LEARNING/OPTIMIZATION ALGORITHMS FOR CONTINUOUS CONTROL TASKS IN DEEP RL

Amit Mate

Deep Reinforcement Learning Project Report

CCE –May-Dec 2019

PROJECT FOCUS

- Evaluate state of the art reinforcement learning methods vs traditional optimization methods for learning policy parameters on continuous action space tasks e.g.,
  - BipedalWalker-v2 (Easy)
  - BipedalWalkerHardcore-v2 (Hard)

- Two algorithms were narrowed down after extensive literature search in each category
  - Reinforcement Learning – PPO (policy gradient and trust region based), Augmented Random Search (automatic gradient based random/evolutionary search)
  - Traditional Optimization – COBYLA (non-derivative trust region based local optimization) and ISRES (global random/evolutionary search)

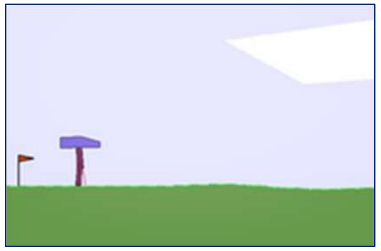
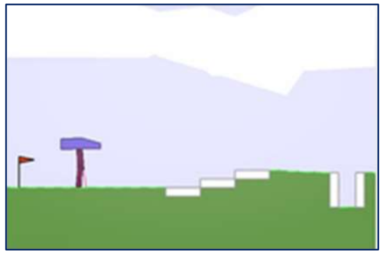
- The algorithms were evaluated on both Bipedal tasks based on following metrics
  - Maximum reward reached in 1000 iterations*
  - Number of iterations taken to reach reward of "200"

  (Optimal hyper-parameters were used from literature wherever possible, extensive hyper-parameter search was not possible with available compute and time)

*Study focus is very narrow due to time/compute limitations. Most research quote results after 10e6 iterations

ENVIRONMENT AND OPTIMIZATION/LEARNING ALGORITHMS

- Algorithm Choice
  - From reinforcement learning and traditional optimization resp.
    - One trust region based method that is good at finding local optimas (**PPO and COBYLA)**
    - One randomized search method each that is good at finding global optimas (**ARS and ISRES)**
- Environment choice
  - Continuous control robotic tasks are challenging and have real world applications
  - **BipedalWalker** is not too easy (e.g., like Cartpole) and is available in openAI gym with the open-source pyBullet physics engine – other Mujoco tasks require Mujoco engine which is licensed and setup is bulkier
  - Two variants – normal terrain and hard terrain (more uneven and stochastic environment)
  - Mission is to get a 2D biped walker to walk through rough terrain in the simulated environment

BIPEDALWALKER-V2 – INPUTS AND OUTPUTS

State Vector

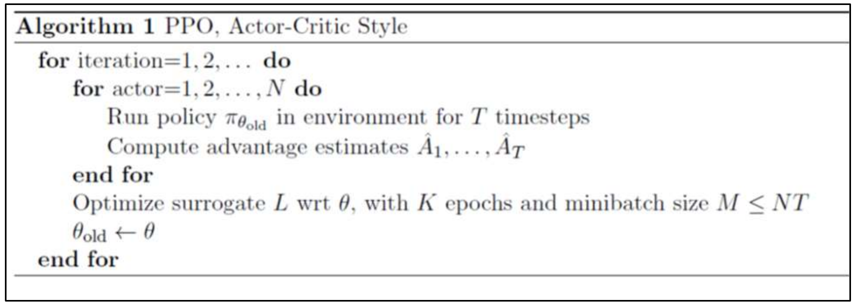
Num	Observation	Min	Max
0	hull_angle	0	2*pi
1	hull_angularVelocity	-inf	+inf
2	vel_x	-1	+1
3	vel_y	-1	+1
4	hip_joint_1_angle	-inf	+inf
5	hip_joint_1_speed	-inf	+inf
6	knee_joint_1_angle	-inf	+inf
7	knee_joint_1_speed	-inf	+inf
8	leg_1_ground_contact_flag	0	1
9	hip_joint_2_angle	-inf	+inf
10	hip_joint_2_speed	-inf	+inf
11	knee_joint_2_angle	-inf	+inf
12	knee_joint_2_speed	-inf	+inf
13	leg_2_ground_contact_flag	0	1
14-23	10 lidar readings	-inf	+inf

