Simple random search of static linear policies is competitive for reinforcement learning

Ablation study

Slides by: *Ivan K. 0860838*
GitHub: BRS and ARS models

Outline

Intro

Simple RL baseline for continuous control.

I will compare 2 optimization methods covered in the paper:

- Augmented Random Search (ARS)
- Base Random Search (BRS)

And 2 linear models, covered on next slide.

Linear models

First version:

$$\text{action} = \pi(s) = (M + \mathcal{N}(0,1)) \cdot s$$

M - is parameters of linear model (n_action,n_state)

Second version:

$$\text{action} = \pi(s) = (M + \mathcal{N}(0,1)) \cdot \operatorname{diag}(\Sigma_j)^{-\frac{1}{2}} \cdot (s - \mu_j)$$

Where Sigma and mu are standard deviation and mean of current sampled states.

BRS update

We perturbed parameters of our model and check what reward we will get based on those weights. Then use the rewards to update our model parameters.

We run model some number of rollouts and do update on collected (reward$^+$, reward$^-$, noise) tripples.

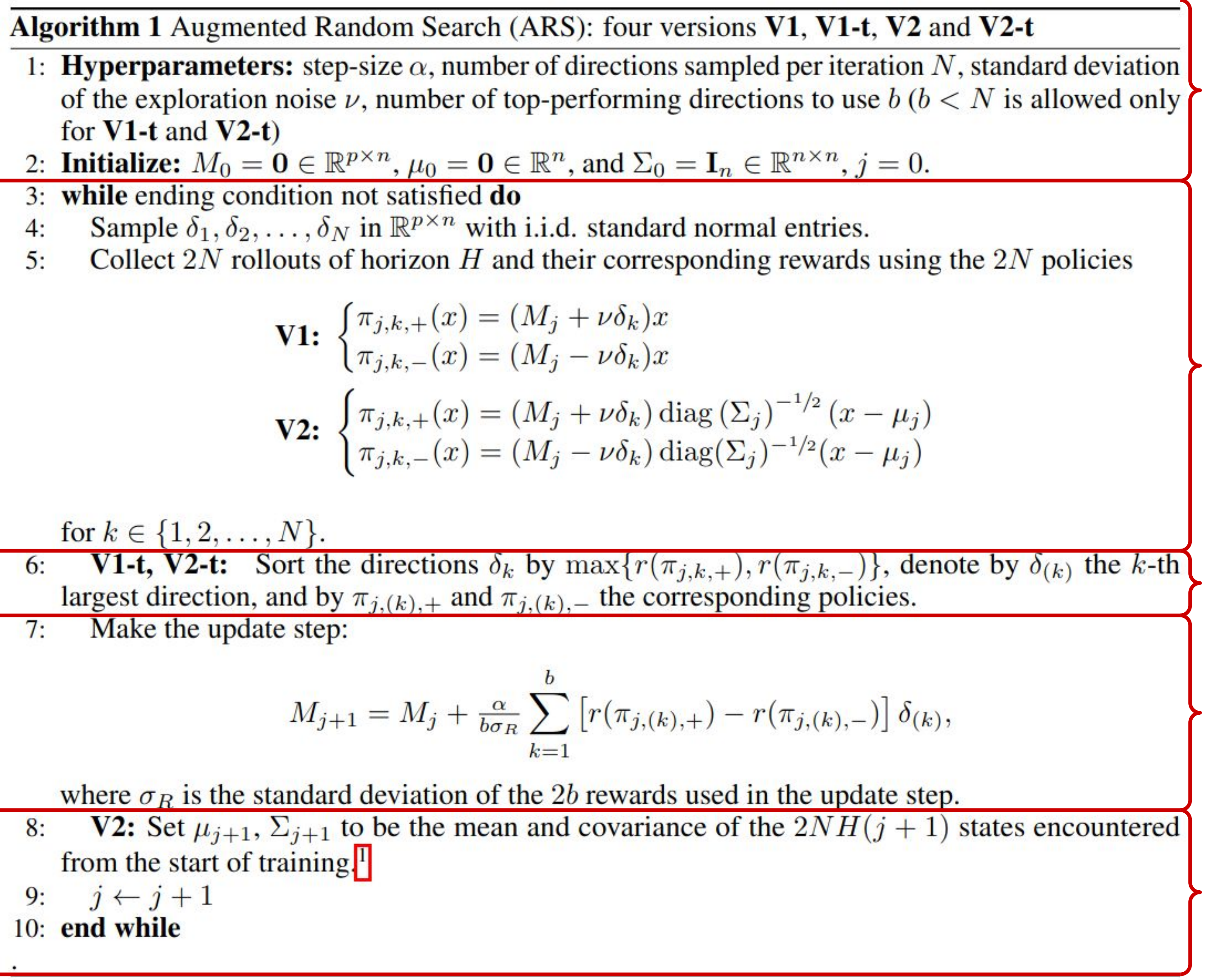
$$\delta_i = \mathcal{N}(0, \sigma_i)$$

$$M_{i+1} = M_i + \frac{r(\pi_{M_i+\delta_i}) - r(\pi_{M_i-\delta_i})}{\sigma_i}\delta_i$$

ARS update

Here we do the same thing as with BRS, but instead of doing reward of all collected triples, we sort them by reward and get ***b*** best ones. Also we use std of the collected rewards and alpha as learning rate.

$$M_{i+1} = M_i + \frac{\alpha}{b\sigma_R} \sum_{k=0}^{b} [r(\pi_{M_i+\delta_k}) - r(\pi_{M_i-\delta_k})]\delta_k$$



