Proceedings of the 8th International Conference on Applied Informatics
Eger, Hungary, January 27–30, 2010. Vol. 1. pp. 87–94.

Using Gaussian Processes for Variance Reduction in Policy Gradient Algorithms*

Hunor Jakab, Lehel Csató

Babes Bolyai University

Abstract

Gradient based policy optimization algorithms suffer from high gradient variance, this is usually the result of using Monte Carlo estimates of the Q-value function in the gradient calculation. By replacing this estimate with a function approximator on state-action space, the gradient variance can be reduced significantly. In this paper we present a method for the training of a Gaussian Process to approximate the action-value function which can be used to replace the Monte Carlo estimation in the policy gradient evaluation. An iterative formulation of the algorithm will be given for better suitability with online learning.

1. Introduction

Gradient based policy optimization algorithms, like REINFORCE [17], Vanilla Policy Gradients, Natural Actor-Critic [11] have many advantages over traditional value-function based methods when it comes to learning control policies of complex systems, however the majority of these methods suffer from high gradient variance, a result of using Monte Carlo estimates of the Q-value function in the calculation of the gradient $Q^\pi(x,a) \sim \left(\sum_{j=o}^{H} \gamma^j r_j\right)$. By replacing this estimate with a function approximation of the value-function on state-action space, the gradient variance can be reduced significantly[15]. We present the training of a Gaussian Process for the approximation of the action-value function $Q(\cdot,\cdot) \sim GP(\mu_q, k_q)$ which can be used to replace the Monte Carlo estimation in the policy gradient evaluation, thereby reducing the gradient variance and quickening learning.

In section 2 we will give a short description of the Reinforcement Learning (RL) problem and basic notation. In section 3 the idea of Policy Gradient methods will be shortly described together with the use of value-function approximators. In

*The authors acknowledge for the financial support from The Sectoral Operational Programme Human Resources Development, POSDRU 6/1.5/S/3 - "Doctoral studies: through science towards society" and PNTCD II 11-039/2007.

section 4 we will describe our approach of using GP regression for approximating state-action value functions. The learning of control policies has to be performed online, since the only available data from which we can draw conclusions is provided by the agent's interaction with its environment. Therefore we will use an iterative variant of Gaussian Process regression as described in [1, 10]. Section 5 will discuss the application of the GP estimator in conjunction with Policy Gradient estimation. At the gradient estimation stage we will choose between using the full Monte Carlo estimates, the predicted Value obtained from the GP predictive mean or a combination of these. We close the paper with a discussion in section 6.

2. Problem Setting

The (RL) problem can be modelled with a Markov Decision Process (MDP): an MDP [9, 12] is a tuple $M\,(S, A, P, R)$ where S is the set of states; A the set of actions; $P : S \times A \times S \to [0, 1]$, $P(s, a, s')$ is the conditional probability of a transition from state s to s' when executing an action a; $R : S\times \to \mathbb{R}$, $a \in A$, $R(s, a)$ denoting the immediate reward r when in state s. MDPs provide the framework to solve the RL problem: the goal being the search for a policy π that maximizes the expected cumulative discounted reward.

$$J_\pi = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_{s_t, a_t} \right] \tag{2.1}$$

where $E[\cdot]$ is the expected value for the policy π, $0 < \gamma < 1$ is a discount rate. The policy is a probability distribution of actions a over state s, denoted $\pi_\theta(s, a)$ and valid for all states $s \in S$. It is parameterized by the parameter vector θ the elements of which will be modified during the learning process. The expression for the cumulative reward on the right hand side of (2.1) is analytically tractable and we can search for a policy that maximizes it in different ways.

3. Policy Gradients

Policy gradient (PG) algorithms optimize the *parameters* $\boldsymbol{\theta}$ of a parametric policy, where the optimization is being done with respect to the expected reward $J(\boldsymbol{\theta})$. We thus need a good approximation to the policy gradient, as in equation (3.1) with respect to the episodic reward $R(\tau) = \sum_{t=0}^{H-1} \gamma^t R_{a_t}(s_t)$, and the update is done based on the expression in (3.2) [11].

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \int \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\tau) R(\tau) d_\tau \tag{3.1}$$

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \alpha_i \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{3.2}$$

Here α is a learning rate, i is the current update step, and τ stands for a history of states and controller outputs for an episode of length H.

$$\tau = \{(s_0, a_0), (s_1, a_1), \dots (s_H, a_H)\} \tag{3.3}$$

$p_{\boldsymbol{\theta}}(\tau)$ from (3.4) is the state-distribution corresponding to an episode starting from an arbitrary state.[1]

$$p_{\boldsymbol{\theta}}(\tau) = p(s_0) \prod_{t=0}^{H-1} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{3.4}$$

Likelihood ratio methods [14] provide a good way of expressing the gradient in a form that doesn't depend on the actual state-transition probabilities and so can be easily approximated. Using the episodic state distribution from (3.4) we can calculate the gradient from (3.1) by using the likelihood ratio trick, and following the derivation from [16].

