Reinforcement Learning
Intelligent Systems Series
Lecture 11

Georg Martius

MPI for Intelligent Systems, Tübingen, Germany

January 11, 2019

Generic optimization methods applied to RL

Today we go through several optimization methods, that are not specific to RL, but quite powerful.
The lecture is at the blackboard. The papers we discuss are linked below.

Zero-order random search for RL problems

Random search and variations

- old algorithm (at least 1965: J. Matyas. Random optimization. Automation and Remote control, 26(2):246253, 1965)
- recently compared with slight modifications to modern RL algos: *Simple random search provides a competitive approach to reinforcement learning* by Horia Mania, Aurelia Guy, Benjamin Recht
  `https://arxiv.org/abs/1803.07055`

Skandal: Random search of linear policies outperforms Deep Reinforcement Learning

Larger is better		Maximum average reward		
Task	**RS**	**NG-lin**	**NG-rbf**	**TRPO-nn**
Swimmer-v1	365	366	365	131
Hopper-v1	3909	3651	3810	3668
HalfCheetah-v1	6722	4149	6620	4800
Walker	11389	5234	5867	5594
Ant	5146	4607	4816	5007
Humanoid	11600	6440	6849	6482

Ben Recht, ICML Tutorial, 2018

Bottom line:

- Simple search can be surprisingly effective
- Policies are simple (linear policies) (few hundred parameters). Standard MuJoCo tasks are maybe to simple and have too little stochasticity. You could not solve Go with it.

PGPE

Parameter Exploring Policy Gradient

Parameter-exploring Policy Gradients by Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, Jürgen Schmidhuber
`http://mediatum.ub.tum.de/doc/1099128/631352.pdf`
Method: PGPE: Policy Gradient with Parameter Exploration

PGPE

Parameter distribution:

$$\Theta_i \sim \mathcal{N}(\mu_i, \sigma_i)$$

Update:

$$\Delta\mu_i = \alpha\frac{1}{2}\epsilon_i(r^+ - r^-)$$

$$\Delta\sigma_i = \alpha\left(\frac{(r^+ - r^-)}{2} - b\right)\frac{(\epsilon_i^2 - \sigma_i^2)}{\sigma_i}$$

With maximum reward adaptation m:

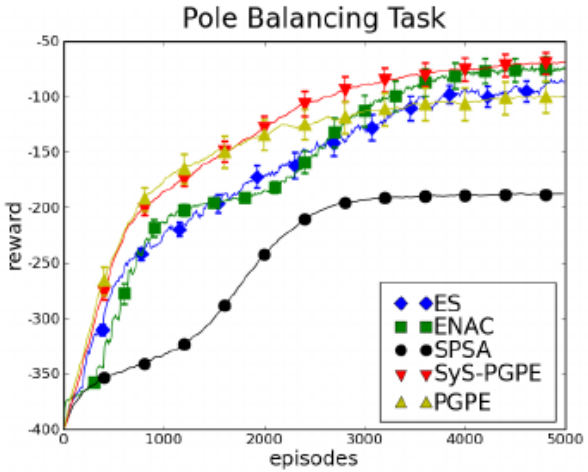
$$\Delta\mu_i = \frac{\alpha\epsilon_i(r^+ - r^-)}{2m - r^+ - r^-}, \qquad \Delta\sigma_i = \frac{\alpha}{m - b}\left(\frac{r^+ + r^-}{2} - b\right)\left(\frac{\epsilon^2 - \sigma_i^2}{\sigma_i}\right).$$

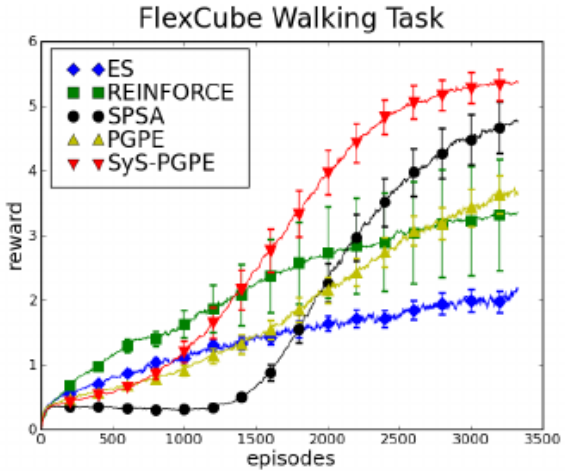
PGPE

Algorithm 1 The PGPE Algorithm without reward normalization: Left side shows the basic version, right side shows the version with symmetric sampling. $\boldsymbol{T}$ and $\boldsymbol{S}$ are $P \times N$ matrices with P the number of parameters and N the number of histories. The baseline is updated accordingly after each step. α is the learning rate or step size.

