REINFORCEMENT LEARNING 1

The power of self-critic?

Policy based estimates as a baseline.

by Maximilian Knaller, Steven van de Graaf, Kumar Pratik and Ruben van Erkelens

The concept of baselines was introduced in Policy Gradient Methods (PGM) to fight an always present battle in Machine Learning, the bias-variance tradeoff. More precise with an algorithm called REINFORCE that, as all other Monte Carlo algorithms, is unbiased but suffers from high variance.
Baselines are a clever way to reduce that variance while not adding any bias. However, they generally come with the cost of learning the baseline in addition to the policy. This introduces several extra challenges, like the need to worry about convergence of a second network or the question if the chosen baseline target can be efficiently learned for the environment at all.
Thus, a new extension called self-critic baselines tries to overcome these issues by estimating a good baseline based on the current policy itself. Using our policy to criticize our policy? While this might sound strange at first, we will show you throughout this post why this is a valid idea, how it works and finally what you can and can't do with it.

If some terms sounded unfamiliar or you feel like you need a refresh on the theory of PGM, REINFORCE or standard baselines, there is no need to go anywhere else, we provide it in the next section. However, if your ready to dive into the world of self-critic, feel free to skip the recap. Let’s start our investigation.

Recap

Policy Gradient Methods (PGM) and REINFORCE

PGM is part of the model-free policy-based function-approximation family in RL. This means we use a function approximator $f_\Theta(s_t)$ (e.g. a Neural Network with weights Θ) to

Contribution in percent per author respectively [30, 25, 25, 20]

directly map the current state s_t to a probability distribution over the possible actions a_t. Thus, the function approximator represents our policy $\pi_\Theta(s_t)$.

$$\pi_\Theta(a|s_t) = p(a_t \mid s_t, \Theta) = f_\Theta(s_t)$$

In simpler terms, we aim to **learn the policy directly**, in comparison to value-based methods (e.g. TD(0), DQN) where we derive the policy form learned state/state-action values.

Why do we care about PGM?

With the requirement of our policy π_Θ being differentiable w.r.t. Θ, PGM buys us many advantages over value-based methods. It has the ability to learn stochastic and deterministic policies, can ensure that changes in the policy occur smoothly or include a prior on the policy. These advantages make PGM very popular.

How does it work in practice?

The main idea of PGM consists of two steps. First, we **rate sampled trajectories** following our policy π_θ **by a scalar performance measure** $J(\Theta)$. Secondly, **use the gradient** of $J(\Theta)$ to update the weights Θ **by applying Stochastic Gradient Ascent** (SGA).

Getting a gradient on $J(\Theta)$ can be hard and many different ways were proposed, which gave rise to many algorithms in the family of PGM. However, they all follow the same general approach. Thus, let's first just assume that we are given a stochastic estimate $\overline{\nabla_{\Theta_t} J(\Theta_t)}$ of the gradient, whose expectation approximates the real gradient of the performance measure $J(\Theta)$ with respect to its argument Θ_t. Then we can simply update our policy parameters by applying

$$\Theta_{t+1} = \Theta_t + \alpha \, \overline{\nabla_{\Theta_t} J(\Theta_t)}$$

with α being the learning rate.

What gradient are we actually using now?

Luckily, the theoretical backbone of PGM, the policy gradient theorem[1], provides us in case of episodic tasks with an analytical expression:

[1] Sutton, Richard S., and Andrew G. Barto. "Reinforcement Learning: An Introduction."

$$\nabla_\Theta J(\Theta) = E_{\pi_\Theta}[\sum_a q_{\pi_\Theta}(S_t, a) \nabla \pi_\Theta(a \mid s_t)]$$

It' is outside our scope here to discuss this in detail, but the intuition is that the performance of a policy starting at any random state s_0 is equivalent to the true value of this state under the policy π_Θ. Or in other "words", that $J(\Theta) = v_{\pi_\Theta}(s_0)$ holds.

This lead directly to one of the first PGMs called ***REINFORCE***[2] , which uses

$$\nabla_\Theta J(\Theta) = E_{\pi_\Theta}[\sum_a q_{\pi_\Theta}(S_t, a) \nabla \pi_\Theta(a \mid s_t)] = E_{\pi_\Theta}[G_t \frac{\nabla \pi_\Theta(A_t|S_t)}{\pi_\Theta(A_t|S_t)}]$$

in the general SGA update rule for PGM.

