Augmented Random Search

Presenters:

1. **Gautam Suryawanshi**
2. **Prajit Krisshna Kumar**

Reinforcement Learning – CSE 510

Model free RL algorithms

- Gives solutions for controlling dynamical systems without need of actual physical models
- This systems successfully learn to play video games or games like GO and chess
- Not deployed in real world physical systems

Problems

- We need to find a best method
- Studies indicated that several RL methods are not robust to changes in parameters.
- Small change affects them a lot
- Is not trustable to deploy in real world that needs to control 100s of motors

New directions

- Evolution Strategies – derivative free optimization method, parallelized training
- Natural policy gradients for training linear policies
- ARS is combination of both

History

- Published in March, 2018 by a team from University at California, Berkeley
- ARS is enhanced version of Basic random search

BRS:

- Policy = $\pi\theta$
- We add +$\boldsymbol{\nu\delta}$ and -$\boldsymbol{\nu\delta}$ to existing policy(v<1 and it is noise) $\boldsymbol{\delta}$ is random number from normal distribution
- Apply the actions and get the rewards
- Update the policy using $\theta^{j+1} = \theta^{j} + \boldsymbol{\alpha}.\Delta$
- Where $\Delta = 1/N * \Sigma[r(\theta+\boldsymbol{\nu\delta}) - r(\theta-\boldsymbol{\nu\delta})]\boldsymbol{\delta}$

[redacted]

Motivation

One of the most mind-blowing algorithms in reinforcement learning,

Up to **15 TIMES FASTER** than other algorithms with higher rewards in specific applications.

Does not require Deep Learning.

Algorithm

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

1: **Hyperparameters:** step-size α, number of directions sampled per iteration N, standard deviation of the exploration noise ν, 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

$$\textbf{V1:} \begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)x \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)x \end{cases}$$

$$\textbf{V2:} \begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k) \operatorname{diag}(\Sigma_j)^{-1/2} (x - \mu_j) \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k) \operatorname{diag}(\Sigma_j)^{-1/2} (x - \mu_j) \end{cases}$$

for $k \in \{1, 2, \ldots, N\}$.

6: Sort the directions δ_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:

$$M_{j+1} = M_j + \tfrac{\alpha}{b\sigma_R} \sum_{k=1}^{b} \left[r(\pi_{j,(k),+}) - r(\pi_{j,(k),-}) \right] \delta_{(k)},$$

where σ_R is the standard deviation of the $2b$ rewards used in the update step.

8: **V2** : Set μ_{j+1}, Σ_{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**

.

Simplified Explanation

- Add Random Noise(δ) to the weights **ϴ.**
- Run a test.
- If reward improves keep the weights.
- Otherwise discard.

Method of Finite Differences

- Generate a random noise(δ) of the same shape of the weights (**ϴ)**
- Clone two versions of our weights.
- Add the noise to **ϴ**[+], subtract from **ϴ**[-]
- Test both versions for one episode each, collect r[+] , r[-]
- Update the weights **ϴ +=** α**(r[+] – r[-]).** δ
- Test and repeat for maximum performance.

Training Loop

- Generate num_deltas deltas and evaluate positive and negative.
- Run num_deltas episodes with positive and negative variations.
- Collect rollouts as (r[+],r[-],delta) tuples.
- Calculate the standard deviation of all rewards.
- Sort the rollouts by maximum reward and select the best num_best_deltas rollouts.
- Step = sum((r[+] – r[-])*delta), for each best rollout.
- Theta += learning_Rate/(num_best_deltas*sigma_rewards)*step
- Evaluate: play an episode with the new weights to measure improvement.