Initialize $\boldsymbol{\mu}$ to $\boldsymbol{\mu}_{\text{init}}$	Initialize $\boldsymbol{\mu}$ to $\boldsymbol{\mu}_{\text{init}}$
Initialize $\boldsymbol{\sigma}$ to $\boldsymbol{\sigma}_{\text{init}}$	Initialize $\boldsymbol{\sigma}$ to $\boldsymbol{\sigma}_{\text{init}}$
while TRUE **do**	**while** TRUE **do**
for $n = 1$ to N **do**	**for** $n = 1$ to N **do**
draw $\boldsymbol{\theta}^n \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{I\sigma}^2)$	draw perturbation $\boldsymbol{\epsilon}^n \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I\sigma}^2)$ $\boldsymbol{\theta}^{+,n} = \boldsymbol{\mu} + \boldsymbol{\epsilon}^n$ $\boldsymbol{\theta}^{-,n} = \boldsymbol{\mu} - \boldsymbol{\epsilon}^n$
evaluate $r^n = r(h(\boldsymbol{\theta}^n))$	evaluate $r^{+,n} = r(h(\boldsymbol{\theta}^{+,n}))$ evaluate $r^{-,n} = r(h(\boldsymbol{\theta}^{-,n}))$
end for	**end for**
$\boldsymbol{T} = [\, t_{ij} \,]_{ij}$ with $t_{ij} := (\theta_i^j - \mu_i)$	$\boldsymbol{T} = [\, t_{ij} \,]_{ij}$ with $t_{ij} := \epsilon_i^j$
$\boldsymbol{S} = [\, s_{ij} \,]_{ij}$ with $s_{ij} := \frac{t_{ij}^2 - \sigma_i^2}{\sigma_i}$	$\boldsymbol{S} = [\, s_{ij} \,]_{ij}$ with $s_{ij} := \frac{(\epsilon_i^j)^2 - \sigma_i^2}{\sigma_i}$
$\boldsymbol{r} = [(r^1 - b), \ldots, (r^N - b)]^{\mathrm{T}}$	$\boldsymbol{r}_T = [(r^{+,1} - r^{-,1}), \ldots, (r^{+,N} - r^{-,N})]^{\mathrm{T}}$ $\boldsymbol{r}_S = [(\frac{r^{+,1} + r^{-,1}}{2} - b), \ldots, (\frac{r^{+,N} + r^{-,N}}{2} - b)]^{\mathrm{T}}$
update $\boldsymbol{\mu} = \boldsymbol{\mu} + \alpha \boldsymbol{T}\boldsymbol{r}$	update $\boldsymbol{\mu} = \boldsymbol{\mu} + \alpha \boldsymbol{T}\boldsymbol{r}_{\boldsymbol{T}}$
update $\boldsymbol{\sigma} = \boldsymbol{\sigma} + \alpha \boldsymbol{S}\boldsymbol{r}$	update $\boldsymbol{\sigma} = \boldsymbol{\sigma} + \alpha \boldsymbol{S}\boldsymbol{r}_{\boldsymbol{S}}$
update baseline b accordingly	update baseline b accordingly
end while	**end while**

PGPE





- key difference Random Search: adaptation of sampling distribution (Variance)
- Policies are simple (linear policies) (few hundred parameters). Standard MuJoCo tasks are maybe to simple and have too little stochasticity. You could not solve Go with it.

CEM

Further reading/methods:

Cross Entropy Method (CEM)

Classical algorithm for rare-event simulation and for optimization. (PGPE is related)
Same key idea: adapt proposal distribution in parameter space
use best samples (elites) to adapt parameters of distribution
Papers:

- *Tutorial on the cross entropy method*
  `https://people.smp.uq.edu.au/DirkKroese/ps/aortut.pdf`
- *Path Integral Policy Improvement with Covariance Matrix* by Freek Stulp and Olivier Sigaud
  `https://icml.cc/2012/papers/171.pdf` that gives a very compact intro to CEM and CMA

CMA-ES

Further reading/methods:

Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

Probably the most commonly used black-box optimization method.
Slides: `http://www.cmap.polytechnique.fr/~nikolaus.hansen/intro-randomized-opt-tutorial2016-key13.pdf`
Tutorial paper on CMA-ES: *The CMA Evolution Strategy: A Tutorial* by Nikolaus Hansen
`https://arxiv.org/abs/1604.00772`

Conceptually the same as CEM and PGPE, but uses more tweaks for stepsize adaptation and estimating and using tthe covariance matrix