Baselines

While the REINFORCE update can be proven to **converge to a local optimum** given a small enough learning-rate it suffers from **high variance** due to the noise samples of the return G_t. This problem gave rise to a generalization of the policy gradient theorem that introduced a ***baseline*** $b(*)$ as a comparison to the sampled return.

$$\nabla_\Theta J(\Theta) = E_{\pi_\Theta}[(G_t - b(*)) \frac{\nabla \pi_\Theta(A_t|S_t)}{\pi_\Theta(A_t|S_t)}]$$

The baseline $b(*)$ can be a function of any variable $*$, even a random variable, as long as it does not depend on the action. The policy gradient theorem remains valid because the expectation does not change by the subtraction of the baseline. This means the baseline has **no impact on the bias, but it can drastically reduce variance**.

Learned baselines

The natural and common choice for the **baseline is a learned estimate of the state-value** $b(s) = v_\Phi(s)$, which is also be learned using a function approximator with weights Φ. The observant reader will note that this is closely related to the assumed performance metric $J(\Theta)$ in the policy gradient theorem. Thus, the single sample of G_t will be normalized by the current estimation of the expected return $E_{\pi_\Theta}[G_t] \approx v_\Phi(s) = b(s)$. This

[2] "Simple statistical gradient-following algorithms for ...."
https://link.springer.com/article/10.1007/BF00992696. Accessed 14 Oct. 2019.

feels natural, because **only sampled returns that differ strongly from the current expected return will lead to a strong update in policy**.

We also want to take a moment to note **the connection to Actor-Critic methods**. These also train a second estimator on the state-value function. However, they do not only use this estimation as a baseline. In Actor-Critic methods the state-value estimation is also used to bootstrap the return:

$$\nabla_\Theta J(\Theta) = E_{\pi_\Theta}[(G_{t:t+n} - v_\Phi(s_t))\frac{\nabla\pi_\Theta(A_t|S_t)}{\pi_\Theta(A_t|S_t)}] \quad with \quad G_{t:t+n} = \sum_{k=0}^{n-1}\gamma^k R_{t+k+1} + \gamma^n v_\Phi(s_{t+n})$$

Which introduces a bias to the expectation and makes it worth to distinguish between the methods.

Why self-critic is worth exploring and how it works?

The approach of learning and baseline, e.g. the state-value $v_\Phi(s)$ is widely used and works well. So why should we even look for an alternative? One clear reason is, the **cost and complexity of learning** a second estimator. Learning comes with many challenges, especially in case of NNs (hyperparameters, architecture, optimizer, ...) and we already have one function approximator to train for our policy. So any path that could prevent that complexity is worth exploring. Furthermore, there are tasks and environments that make it very **hard to properly fit a value function in the first place**, e.g. if our state-space is vast and hard to capture like it is in the case of image captioning.

This second reason and it's example directly links us to the introduction of the self-critic concept. Rennie et al. (2017)[3] proposed to use a **sample average of the return following the learned policy** π_θ as a baseline in image captioning. More concrete, they sampled the average return of the current policy on the test-set and successfully used this baseline in training.
This inspired Kool et. al (2018)[4] to show the applicability on a different environment, in

[3] "Self-critical Sequence Training for Image Captioning." 2 Dec. 2016, https://arxiv.org/abs/1612.00563. Accessed 14 Oct. 2019.
[4] "Attention, Learn to Solve Routing Problems!." https://arxiv.org/abs/1803.08475. Accessed 14 Oct. 2019.

which it might be even harder to fit a value function directly, namely the Traveling Scientist Problem (TSP). He made the important observation that ***"The difficulty of an instance can (on average) be estimated by the performance of an algorithm applied to it."***[5]. Or in other words, that the sample return of the greedy version of our policy π_θ, can successfully be used as an indication of the value/cost of an state. Thus we can define our self-critic baseline in this case as

$$b(s_t) = \frac{1}{K} \sum_{n=1}^{K} G_{t:T}{}^{greedy(\pi_\Theta)}$$

$$with \quad G_{t:T}{}^{greedy(\pi_\Theta)} = \sum_{k=0}^{T-t} \gamma^k R_{t+k+1} \quad and\ R\ following\ actions\ of\ greedy(\pi_\Theta)\,.$$

Can you see the self-critic part? While learned baselines use the same data to learn as the policy does, they are dependent on data generated by the full history of policies explored so far. **In the self-critic set-up we only use the current policy and it's behaviour to generate our baseline.**

Rennie and Kool showed us the power of self-critic, but let's be self-critic for a second ourselves and look at the possible limitations hiding behind the great advantages.
Frist, we notice that both papers showed the value of the self-critic approach in very specific and **to RL somewhat atypical environments**. However, we know that RL became mainly famous for its power to tackle difficult control tasks in which it is hard to decorelate the impact of actions that provide very noisy estimates of returns.
Secondly and closely related, both use completely deterministic environments, but as we know many of the very interesting **real world environments (e.g. in robotics) are inherently stochastic**.
Thus, we want to spend some time to extend the view on the self-critic baseline by tackling two questions:

- Does the self-critic approach work in more common RL environments?
- Which impact has environment stochasticity on the performance of baselines and specifically the self-critic approach?