Action Vector

Num	Name	Min	Max
0	Hip_1 (Torque / Velocity)	-1	+1
1	Knee_1 (Torque / Velocity)	-1	+1
2	Hip_2 (Torque / Velocity)	-1	+1
3	Knee_2 (Torque / Velocity)	-1	+1

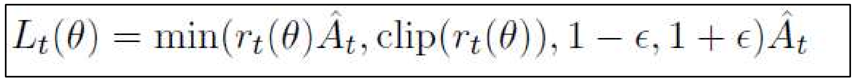
- Physics engine is Pybullet based
- Both easy and hardcore environment have same inputs and outputs
- Reward of -100 when robot falls down and +300 on completion
- Reward function in algorithm is normalized to give value of 300 on mission completion

ALGORITHMS – PPO & ARSV2T

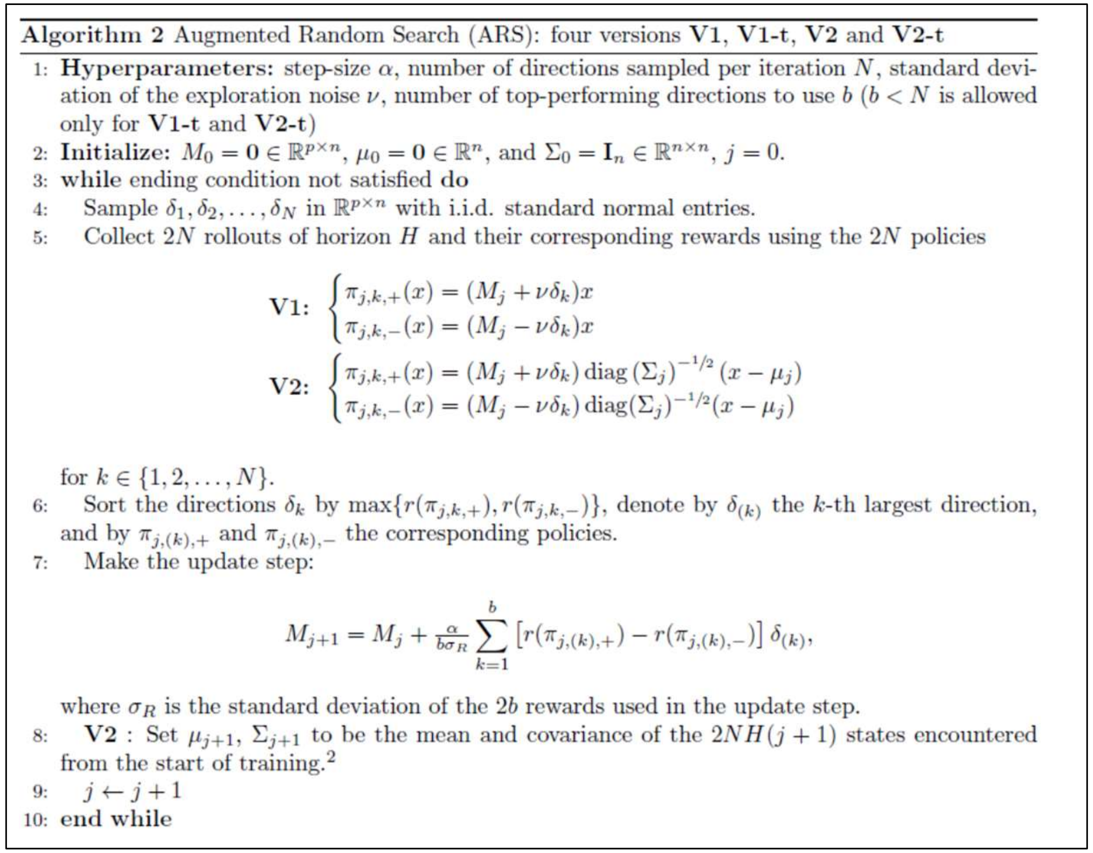


Algorithm 1 PPO, Actor-Critic Style

```
for iteration=$1, 2, \ldots$ do
    for actor=$1, 2, \ldots, N$ do
        Run policy $\pi_{\theta_{\text{old}}}$ in environment for $T$ timesteps
        Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
    end for
    Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and minibatch size $M \leq NT$
    $\theta_{\text{old}} \leftarrow \theta$
end for
```



$$L_t(\theta) = \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta)), 1-\epsilon, 1+\epsilon)\hat{A}_t$$



