Présentation de l'article :
Interactive Control of Diverse Complex Characters with Neural Network [4]

Kozolinsky Jules

8 février 2017

Table des matières

1 Introduction

1.1 Contexte

Présenté lors de la conférence d'intelligence artificielle NIPS 2015, cet article écrit par les chercheurs, Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic et Emanuel Todorov,

de l'université de Washington propose une méthode capable d'entrainer un réseau de neurones à se comporter comme une politique dotée d'un feedback quasi optimal.

1.2 Objectif de l'article

On souhaite que la politique issue du réseau de neurones entrainé s'applique à tout type d'entités et de morphologies, soit capable d'opérer en temps réel et soit insensible au bruit pouvant exister. Elle est appliquée ici à des tâches de déplacement (nage, vol, marche bipède et quadrupèdes pour différentes morphologies). La méthode utilisée s'appuie principalement sur l'optimisation de trajectoires avec du bruit et l'apprentissage supervisé d'un réseau de neurones.

2 Principe global

La méthode consiste à apprendre au réseau de neurones à générer des trajectoires similaires à celle de l'optimiseur hors ligne.

2.1 Définitions

Etat de l'entité On définit l'état d'une entité comme la matrice $(q\ f\ r)$ où q est l'état physique, (par exemple la position, l'orientation, l'angle des joints, ...), f sont les forces de contact avec le sol et r est la mémoire (non définie dans l'article, mais permettant sûrement de stocker des informations numériquement utile pendant la trajectoire).

Trajectoire La trajectoire de l'entité est la succession d'états pendant une période T :

$$X = (q^0\ f^0\ r^0\ ...\ q^T\ f^T\ r^T)$$

Politique issue du réseau de neurones
La politique est une fonction :

$$\pi_\theta : s \mapsto a$$

où
- θ est le vecteur des poids du réseau de neurones,
- $s^t(X) = (q^t\ r^t\ \dot{q}^{t-1}\ f^{t-1})$ est *l'état sensoriel* au temps t de la trajectoire X,
- $a^t(X) = (\dot{q}^t\ \dot{r}^t\ f^t)$ est *l'action optimale* au temps t de la même trajectoire X.

(On a noté $\dot{x}^t = x^{t+1} - x^t$, proche de la dérivée temporelle de x pour Δt petit).
Note : On peut remarquer en observant l'expression de l'action optimale que le réseau de neurones apprends non seulement le contrôle optimal mais aussi le modèle dynamique sous-jacent.

2.2 Description du problème

L'objectif se réduit au problème d'optimisation sous contrainte égalité suivant :

$$\min_{\theta, X^1, ..., X^N} \sum_{i=1}^{N} C_i(X^i) \qquad \text{sachant} \quad \forall i, t : a^t(X^i) = \pi_\theta(s^t(X^i)) \tag{1}$$

où
- θ est le vecteur des poids du réseau de neurones (qui sera utilisé pour exécuter la politique en temps réel après optimisation)
- X^i trajectoire d'objectif i (on prends différentes conditions initiales pour les différentes tâches)
- $C(X)$ est le coût total de la trajectoire X (explicité plus tard).

2.3 Trajectoire bruitée

On ne va pas calculer uniquement les trajectoires optimales mais aussi les mêmes trajectoires *entourées* de bruits, *i.e.* ajouter du bruit à l'état sensoriel, en entrée de la politique.

2.3.1 L'intérêt du bruit

D'après d'autres articles, ajouter du bruit lors de l'entrainement du réseau de neurones produits des mouvements plus robustes [3] [6]. La politique est en effet capable de se corriger et de tendre alors vers la trajectoire optimale (sans bruit). D'autre part, ajouter du bruit réduit le sur-apprentissage [1] et stabilise le comportement du réseau de neurones [2]. Ici on ajoute principalement du bruit pour éviter que la politique diverge lors de son exécution en temps réel.

2.3.2 L'ajout du bruit

On ajoute donc un bruit gaussien $\varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2 I)$ (où $\sigma_\varepsilon^2 \simeq 10^{-2}$) dans l'expression de la politique :

$$\pi_\theta(s + \varepsilon) = a + a_s \varepsilon$$

où a_s est le gradient lors du développement limité au premier ordre. Cette matrice représente le feedback optimal autour de l'état sensoriel s. Nous verrons dans la partie suivante pour son calcul explicite. (a sera quant à lui calculer par optimisation de la trajectoire X)

2.4 Algorithme

Reprenons le problème d'optimisation avec contraintes égalités en ajoutant ce bruit gaussien. Ce problème n'étant pas convexe, on remplace cette contrainte *dure* par une pénalité quadratique :

$$\min_{\theta, X^1, \dots, X^N} \sum_{i=1}^{N} C_i(X^i) + \sum_{i,t} R(s^t(X^i), a^t(X^i), \theta, \varepsilon^{i,t})$$
$$\text{avec} \quad R(s, a, \theta, \epsilon) = \frac{\alpha}{2} ||(a + a_s \varepsilon) - \pi_\theta(s + \epsilon)||^2$$

où α est la pénalité quadratique. ($\alpha \simeq 10.$)