[5] "Attention, Learn to Solve Routing Problems!." https://arxiv.org/abs/1803.08475. Accessed 14 Oct. 2019.

Experimental Design

Let's take a moment to outline how we plan to answer these questions properly.

PGM method and baselines

We decided for REINFORCE as a base of all our experiments, because it is known to suffer form especially high variance. Understand why it is perfect? Correct! Our baselines shall help to reduce the variance, so if we want to study their performance it is good to choose a base that has a lot of variance to reduce.

In terms of baselines we use a value network for the learned baselines. This is a very common choice and we will give the details a closer look in the next subchapter. As a self-critic baseline, we use a **sample of the return following the greedified learned policy (SRGP),** which is equal to the presented approach of Kool with $K = 1$. Furthermore, it is always a good idea to have an independent measure to compare results to, thus we also include REINFORCE without baseline.

Putting it into a picture we can derive the following experiment structure.

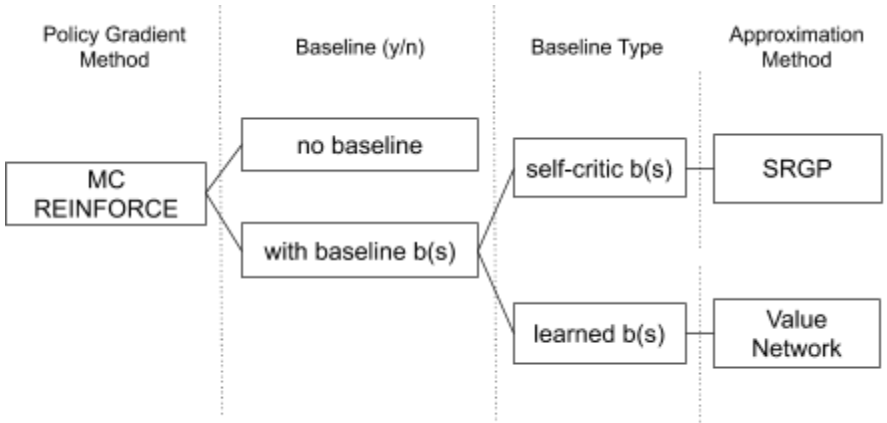

Fig.1: An overview on the compared methods and how they connect.

Environments

Our first question aims to check the applicability of self-critics to more RL typical environments. To make results and observations from our experiments more

generalizable, we choose environments with both **discrete and continuous state space**. While the discrete state space allows for a better analysis, the continous is a more realistic application of function approximators. The OpenAI gym library[6] provides great and well studied examples and is perfect for our needs.

As a **discrete state environment we use a simple 10x10 Grid World,** where the goal of the agent is to reach one of two fixed terminal states in opposite corners of the grid as fast as possible. Each step up to termination leads to a negative reward of -1. As a state the agent receives is current position in (x,y). It's possible actions are to move left, right, up or down. It should be easy to fit the value function here and the impact of stochasticity can be imagined easily. We enforce a maximum of 30 steps, to prevent very bad policies to take unreasonably long to terminate.

For the **continuous state environment we chose the CartPole task**, in which the goal is to balance an inverted pendulum for as many steps as possible on a cart by moving it left or right. Here the state space is given by the position and the velocity of the cart and pole. Each step until termination yields a reward of +1. As this task is not episodic in nature it is made so by staring with the pendulum upright and terminate if it is more than 15 degrees from vertical or 2.4 units form the center. Moreover, a maximum of 200 is set to the number of steps.

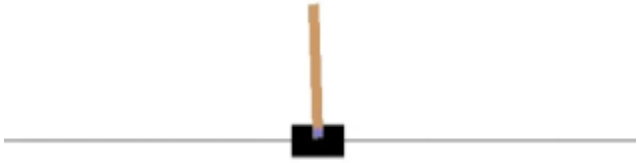

Fig.2: CartPole Task

The two chosen environments generalize to a wide range of tasks we encounter in RL, e.g. the Cartpole environment generalizes to other classic control tasks like the MountainClimber. Thus we allow to kept the number of environments in our experiments to two.