The gradient is expressed as:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = E\left[\sum_{t=0}^{H-1} \nabla_{\boldsymbol{\theta}} \log \pi(a_t|s_t) R(\tau)\right] \tag{3.5}$$

The advantage in expressing the gradient as above lies in the fact that it can be approximated via averaging over a number of controller output histories [17]. The term $R(\tau)$ is in fact the Monte Carlo estimation of $Q(s_t, a_t)$ the value function over state-action space. Although $R(\tau)$ is an unbiased estimator of the true Q-value function, it has high variance however it can be replaced with a function approximator. In case the function approximator and policy parameterization fulfill the compatibility conditions from [15] the approximated function can be viewed as the approximation of the advantage function, which is the difference between the state-action and state value functions. In this paper we use Non-parametric Gaussian Processes[2] to approximate the Q-value function: $Q(\cdot, \cdot) \sim GP(\mu_q, k_q)$

4. Online GPR for Value-function approximation

Different approaches of using Gaussian Processes for the Value-function approximation problem can already be found in the literature. In [8, 6, 7, 5] a generative model is being used to express the value function in a one-step TD-like formulation and perform regression based on this model on the individual returns. In [4] the transition dynamics of the MDP are also estimated and so the Value function can be completely reevaluated in a Dynamic Programming manner at each episode. We choose a simpler method of performing Gaussian Process Regression using the

[1] This way of expressing the state distribution is only possible if the environment is Markovian. Otherwise the joint probability could not be written as the product of the individual probabilities

[2] The compatibility conditions can be fulfilled by using a kernel composed from a regular state-action kernel and a fisher kernel, and parameterizing the policy accordingly [7]

Monte Carlo returns as noisy targets and the visited state-action pairs as support points. At the gradient estimation step we combine the Q-values predicted by our trained GP and the actual Monte Carlo returns to obtain a hybrid learning algorithm. Let us consider an episode τ consisting of $(s_t, a_t)_{t=\overline{1,H}}$ state-action pairs. For each of these state-action pairs we can calculate the Monte Carlo estimation of the Q-value function based on the sum of discounted rewards until the end of the episode:

$$\hat{Q}(s_t, a_t) = \sum_{i=0}^{H-t} \left(\gamma^i R(s_{t+i}, a_{t+i})\right) \tag{4.1}$$

Let us denote this value for the state-action pair at time t with Q_t. We consider these values as noisy observations of the true state-action value function which in our case is the latent function $f(.,.)$ modelled by the Gaussian Process $GP(\mu_q, k_q)$.

$$Q_t = f(s_t, a_t) + \epsilon_s \quad \epsilon_s \sim N(0, \sigma^2)$$

The regression is performed directly in function space with prior mean and covariance:

$$E\left[f(s,a)\right] = \mu_q(s,a) = 0$$
$$Cov(f(s,a), f(s',a')) = k_q\left((s,a),(s',a')\right)$$

Here $k_q(.,.)$ is the kernel function which operates on state-action pairs, and gives the element of the kernel matrix K_q. Since the arguments of the kernel function are state-action pairs it makes sense to construct it from the composition of two kernel functions which appropriately capture the covariance properties of states and actions respectively as suggested in [6]. Suppose we have a set D of state-action pairs and the corresponding noisy measurements which constitute our support points: $D = \{(s_t, a_t)\}$,$\hat{\boldsymbol{Q}} = \{\hat{Q}_t\}; t = \overline{1,n}$; These points have a joint Gaussian distribution with mean 0 and covariance matrix $K_q^n + \Sigma_n$; Given a new data point $x_{n+1} = (s_{n+1}, a_{n+1})$ and denoting $\boldsymbol{k}_{n+1} = [k_q(x_1, x_{n+1}), \ldots, k_q(x_n, x_{n+1})]$, the joint distribution of the prediction and the target values (4.2), as well as the predictive mean (4.3) and variance (4.4) for the new data-point (s_{n+1}, a_{n+1}) conditioned on the support points can be exactly calculated [13]:

$$\begin{bmatrix} \hat{\boldsymbol{Q}} \\ f_{n+1} \end{bmatrix} \sim N\left(0, \begin{bmatrix} K_q^n + \Sigma_n & \boldsymbol{k}_{n+1} \\ \boldsymbol{k}_{n+1}^T & k_q\left(x_{n+1},\right)) \end{bmatrix}\right) \tag{4.2}$$

$$f_{n+1} | \hat{\boldsymbol{Q}}, D \sim N\left(\mu_{n+1}, cov(f_{n+1})\right) \tag{4.3}$$
$$E[f_{n+1}] = \mu_{n+1} = \boldsymbol{k}_{n+1}\boldsymbol{\alpha}_n \tag{4.4}$$
$$cov(f_{n+1}, f_{n+1}) = k_q\left(x_{n+1}, x_{n+1}\right) - \boldsymbol{k}_{n+1}\boldsymbol{C}_n\boldsymbol{k}_{n+1}^T \tag{4.5}$$

$\boldsymbol{\alpha}_n$ and $\boldsymbol{c}_n$ are the parameters of the GP and have the following form:

$$\boldsymbol{\alpha}_n = [K_q^n + \Sigma_n]^{-1}\hat{\boldsymbol{Q}} \qquad \boldsymbol{C}_{n+1} = [K_q^n + \Sigma_n]^{-1} \tag{4.6}$$

The inversion of the $n \times n$ kernel matrix is computationally demanding, moreover the size of the matrix increases quadratically with the number of data-points. Because the Policy Gradient algorithm uses data acquired through interaction with the process environment we cannot batch-process. Therefore a suitable online version of the GP regression must be used to calculate the inverse matrix in an iterative way each time a new data-point is added to the set of support points D.

4.1. Online learning

We can update the parameters $\boldsymbol{\alpha}$ & $\boldsymbol{C}$ of the mean and covariance functions of the Gaussian process iteratively each time a new point $\{(s_{n+1}, a_{n+1}), \hat{Q}_{n+1}\}$ is added to the set of Support Points, by combining the likelihood of the new data-point and the Gaussian Prior from the previous step. This method is described in [3] for the generalized linear models with arbitrary likelihood functions.

$$\boldsymbol{\alpha}_{n+1} = \boldsymbol{\alpha}_n + q^{n+1}\boldsymbol{s}_{n+1} \tag{4.7}$$

$$\boldsymbol{C}_{n+1} = \boldsymbol{C}_n + r^{n+1}\boldsymbol{s}_{n+1}\boldsymbol{s}_{n+1}^T \tag{4.8}$$

$$\boldsymbol{s}_{n+1} = \boldsymbol{C}_n\boldsymbol{k}_{n+1}^T + \boldsymbol{e}_{n+1} \tag{4.9}$$

$$\boldsymbol{e}_{n+1} = [0, 0, \ldots, 1]^T_{\mathcal{M}_{1\times(n+1)}} \tag{4.10}$$

Fortunately in the case of regression with Gaussian noise the resulting posterior is analytically tractable, and q^{n+1}, r^{n+1} can be given exactly. For this first we need the marginal predictive distribution for the new data-point which is a Gaussian with mean μ_{n+1} given by (4.4) and variance σ^2_{n+1} given by (4.5). Than by calculating the first and second derivatives of the average likelihood with respect to the mean and setting them to 0 we get:

$$q^{n+1} = \frac{\hat{Q}_{n+1} - \mu_{n+1}}{\sigma(s_{n+1})^2 + \sigma^2_{n+1}} \quad \text{and} \quad r^{n+1} = -\frac{1}{\sigma(s_{n+1})^2 + \sigma^2_{n+1}} \tag{4.11}$$

The hyper-parameters of the Gaussian Process can be optimized with the help of evidence maximization [13].

5. Gradient Estimation

Using our GP function approximator we can now replace the Monte Carlo estimation of the Q-values in the policy gradient formula (3.5). We can choose to use solely the GP predicted mean instead of the term $R(\tau)$ or we can combine an arbitrary number immediate returns from the episode with the GP prediction which we will call m-step q-value.

$$Q^m(s_t, a_t) = \sum_{i=0}^{m-1} \gamma^i R(s_{t+i}, a_{t+i}) + \gamma^m E[f(s_{t+m}, a_{t+m})] \tag{5.1}$$

Where the $E\{f(s_{t+m}, a_{t+m})\}$ is given by equation (4.4). If m is grater than the number of time-steps in the respective episode than the full Monte Carlo estimate from (4.1) is used. By choosing $m = 0$ or $m = H$ we get the pure GP prediction-based version or the REINFORCE version of the policy gradient algorithm respectively. The formula for the gradient estimate then becomes :

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = E\left[\sum_{t=0}^{H-1} \nabla_{\boldsymbol{\theta}} \log \pi(a_t|s_t) Q^m(s_t, a_t)\right] \tag{5.2}$$

In Algorithm 1 we can see a description of the above discussed method in an algorithmic form.