Algorithm 2 Augmented Random Search (ARS): four versions **V1**, **V1-t**, **V2** and **V2-t**

```
1:  Hyperparameters: step-size $\alpha$, number of directions sampled per iteration $N$, standard deviation of the exploration noise $\nu$, number of top-performing directions to use $b$ ($b < N$ is allowed only for V1-t and V2-t)
2:  Initialize: $M_0 = \mathbf{0} \in \mathbb{R}^{p \times n}$, $\mu_0 = \mathbf{0} \in \mathbb{R}^n$, and $\Sigma_0 = \mathbf{I}_n \in \mathbb{R}^{n \times n}$, $j = 0$.
3:  while ending condition not satisfied do
4:      Sample $\delta_1, \delta_2, \ldots, \delta_N$ in $\mathbb{R}^{p \times n}$ with i.i.d. standard normal entries.
5:      Collect $2N$ rollouts of horizon $H$ and their corresponding rewards using the $2N$ policies
            V1: $\begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)x \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)x \end{cases}$
            V2: $\begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)\,\text{diag}\,(\Sigma_j)^{-1/2}\,(x - \mu_j) \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)\,\text{diag}(\Sigma_j)^{-1/2}(x - \mu_j) \end{cases}$
        for $k \in \{1, 2, \ldots, N\}$.
6:      Sort the directions $\delta_k$ by $\max\{r(\pi_{j,k,+}), r(\pi_{j,k,-})\}$, denote by $\delta_{(k)}$ the $k$-th largest direction, and by $\pi_{j,(k),+}$ and $\pi_{j,(k),-}$ the corresponding policies.
7:      Make the update step:
            $M_{j+1} = M_j + \frac{\alpha}{b\sigma_R}\sum_{k=1}^{b}\left[r(\pi_{j,(k),+}) - r(\pi_{j,(k),-})\right]\delta_{(k)},$
        where $\sigma_R$ is the standard deviation of the $2b$ rewards used in the update step.
8:      V2 : Set $\mu_{j+1}$, $\Sigma_{j+1}$ to be the mean and covariance of the $2NH(j+1)$ states encountered from the start of training.[2]
9:      $j \leftarrow j + 1$
10: end while
```

ALGORITHM KEY POINTS - STUDY & RESEARCH – PPO&ARS-V2T

- PPO algorithm (clipped objective)– the derivative of the loss function of the neural network representing the actor should be same as the policy gradient estimate. An estimate of advantages weighted by the clipped policy divergence ratio is used to drive parameter updates – a single hidden layer deep network was used for both actor and critic. The actor and critic networks are exactly same according to policy gradient compatibility requirement. The critic is driven by MSE loss and actor loss function is as per policy gradient estimate

- ARS-V2t algorithm – random search directions are chosen on a sphere at each point and the update is a weighted sum of automatic differentiation of the trajectory returns. The update step is scaled by the standard deviation of the rewards collected at each iteration. It drops perturbation directions that result in the least improvement of the reward at each step

COBYLA AND ISRES

- COBYLA is a derivative free optimization method that works by using piecewise linear approximation of the objective function and defines a trust region that is successively reduced in search of optima. NLOPT library provides an interface to this algorithm, the hyper-parameters used are policy parameter sensitivity (0.01) and initial starting point. (obtained from heuristics of BipedalWalker solutions)
- ISRES is an algorithm of evolutionary strategy is based on a combination of a mutation rule with a log-normal step-size update and exponential smoothing. It uses differential variation of simplex update rule

Both these algorithms are used as a black-box and the objective function is "returns from a single rollout of the environment" using the point in parameter space provided by the algorithms

SUMMARY OF RESULTS

Algorithms		ARS	PPO	COBYLA	ÌSRES
Tasks	Metrics				
Bipedal-v2	Num iterations to score 200 & above	430 (200) 450 (315)	48 (200) 100 (262)	16**(200)	26 (200) 250 (260)
	Max score in 1000 iterations	**317**	262	205	260
BipedalHardcore-v2	Num iterations to score 200 & above	966	*	*	*
	Max score in 1000 iterations (1000*40 evals)	**265**	0	150	150

Normalized at 40 rollouts/iteration across algorithms
Numbers reported from best of 10 runs
** Starts with a known good point
* Not able to reach milestone, various hyper-parameter combinations were tried.