Algorithm 1 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 $\delta_1, \delta_2, \dots, \delta_N$ 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: $\begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)\operatorname{diag}(\Sigma_j)^{-1/2}(x - \mu_j) \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)\operatorname{diag}(\Sigma_j)^{-1/2}(x - \mu_j) \end{cases}$
        for $k \in \{1, 2, \dots, N\}$.
6:      V1-t, V2-t: 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.[1]
9:      $j \leftarrow j + 1$
10: end while
```

Initialize matrices & parameters

Make predictions with random weights offset, for both versions of the model

Sort trajectories by reward

Update weights of an linear model

Calculate additional variables

Experiments setup

Each model trained with same number of epochs(10k) and same start random seed(42), then evaluated on environments(1k) initialized from a fixed list of random seeds.

Also linear model policies parameters will be initialized with 0 for each model, to ensure reproducibility.

For all experiments was used “Pendulum-v1” environment.

You can see precise parameters values in the provided source code files.

Experiments setup

ARS constructed out of BRS model by adding elements that increase model complexity. By adding one new element to previous model(half buffer update to BRS) we can generate new model. Elements include:

- Use full replay buffer for update
- Use random subset of replay buffer for update
- Use best part of sorted replay buffer for update
- Normalize state before training

For BRS and ARS, and for linear_v1 and linear_v2 model, so 16 tests in total.

Tests setup BRS models

✅BRS_1	Use full replay buffer for update
✅BRS_1_RS	Use random subset of replay buffer for update
✅BRS_1_SS	Use best part of sorted replay buffer for update
✅BRS_1_SSn	Normalize state before training
✅BRS_2	Use full replay buffer for update
✅BRS_2_RS	Use random subset of replay buffer for update
✅BRS_2_SS	Use best part of sorted replay buffer for update
✅BRS_2_SSn	Normalize state before training

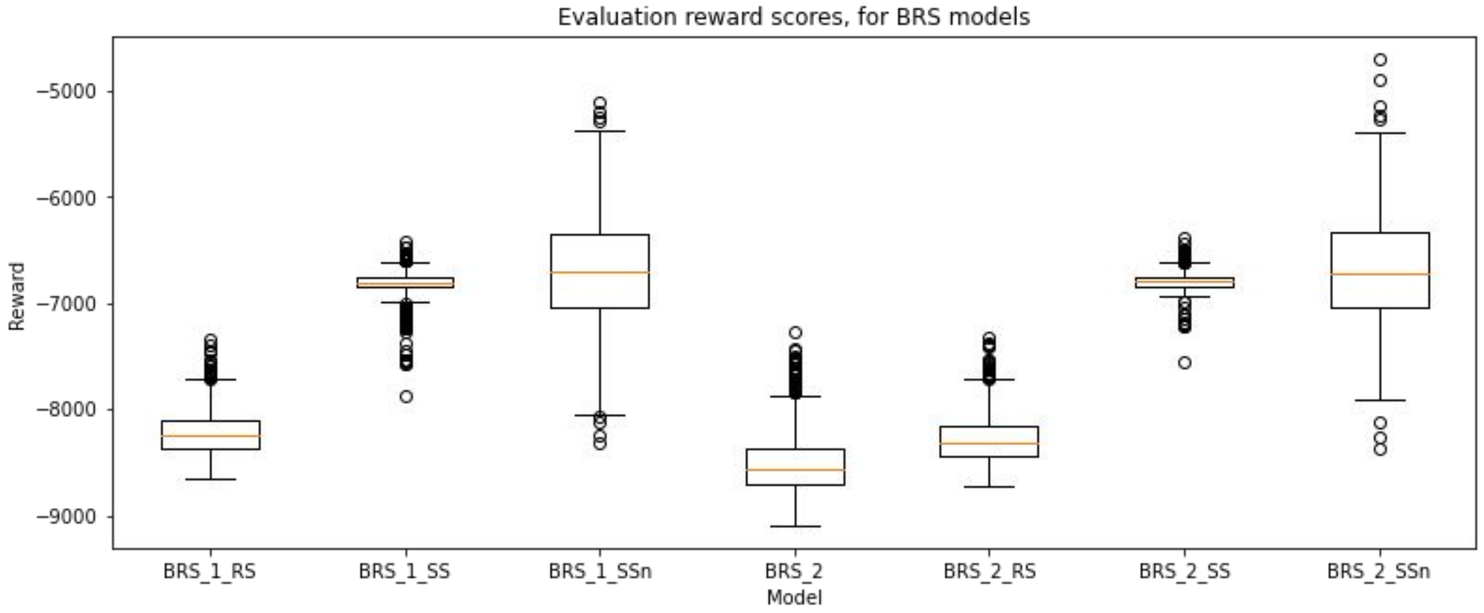
Tests setup ARS models

✅ARS_1	Use full replay buffer for update
✅ARS_1_RS	Use random subset of replay buffer for update
✅ARS_1_SS	Use best part of sorted replay buffer for update
✅ARS_1_SSn	Normalize state before training