Algorithm 1 Policy Gradient algorithm with GP function approximator

Initialize policy parameterization $\pi(s, a|\boldsymbol{\theta})$
Initialize GP parameters $\boldsymbol{\alpha} = 0, \boldsymbol{C} = 0, \mathcal{D} = 0, n = 0, m = const$
repeat ▷
  Sample data $\{(s_t, a_t, R(s_t, a_t)\}_{t=\overline{1,H}}$ from an episode τ
  for each time-step $t = \overline{1, H}$ **do**
    calculate $\hat{Q}_t$ (4.1)
    calculate $q^{n+1}; r^{n+1}$ (4.11)
    increment n, add $\{(s_t, a_t), \hat{Q}_t\}$ to the set of support points
    update GP parameters $\boldsymbol{\alpha}; \boldsymbol{C}$ (4.7)
  end for
  Estimate gradient $\nabla \boldsymbol{J}_{\boldsymbol{\theta}}$ (5.2)
until gradient has converged
Perform policy update (3.2)

6. Discussion

We have described a way of approximating state-action value functions using a Gaussian Process as a function approximator to reduce the variance in Policy Gradient algorithms. Unlike in [8] where a generative model based on the idea of TD is used to approximate the value function, we performed regression on the Monte Carlo returns without bootstrapping. This makes the approximation independent from the Markov assumption. Although the formulation of the episodic state-distribution (3.4) in the gradient estimation assumes that the environment is markovian, in our opinion the algorithm can still benefit from the nature of the state-action value function approximation.

One major obstacle in efficiently using the described algorithm in RL problems is the computational complexity of the GP prediction which increases exponentially with the number of training points. This could be avoided by extending the algorithm with a sparsification mechanism [2]. The procedure could be further

enhanced by using the predictive variance of the GP to influence the search directions of the Policy Gradient algorithm. These could be the major steps of future research.

References

[1] L. Csató. *Gaussian Processes – Iterative Sparse Approximation*. PhD thesis, Neural Computing Research Group, March 2002. URL `www.ncrg.aston.ac.uk/Papers`.

[2] L. Csató and M. Opper. Sparse representation for Gaussian process models. In *NIPS*, volume 13, pages 444–450. The MIT Press, 2001.

[3] L. Csató, E. Fokoué, M. Opper, B. Schottky, and O. Winther. Efficient approaches to Gaussian process classification. In *NIPS*, volume 12, pages 251–257. The MIT Press, 2000.

[4] M. P. Deisenroth, C. E. Rasmussen, and J. Peters. Gaussian process dynamic programming. *Neurocomputing*, 72(7-9):1508–1524, 2009. ISSN 0925-2312.

[5] Y. Engel, S. Mannor, and R. Meir. Bayes meets Bellman: The Gaussian process approach to temporal difference learning. In *Proc. of the 20th International Conference on Machine Learning*, pages 154–161, 2003.

[6] Y. Engel, S. Mannor, and R. Meir. Reinforcement learning with Gaussian processes. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 201–208, New York, NY, USA, 2005. ACM.

[7] M. Ghavamzadeh and Y. Engel. Bayesian actor-critic algorithms. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 297–304, New York, NY, USA, 2007a. ACM. ISBN 978-1-59593-793-3.

[8] M. Ghavamzadeh and Y. Engel. Bayesian policy gradient algorithms. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *NIPS '07: Advances in Neural Information Processing Systems 19*, pages 457–464, Cambridge, MA, 2007b. MIT Press.

[9] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[10] M. Opper. Online versus offline learning from random examples: General results. *Phys. Rev. Lett.*, 77(22):4671–4674, 1996.

[11] J. Peters and S. Schaal. Policy gradient methods for robotics. In *IROS*, pages 2219–2225. IEEE, 2006.

[12] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.

[13] C. E. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[14] T. Rückstieß, M. Felder, and J. Schmidhuber. State-dependent exploration for policy gradient methods. In *ECML/PKDD*, pages 234–249, 2008.

[15] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *NIPS*, pages 1057–1063, 1999.

[16] D. Wierstra, A. Foerster, J. Peters, and J. Schmidhuber. Solving deep memory pomdps with recurrent policy gradients. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, 2007.

[17] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Hunor Jakab, Lehel Csató
Mihail Kogalniceanu str. 1
Cluj Napoca
Romania