-C-D-A-B-...`". Dans l'arc de cercle à droite, on positionne les derniers neurones vainqueur de cette carte mono-dimensionnelle par des petits ronds, la couleur des ronds est liée à l'observation de leur prototype et la flèche indique le contexte auquel ils réagissent. Les courbes montrent l'évolution de la distance du prototype vainqueur (observation en noir, contexte en bleu) et l'erreur de prédiction en observation. Ici, le réseau est capable d'apprendre la structure de la séquence (les ronds et flèches s'enchaînent bien).

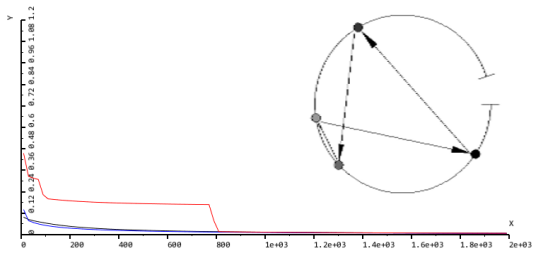

FIGURE 2 – **ABCD**.

Cela fonctionne aussi avec des HMM où il faut "compter" pour savoir quand l'observation change, comme par exemple avec des séquences du type "`A-A-A-A-A-F`", figure 3.

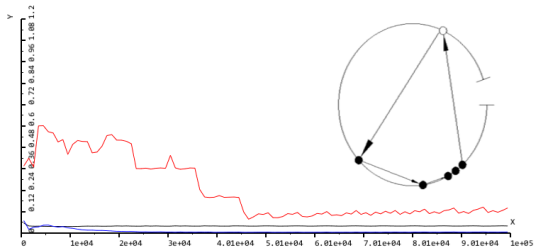

FIGURE 3 – **AAAAF**.

Mais les résultats sont – pour l'instant – décevants dans des cas comme "`A-B-C-B-A-...`", voir figure 4. Nous travaillons donc actuellement sur une meilleure différenciation des prototypes liés au contexte.

4 Références

Références

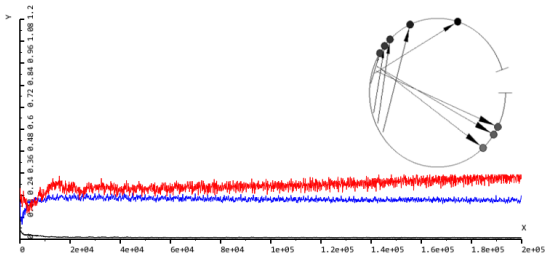

FIGURE 4 – **ABCBABCBA...**.

[1] ASTRÖM, K. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications 10* (1965), 174–205.

[2] CASSANDRA, A. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown University, Department of Computer Science, Providence, RI, 1998.

[3] DASWANI, M., SUNEHAG, P., HUTTER, M., ET AL. Feature reinforcement learning using looping suffix trees. In *10th European Workshop on Reinforcement Learning : JMLR : Workshop and Conference Proceedings 24* (2012), Journal of Machine Learning Research.

[4] DUTECH, A., AND SAMUELIDES, M. Apprentissage par renforcement pour les processus décisionnels de Markov partiellement observés. *Revue d'Intelligence Artificielle (RIA) 17(4)* (2003), 559–589.

[5] KOHONEN, T. Self-organized formation of topologically correct feature maps. *Biological Cybernetics 43* (1982), 59–69.

[6] MCCALLUM, A. Learning to use selective attention and short-term memory in sequential tasks. In *From Animals to Animats, Proc. of the Fourth Int. Conf. on Simulating Adaptive Behavior* (1996).

[7] NGUYEN, P., SUNEHAG, P., AND HUTTER, M. Feature reinforcement learning in practice. In *European Workshop on Reinforcement Learning* (2011), Springer, pp. 66–77.

[8] PUTERMAN, M. *Markov Decision Processes : discrete stochastic dynamic programming*. John Wiley & Sons, Inc. New York, NY, 1994.

[9] ROUGIER, N. P., AND BONIFACE, Y. Dynamic Self-Organising Map. *Neurocomputing 74*, 11 (2011), 1840–1847.

[10] SUTTON, R. Generalization in reinforcement learning : Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8 (NIPS)* (1996), MIT Press, pp. 1038–1044.