✅ARS_2	Use full replay buffer for update
✅ARS_2_RS	Use random subset of replay buffer for update
✅ARS_2_SS	Use best part of sorted replay buffer for update
✅ARS_2_SSn	Normalize state before training

ARS_1_SSn and **ARS_2_SSn**: final models of this paper

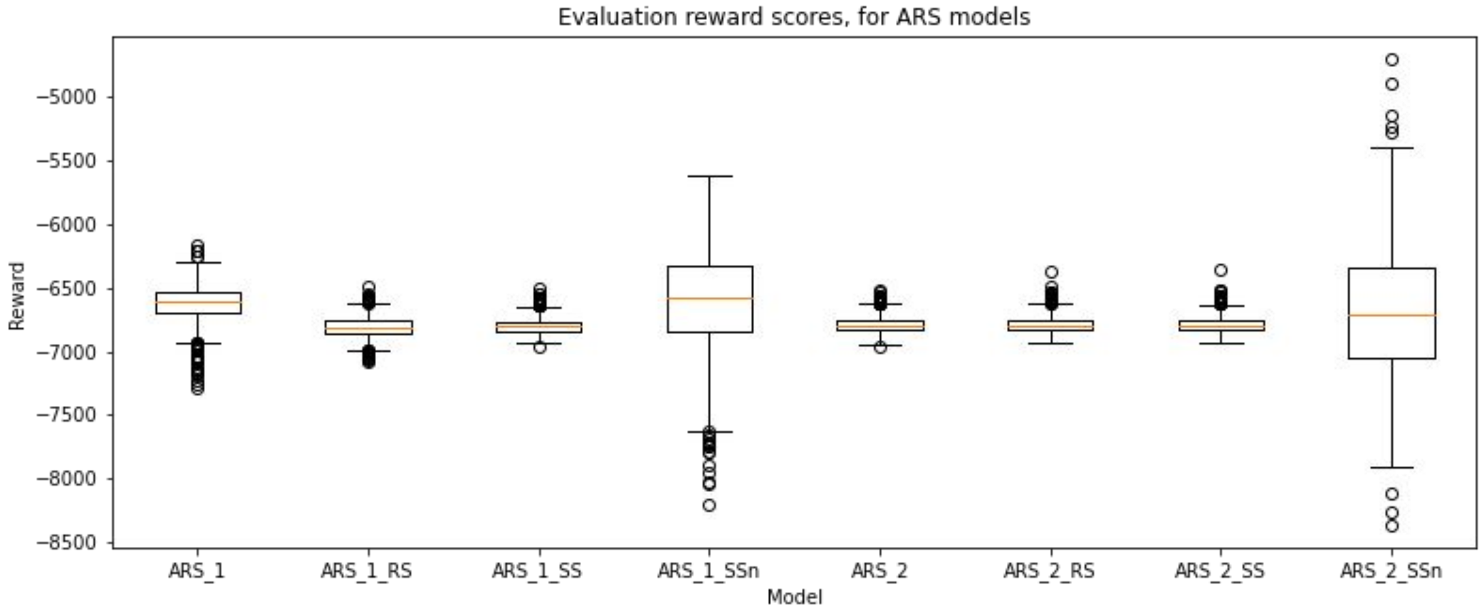
BRS scores



Parameters:

SEED=42	Rollout=64	Eval env=1k	alpha=2e-2
Train steps=10k	B=4	STD=1	

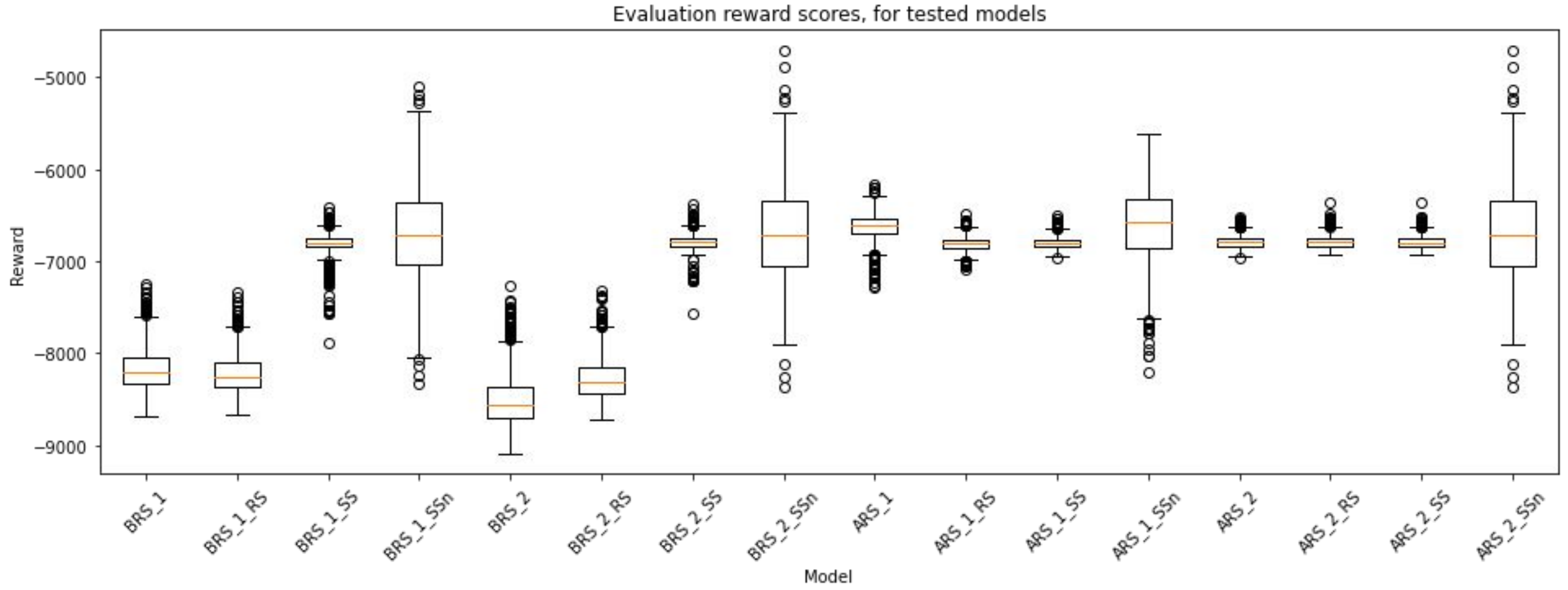
ARS scores



Parameters:

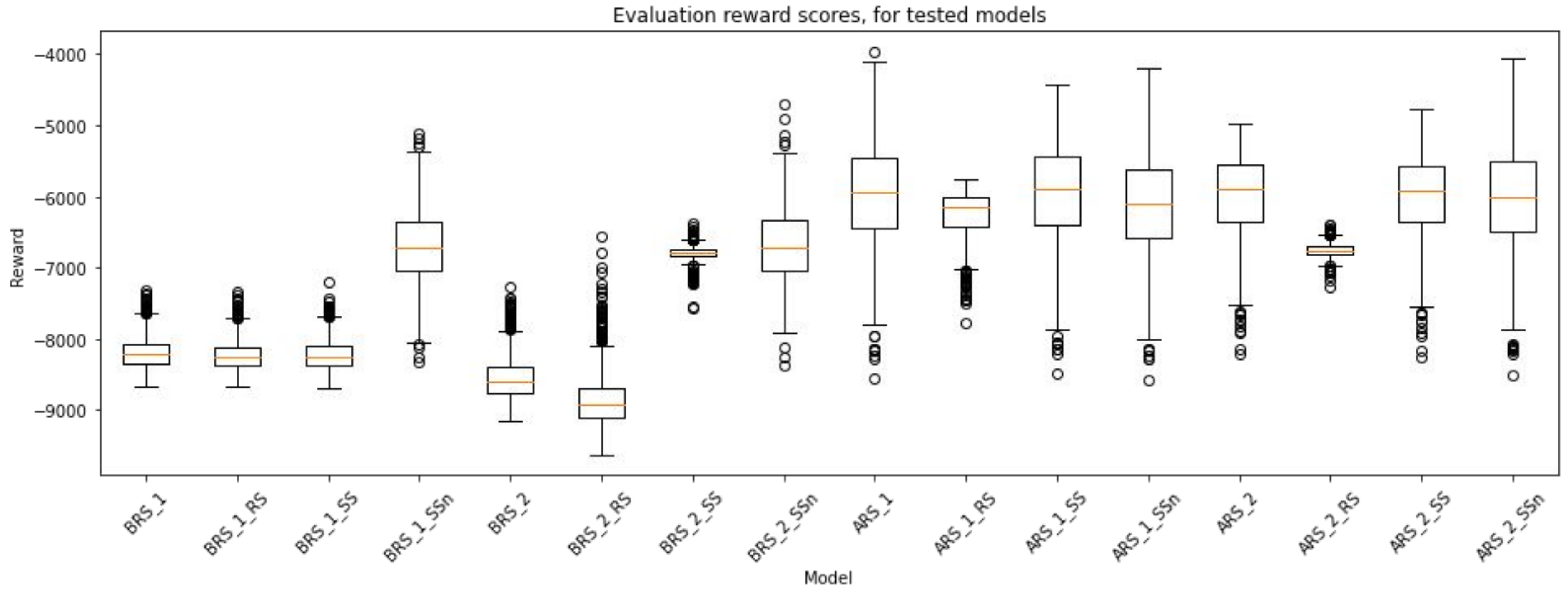
SEED=42	Rollout=64	Eval env=1k	alpha=2e-2
Train steps=10k	B=4	STD=1	

Evaluation results #1



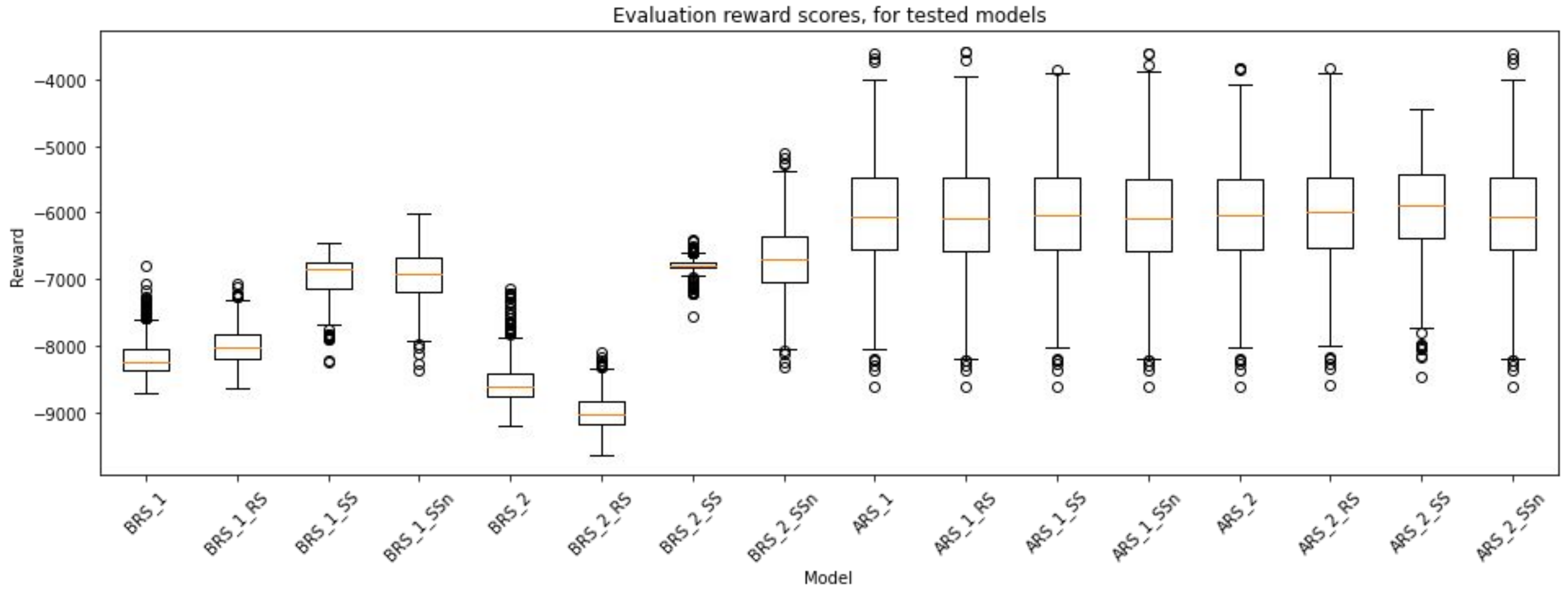
```
Parameters:
SEED=42           Rollout=64  Eval env=1k  alpha=2e-2
Train steps=10k   B=4         STD=1
```

Evaluation results #2