MAIN OBSERVATIONS - EXPERIMENTS

- BipedalWalker - Easy
  - PPO, ARS and ISRES were able to find solutions that completed the mission. COBYLA requires initialization to find a solution that completes the mission
  - ISRES and ARS are faster – fastest depends on hyper-parameter settings.

- BipedalWalker - Hardcore
  - Only ARS was able to find solutions with 250+ score, however the solution was not scalable to all randomized environments.

MAIN CONTRIBUTIONS

- Software Contributions
  - Python implementation of ARS-V2t (open source enhancements)
  - Python implementation of PPO-clipped (open source enhancements)
  - Integration of NLOPT library with open-AI Gym (new framework)
    - NLOPT library contains several legacy optimization algorithms – with/without constraints, global/local, gradient/non-gradient etc

- Research Contributions
  - Detailed study of ARS and PPO and application of theory in a programmable environment
  - Verified claims that ARS is faster than PPO on "both" tasks in finding "best" solutions
  - Validated claims of ARS that "linear" representation of policy is sufficient for both tasks
    - Hidden layer policy representation does not give advantage for both tasks (simple and easy terrain)
    - Modified policy representation to include bias and tanh clipping to match problem at hand and improve variance across rollouts
  - Observed that traditional optimization algorithms (randomized search and trust region based) also work on the problem under study – without any extensive hyper parameter tuning (saving compute and time in the process)
    - ISRES was able to find better solutions in shorter time than other RL algorithms on easy problem, however better averaging techniques are needed for hardcore version to scale the solution for different rollouts.
    - COBYLA worked better than PPO on harder problems.

CONCLUSIONS AND NEXT STEPS

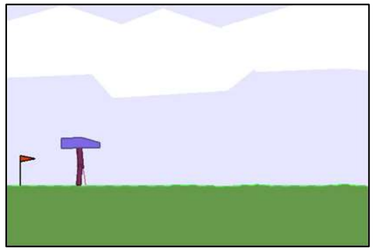
- Conclusions and questions
  - ARS is the go-to algorithm for continuous action space in environments with medium complexity
  - The legacy optimization toolbox is still useful in solving many problems where e.g., PPO or ARS may fail
  - The best way forward is to use random search policies to identify the proximal regions where policies like PPO can be used to further fine-tune the results
  - Linear policies may be sufficient for most continuous control problems of low/medium complexity. If the learning algorithm cannot solve linear policies , should we even try them for deep networks?
  - Why do all members of a population in an evolutionary algorithm use the same learning rate?
- Next Steps
  - Come up with a framework to solve this problem in a cascaded manner, apply combination of randomized/evolutionary and proximal policies to yield world class results

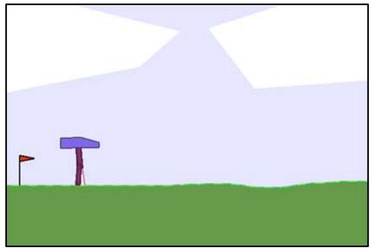
IMPLEMENTATIONS AND REFERENCES

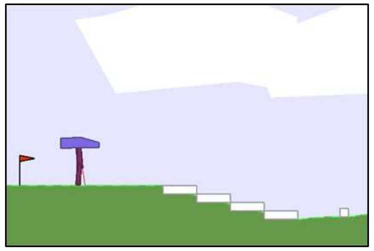
- Python implementation:
  - ARS v2T algo with a linear policy function with 100 parameters
  - ISRES integration with a linear policy function with 100 parameters
  - COBYLA integration with a linear policy function with 100 parameters
  - PPO with clipped surrogate objective function and 73K parameters

- REFERENCES
  - Proximal Policy Optimization Algorithms , John Schulman (OpenAI) et.al, Aug 2017
  - Simple random search provides a competitive approach to reinforcement learning, Benajamin Recht (Berkeley) et.al Mar 2018
  - NLOPT optimization library
  - Github (various repositories)

SOME SAMPLE VIDEOS FROM LEARNED POLICIES


ISRES


PPO


ARS