5 Annexe : Architecture d'un RDSOM

L'architecture neuronale récurrente que nous utilisons est décrite à la figure 1. En fonction de l'observation courante et de l'état actuel du réseau (le contexte), un neurone *vainqueur* est déterminé et l'état d'information est porté par ce neurone vainqueur, cela peut être son indice ou la valeur de son prototype.
Plus formellement, les RDSOM que nous utilisons sont constitués d'un ensemble de N neurones qui sont chacun définis par :

- $i \in [0 \dots N)$: un indice
- $pos(i) \in [0, 1]$: une position dans l'espace des neurones (ici, un espace unidimensionnel, mais c'est arbitraire)
- $w(i) = (w_{in}(i), w_{rec}(i))$: un prototype où
  - $w_{in}(i) \in [0, 1]$: vecteur de l'espace des entrée $\mathcal{X}$ (ici de dimension 1, mais c'est arbitraire).
  - $w_{rec}(i) \in [0, 1]$: vecteur des poids récurrents dans l'espace des positions des neurones (et donc, ici, de dimension 1).

L'algorithme d'apprentissage se déroule en deux étapes.

Activation et détermination du neurone vainqueur A l'instant t, on présente au réseau une entrée $x_t = (o_t, c_t)$ composée d'une observation o_t et d'un contexte c_t. On calcule la *similarité* entre chaque neurone i et cette entrée, similarité qui se décompose en :

$$sim_{in}(o_t, i) = \exp\left(-\frac{{||o_t - w_{in}(i)||_2}^2}{2\sigma_{in}^2}\right), \quad (1)$$

$$sim_{rec}(c_t, i) = \exp\left(-\frac{{||c_t - w_{rec}(i)||_2}^2}{2\sigma_{rec}^2}\right). \quad (2)$$

Ces deux similarités sont combinées en une similarité globale

$$sim_g(x_t, i) = \sqrt{sim_{in}(o_t, i) \times \beta(1-\beta) sim_{rec}(c_t, i)} \quad (3)$$

où β est un réel de $[0; 1]$.
La similarité globale est elle-même convoluée [2] avec une gaussienne pour la lisser

$$sim(x_t, i) = \frac{1}{N} \sum_{k=-N/2}^{k=N/2} sim_g(x_t, \overline{i+k}) \exp\left(-\frac{k^2}{2\sigma^2}\right) \quad (4)$$

avec σ un paramètre à choisir et où $\overline{i+k} = (i+k)(mod N)$, ce qui permet de déterminer le **neurone vainqueur** au temps t, dont l'indice est noté i_t^* :

$$i_t^* = \underset{i}{\operatorname{argmax}}\, sim(x_t, i). \quad (5)$$

On obtient aussi le **contexte** pour le pas de temps suivant en fonction de la position du neurone vainqueur.

$$c_{t+1} = pos(i_t^*) \quad (6)$$

Apprentissage des prototypes Le principe de l'apprentissage à l'instant t est de rapprocher les poids d'entrées et les poids récurents du couple (x_t, c_t). Chaque schéma d'apprentissage suit la même logique. On a donc pour tout neurone $j \in [0 \dots N)$:

$$w_{in}(j) \leftarrow w_{in}(j) + \epsilon \,.\, h(\nu_{in}, j, \underline{\mathrm{d}}(x_t, w_{in}(i_t^*)) \times \underline{\mathrm{d}}(x_t, w_{in}(j))\,(x_t - w_{in}(j)) \quad (7)$$

$$w_{rec}(j) \leftarrow w_{rec}(j) + \epsilon \,.\, h(\nu_{rec}, j, \underline{\mathrm{d}}(c_t, w_{rec}(i_t^*)) \times \underline{\mathrm{d}}(c_t, w_{rec}(j))\,(c_t - w_{rec}(j)) \quad (8)$$

avec

$$h(\nu, j, d) = \exp\left(-\frac{{||pos(i_t^*) - pos(j)||_2}^2}{\nu^2\, d^2}\right) \quad (9)$$

où ν, réél, est le paramètre d'élasticité de l'architecture.
Ici, on utilise des "distances normées", notées $\underline{\mathrm{d}}(.,.)$ qui sont toutes comprises entre 0 et 1.

$$\underline{\mathrm{d}}(x, y) = \frac{||x - y||_2}{\max_{x,y} ||x - y||_2} \quad (10)$$

Le noyau $h(\nu, j, d)$ permet de réguler la tendance des prototypes de chaque neurone j à se rapprocher de (x_t, c_t) en tenant compte de trois facteurs :

- plus le neurone vainqueur i_t^* est proche du prototype cible (x_t ou c_t), moins les autres neurones sont modifiés ;
- plus un neurone j est éloigné du vainqueur i_t^* (dans l'espaces des positions pos), moins il sera modifié ;
- enfin, plus le paramètre d'élasticité ν est petit, moins les neurones sont modifiés.

2. Pour cette convolution, nous considèrons que la similarité est un signal périodique, de période N.