Results

- Episode 1

Results

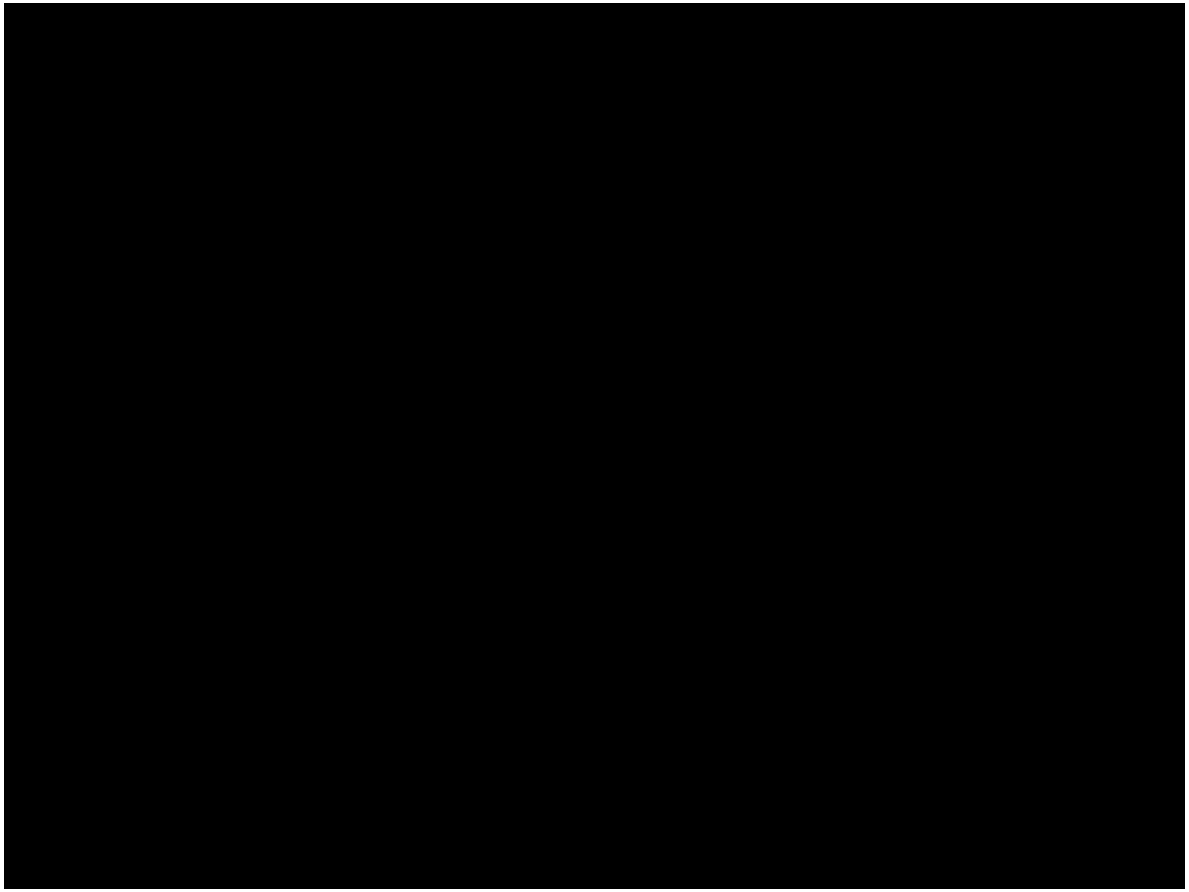
- Episode 1000

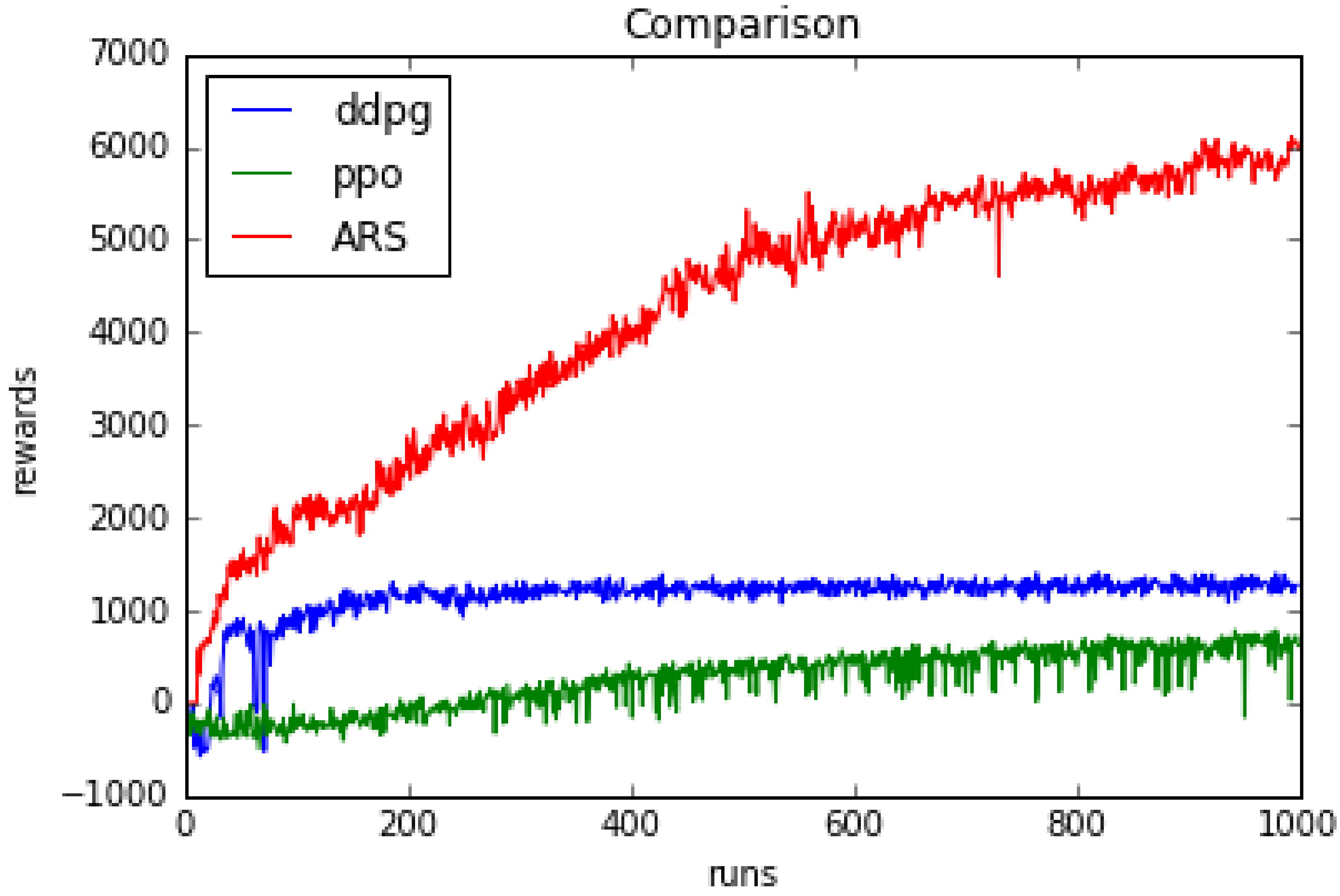
Comparison with DDPG

- At 1000 episode

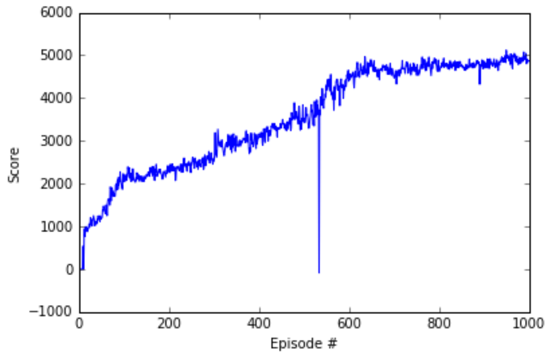
Comparison with PPO

- At 1000 episode

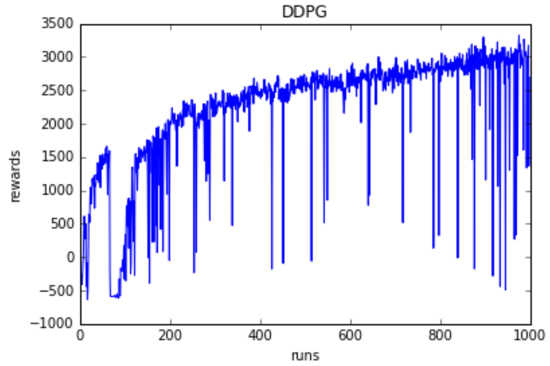



Comparison
ddpg
ppo
ARS
rewards
runs
7000
6000
5000
4000
3000
2000
1000
0
−1000
0
200
400
600
800
1000