Cela mène à l'algorithme suivant :

Algorithm 1

1. Échantillonner le bruit $\varepsilon^{i,t}$
2. Optimiser N trajectoires indépendamment :

$$\forall i, \ \bar{X}^i = \text{argmin}_X \ C_i(X) + \sum_t R(s^t(X), a^t(X), \bar{\theta}, \bar{\varepsilon}^{i,t}) + \frac{\eta}{2} ||X - \bar{X}^i||^2$$

3. Entrainer le réseau de neurones (régression) :

$$\bar{\theta} = \text{argmin}_\theta \ \sum_{i,t} R(s^t(\bar{X}_i), a^t(\bar{X}_i), \theta, \bar{\varepsilon}^{i,t}) + \frac{\eta}{2} ||\theta - \bar{\theta}||^2$$

4. Répéter

Note : On a ajouté un terme de régularisation de paramètre $\eta \simeq 10^{-2}$.

3 Optimisation de trajectoire

Détaillons dans cette section la deuxième étape de l'algorithme. On souhaite trouver une trajectoire optimale débutant d'une certaine condition initiale et réalisant une tâche tout en vérifiant la réalité physique, *i.e.* les équations de la dynamique. (La partie d'optimisation de trajectoire simple (sans bruit) est basé sur *Discovery of Complex Behaviors through Contact-Invariant Optimization* [5] impliquant une partie des chercheurs travaillant sur cet article.)

3.1 Vérifier les équations de la dynamique

Les paramètres doivent vérifier les équations de la dynamique suivantes :

$$H(q)\ddot{q} + \hat{C}(q,\dot{q}) = \tau + J(q,\dot{q})^\top f, \qquad d(q) \geq 0, \quad d(q)^\top f = 0, \quad f \in K(q)$$

où

- f forces de contacts sur les organes terminaux
- τ inverse de la dynamique
- H matrice d'inertie
- $\hat{C}$ matrice de Coriolis et des forces centrifuges
- J matrice Jacobienne
- $d(q)$ distance au sol
- $K(q)$ cône de friction

Ces contraintes sont ajoutées avec un terme de pénalité dans le coût total $C(X)$. (On fait de même avec les conditions initiales.)

3.2 Calcul de la trajectoire optimale

3.2.1 Définition du coût total

Définissons le coût total de la manière suivante :

$$C(X) = \sum_t c^t(\phi^t(X))$$

où

- $\phi^t(X)$ extrait les caractéristiques nécessaires pour calculer le coût de la trajectoire X
- $c(\phi)$ coût à partir de ces caractéristiques de la trajectoire

Pour simplifier, on écrit le problème d'optimisation précédant sous la forme suivante (on mets tous les objectifs *à l'intérieur* de C) :

$$X^* = \text{argmin}_X C(X)$$

3.2.2 Méthode de Quasi-Newton

On utilise la méthode de quasi-Newton pour résoudre ce problème d'optimisation. D^2_X est ainsi calculé par approximation de Gauss-Newton. D'autre part, c est choisi tel que son gradient et sa Hessienne soit calculable analytiquement, alors que ϕ est calculé par différence finie. Ainsi la trajectoire optimale vérifie l'équation suivante :

$$X^* = X^* - D^2_X C(X)^{-1} \nabla_X C(X)$$

Note : On ne lance pas cet algorithme jusqu'à convergence de X^* mais on ne fait que quelques itérations (entre 1 et 10).

3.3 Calcul de la trajectoire optimale bruitée

On souhaite calculer la trajectoire optimale bruitée $\tilde{X}$, perturbation de la trajectoire optimale X telle que $s(\tilde{X}) = \bar{s} = s + \varepsilon$.
Le bruit intervient dans l'expression de la politique développée au premier ordre :

$$\pi_\theta(s(\tilde{X})) = \pi_\theta(s + \varepsilon) = a + a_s\varepsilon = a(\tilde{X})$$

.

3.3.1 Calcul de la matrice de feedback optimale a_s

L'action optimale a a été calculée lors du calcul de la trajectoire optimale X. Il suffit donc de calculer la matrice de feedback optimal a_s. Par une règle de gradient composé, on obtient :

$$a_{\bar{s}} = a_X \tilde{X}_{\bar{s}}$$

Or le gradient de a par rapport à X a déjà été calculé lors du calcul de ϕ_X, car les fonctions a et ϕ sont des fonctions qui extraient les caractéristiques de la trajectoire X. (Il en va de même pour s_X.)