```
Parameters:
SEED=42           Rollout=64   Eval env=1k  alpha=2e-2
Train steps=10k   B=4          STD=0.1
```

Evaluation results #3



Parameters:

```
SEED=42            Rollout=64   Eval env=1k  alpha=2e-2
Train steps=10k    B=16         STD=0.01
```

Discussion

- Algorithm for BRS and ARS was not in public domain and was implemented from scratch with help of external libs.
- Due to hardware limitations only “Pendulum-v1” environment was used, but code can be tested on other continuous environments as well.
- All models that normalize state before training have wider range of behaviors, you can see that variance across tests are the highest.

Discussion

How each component affects ARS model?
Use full replay buffer for update:
    Performs good, sometimes even better then sorted buffer

Use random subset of replay buffer for update:
    This was reference to see if “*best part of sorted replay buffer for update*” actually works. As we can see sorted replay update weights better then a random one, but not for each case.

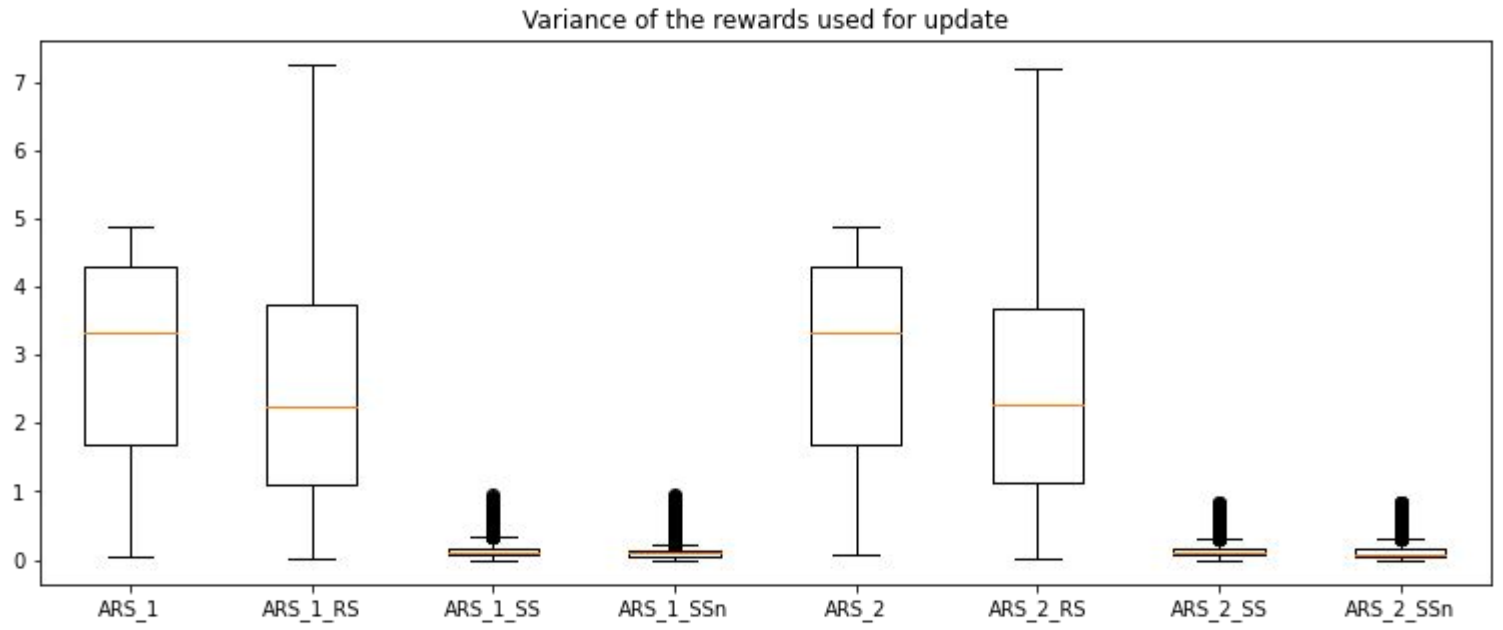
Normalize state before training:
    Lead to high variance in policy behavior.

Conclusions