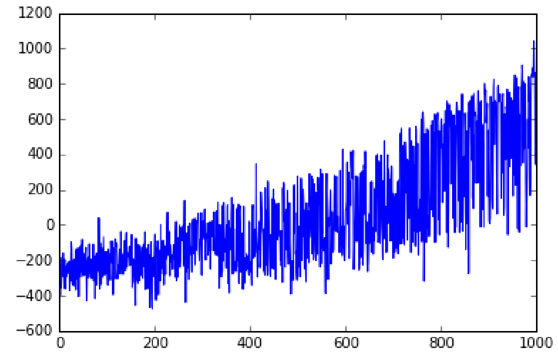
ARS



DDPG



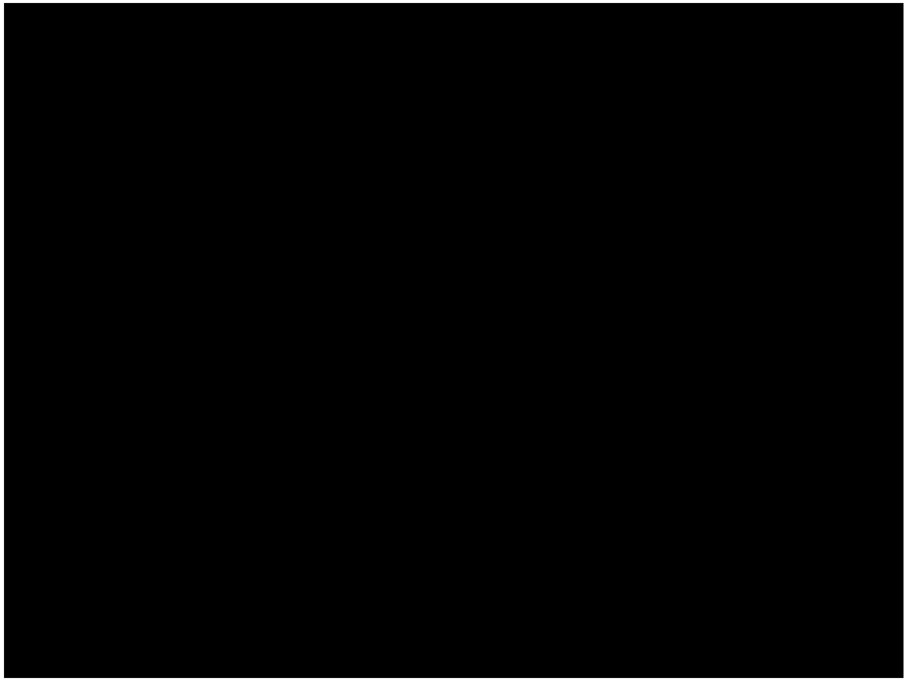
PPO

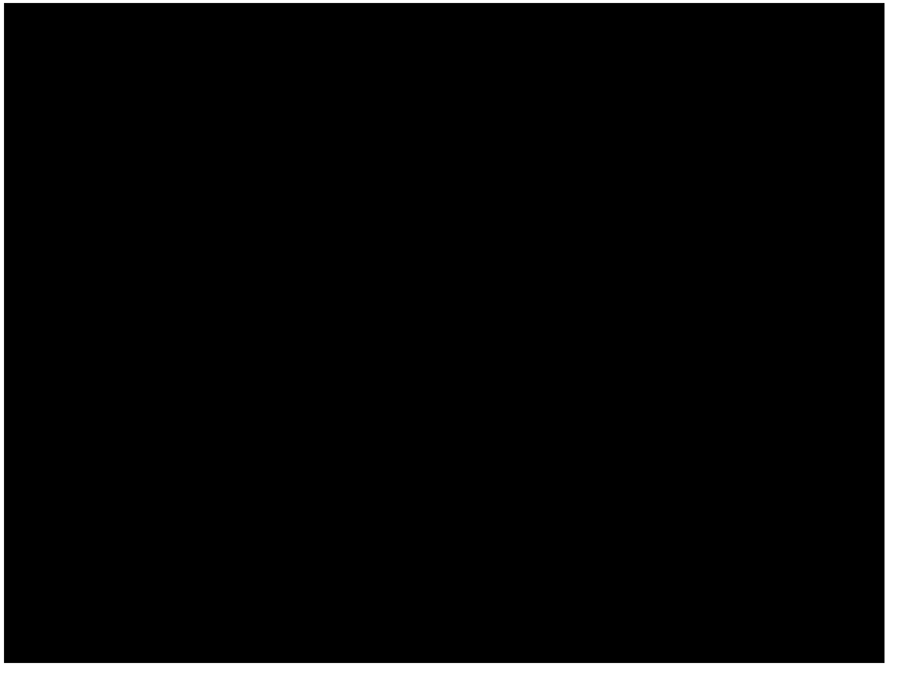


Comparing rewards

- Time for ARS to run : 5630 seconds
- Time for PPO to run : 2142 seconds
- Time for DDPG to run : 9270 seconds