3.3.2 Calcul du gradient $\tilde{X}_{\bar{s}}$

La trajectoire optimale bruitée est solution du problème d'optimisation suivant :

$$\tilde{X}(\bar{s}) = \text{argmin}_{X^*}\big(C(X^*) + \frac{\lambda}{2}||s(X^*) - \bar{s}||^2\big)$$

Ainsi, par approximation à partir de la trajectoire optimale non bruitée et de la méthode de Quasi-Newton (où $\lambda \simeq 10^2$),

$$\tilde{X}(\bar{s}) = X - (C_{XX} + \lambda s_X^\top s_X)^{-1}(C_X + \lambda s_X^\top (s(X) - \bar{s})$$

Dont le gradient est :

$$\tilde{X}_{\bar{s}} = \lambda(C_{XX} + \lambda s_X^\top s_X)^{-1} s_X^\top$$

dont on a déjà calculé l'ensemble des termes.

4 Entrainement du réseau de neurones

4.1 Utilisation du réseau de neurones

Passons à l'étape suivante de l'algorithme. On souhaite entrainer le réseau de neurones, *i.e.* effectuer une régression. En effet, on a les données d'entrainement $(s + \varepsilon, a + a_s\varepsilon)^{i,t}$ (calculées lors de l'étape précédentes) et on souhaite calculer θ le vecteur des poids du réseau de neurones apparaissant dans la politique $\pi_\theta : s \mapsto a$.

L'article ne détaille que très peu le fonctionnement des réseaux de neurones, utilisés comme une boîte noire d'optimisation, là où nous aurions pu utiliser d'autres méthodes d'apprentissage supervisé pour les régressions.

4.2 Paramètres du réseau de neurones

Le réseau de neurones est donc constitué, en plus des couches d'entrée et de sortie, de trois couches cachées, chacune possédant 250 neurones. La fonction d'activation est une fonction classique des réseaux de neurones : tanh. Ces paramètres ont été choisis grâce au compromis complexité/erreur suivant :

10 neurons	0.337 ± 0.06
25 neurons	0.309 ± 0.06
100 neurons	0.186 ± 0.02
250 neurons	0.153 ± 0.02
500 neurons	0.148 ± 0.02

(a) Increasing Neurons per layer with 4 layers

1 layer	0.307 ± 0.06
2 layers	0.253 ± 0.06
3 layers	0.153 ± 0.02
4 layers	0.158 ± 0.02

(b) Increasing Layers with 250 neurons per layer

Table 1: Mean and variance of joint position error on test rollouts with our method after training with different neural network configurations.

5 Exécution de la politique

A présent, l'algorithme hors-ligne a été détaillé et lorsqu'il termine, nous obtenons un réseau de neurones entrainé agissant comme une politique $\pi_{\theta^*} : s \mapsto a$.
En robotique, on appliquerait cette politique sur le robot et cela déterminerait les forces à appliquer à chacun des moteurs.
En animation, on souhaite calculer la position suivante de l'entité. Pour cela un simple problème d'optimisation à horizon $T = 1$ suffit.

6 Résultats

Pour les expériences, un pas de temps de $\Delta t = 50$ ms a été utilisé. Les résultats des expériences sont présentés dans la vidéo suivante. Il a été demandé à des entités de morphologies différentes de nager, voler et de marcher.
Video : `https://www.youtube.com/watch?v=IxrnT0JOs4o` Le comportement cyclique lors du vol est assez impressionnant car il n'a été codé nulle part. Celui-ci découle simplement de l'optimisation faite lors de l'algorithme. Il en va de même pour la marche où les pas droite-gauche n'ont pas été directement implémentés dans l'interface. L'entité lève les pieds, car conformément à l'article *Discovery of Complex Behaviors through Contact-Invariant Optimization* [5], le glissement n'est pas autorisé.

7 Conclusion

Finalement cette algorithme comportant une phase hors-ligne d'optimisation et d'apprentissage supervisé, pour obtenir un réseau de neurones entrainé se comportant comme une politique, semble très efficace, d'autant plus qu'il se déploie en temps réel. Cet article va dans le sens où on souhaite trouver des politiques en temps réel pour les appliquer sur les robots.

Dans l'article, seulement des tâches de déplacement ont été implémentés. On pourrait s'intéresser à d'autres tâches moins triviales tels escalader, se relever ou encore ouvrir une porte.
D'autre part, on pourrait utiliser d'autres algorithmes d'apprentissage que les réseaux de neurones.

Références

[1] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv :1207.0580*, 2012.

[2] Gregor M Hoerzer, Robert Legenstein, and Wolfgang Maass. Emergence of complex computational structures from chaotic neural networks through reward-modulated hebbian learning. *Cerebral cortex*, 24(3) :677–690, 2014.

[3] Dongsung Huh and Emanuel Todorov. Real-time motor control using recurrent neural networks. In *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL'09. IEEE Symposium on*, pages 42–49. IEEE, 2009.

[4] Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic, and Emanuel V. Todorov. Interactive control of diverse complex characters with neural networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3132–3140. Curran Associates, Inc., 2015.

[5] Igor Mordatch, Emanuel Todorov, and Zoran Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics (TOG)*, 31(4) :43, 2012.

[6] Jack M Wang, David J Fleet, and Aaron Hertzmann. Optimizing walking controllers for uncertain inputs and environments. In *ACM Transactions on Graphics (TOG)*, volume 29, page 73. ACM, 2010.