We can see that policies trained with ARS perform on average better and have smaller variance of accumulated rewards, that means more stable policy, than one that trained with BRS.

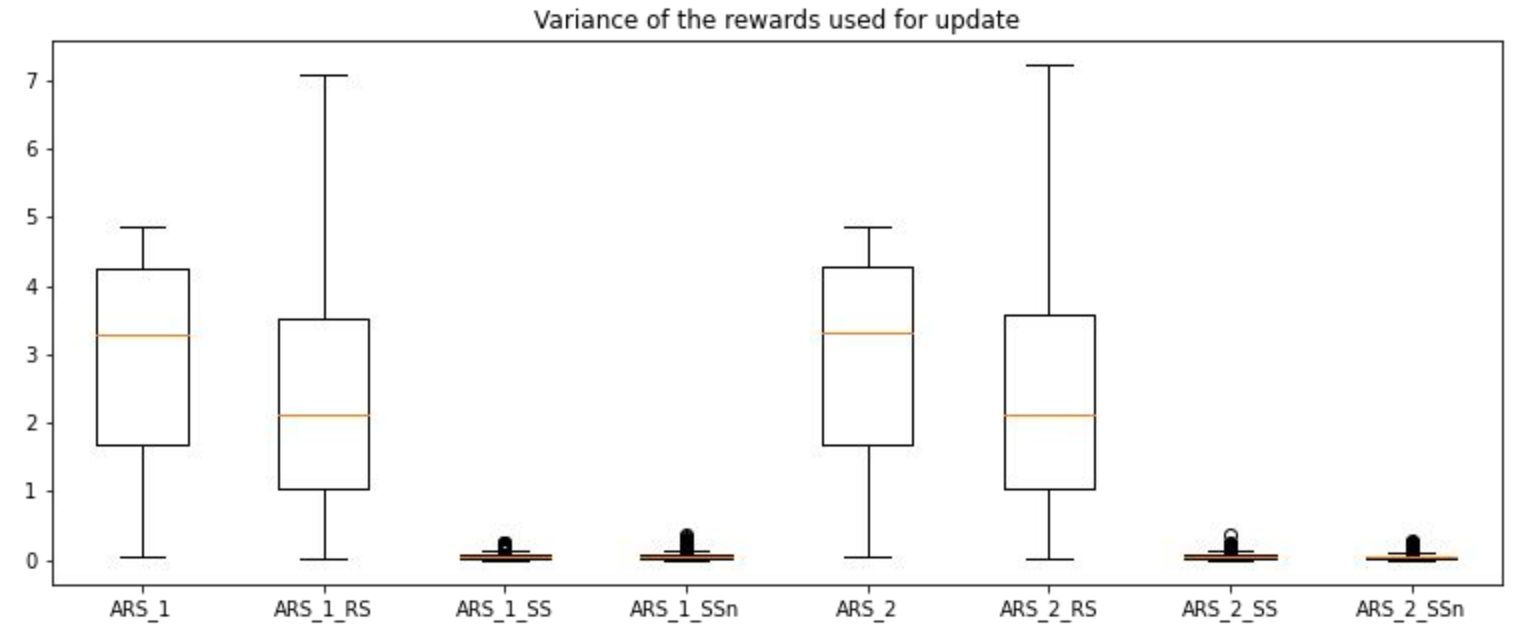
Appendix

ARS training rewards STD #1



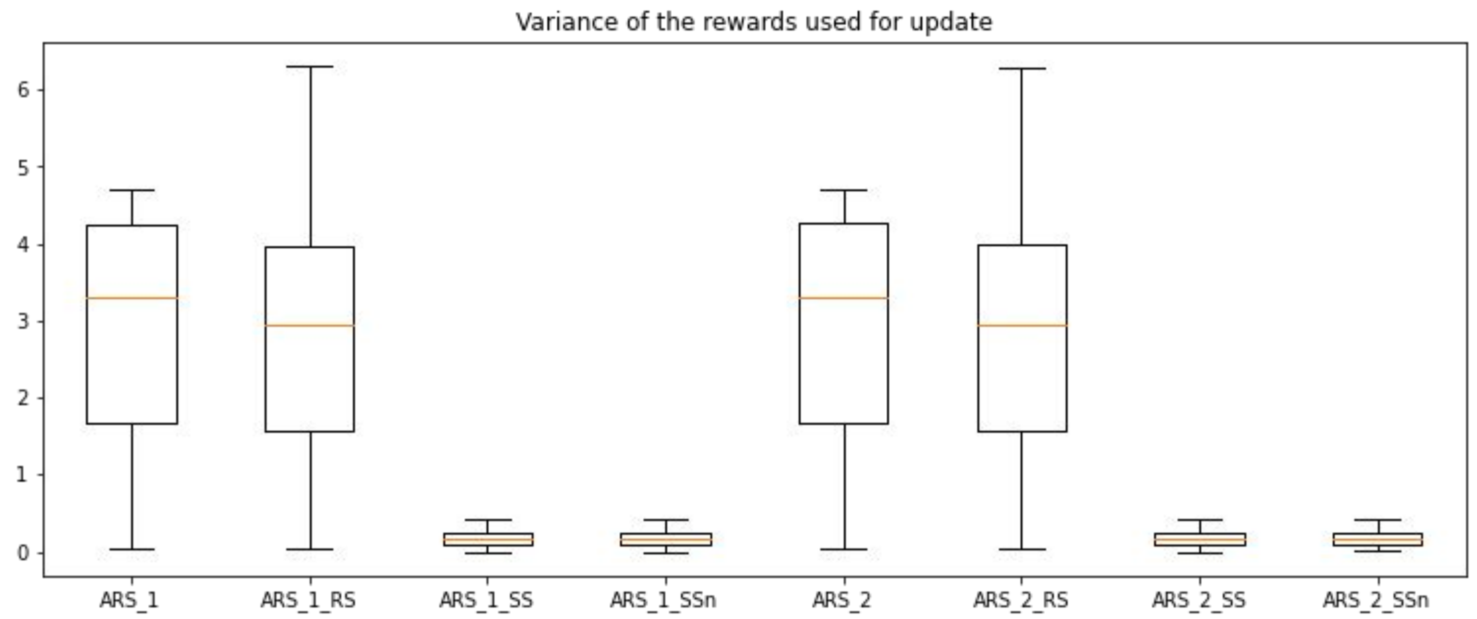
```
Parameters:
SEED=42          Rollout=64   Eval env=1k  alpha=2e-2
Train steps=10k  B=4          STD=1
```

ARS training rewards STD #2



```
Parameters:
SEED=42          Rollout=64   Eval env=1k  alpha=2e-2
Train steps=10k  B=4          STD=0.1
```

ARS training rewards STD #3



```
Parameters:
SEED=42           Rollout=64   Eval env=1k  alpha=2e-2
Train steps=10k   B=16         STD=0.01
```