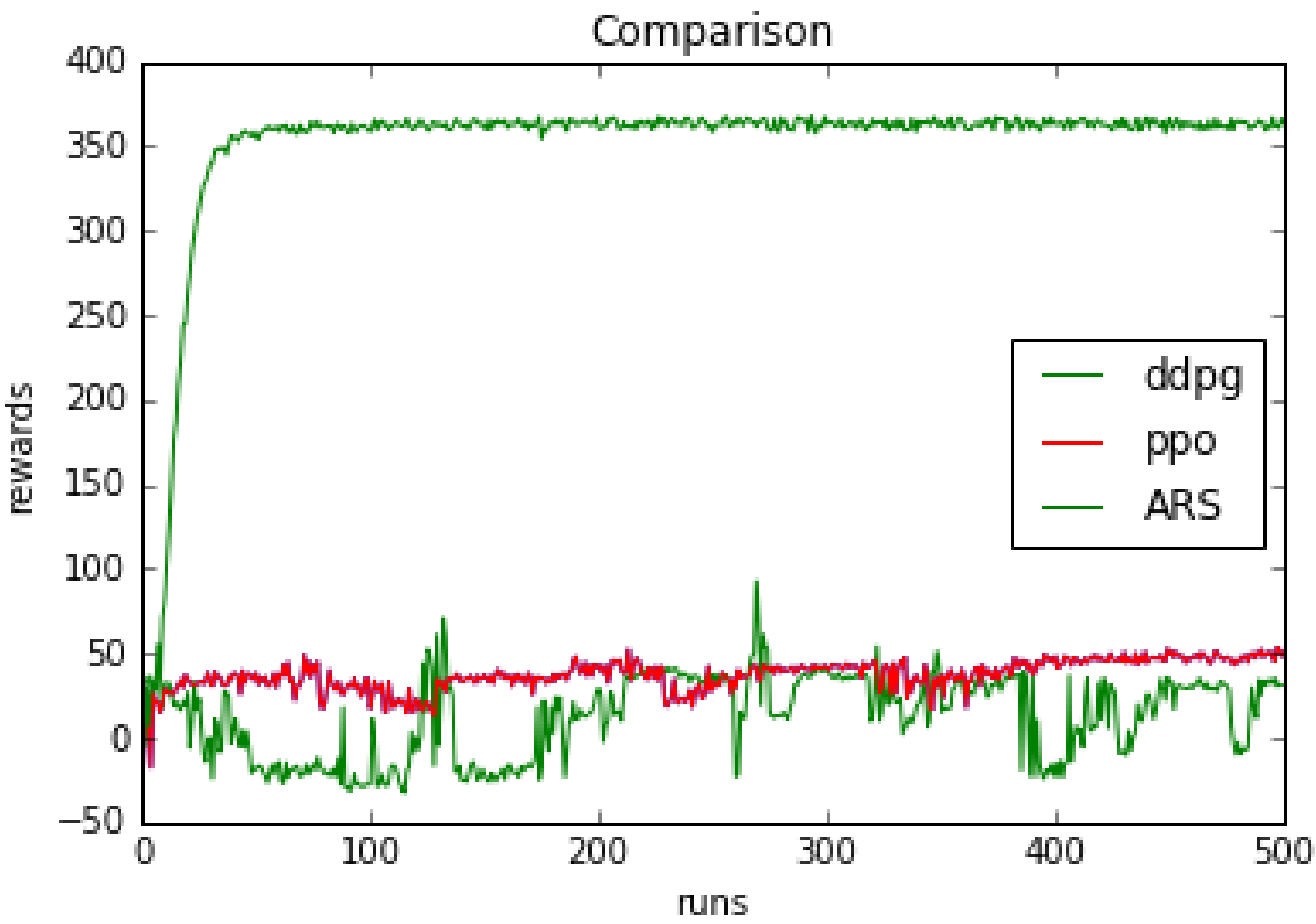
Humanoid





Bipedal Walker

Results of Swimmer



References

1. https://towardsdatascience.com/introduction-to-augmented-random-search-d8d7b55309bd
2. https://arxiv.org/pdf/1803.07055.pdf)
3. Codes for DDPG and PPO from https://github.com/Anjum48/rl-examples/