We note that the **introduction of a maximum episode length theoretical breaks the Markov Property** as time is not part of the state of the agent. Thus, the time dependent transition to termination can't be modeled. While this can lead to practical convergence problems if many episodes terminate like this (e.g. in situations where the actor found the

[6] "openai/gym: A toolkit for developing and comparing ... - GitHub." https://github.com/openai/gym. Accessed 14 Oct. 2019.

optimal policy in cart-pole), the algorithms are still decently applicable because learning is normally stopped when the actor converged. Nevertheless, this is a widespread issue in RL and is often not given enough attention.

Our second question plans on studying the performance of different baseline techniques as a function of the stochasticity of the environment. An easy and commonly used technique to do so is by introducing a **stochasticity parameter** ε that describes the probability of the environment executing a random action instead of the action taken by the agent under the policy π_Θ. This modification gives us a **straightforward way to manipulate the level of stochasticity on demand**.

Function Approximators and Hyperparameter Tuning

We use Neural Networks as function approximators at two points. First, in our base REINFORCE algorithm to estimate our actor policy. Second, to estimate state-values in the learned baseline method. In both cases we use a very simple network consisting of one hidden fully-connected RELU layer combined with a log soft-max or linear output layer, respectively.

We take a moment to note that while **PGM can learn stochastic policies** (i.e. give several actions a good chance to be sampled), the random initialization often leads to close to deterministic behaviour. Thus, we add a temperature parameter to the multinomial action draw that leads to more uniform action probabilities and ensures exploration in the first stage of training. The temperature parameter is exponentially decayed over the first 100 episodes.

As the performance of a neural network is heavily dependent on the setting of its hyperparameters, it is **important to tune all methods independently to allow for a fair comparison**. To achieve this we conduct a grid search over the discount factor, learning rate, dimension of the hidden layers and initial temperature. The value range used in these experiments can be found below in Tab. 1, where selection is based on prior experiments that identified promising regions. We tuned all experiments independently over the whole range of values, which is very expensive and time consuming. However, this makes the comparison as fair as possible. Complete results can be found on our GitHub repo that is linked at the end of this section.

Hyperparameter	Values
Discount factor	0.98, 0.985, 0.99
Learning rate	0.01, 0.001, 0.0001
Dimension of the hidden layers	64, 128, 256
Initial temperature	1.05, 1.10, 1.15

Tab.1: The hyperparameters and their values tried during grid search

Experiments

We finally are ready to start our experiments and take a moment to summarize them in the 1970-like diagram in Figure 3 and discuss last decisions. We see that our three main algorithms in the center have to be tuned and run on each of the environments and for varying degrees of stochasticity. We note that all these experiments are stochastic in nature. Thus, it is **necessary to average the results across several runs in order to be confident of our findings**. A reasonable way to determine the number of runs is to monitor the change in mean and variance while increasing the sample size. If the changes stays below a bound for successive increases in sample size, we can infer that our estimate has converged. After receiving a lower bound for each of the experiments following this approach, we empirically set the general number of runs to 150, so that it is larger than two times the maximum needed by any experiment. In a similar way we set the number of episodes to 750. These approximations does not harm the comparability, but it reduces the need of bookkeeping.

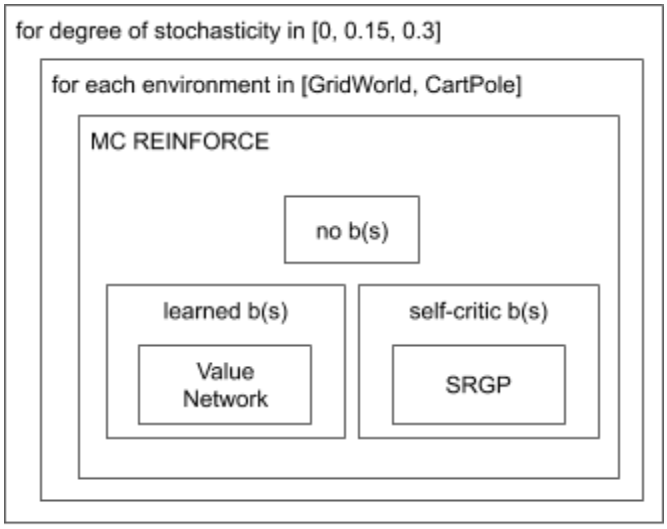

Fig.3: The set of experiments to run.

While the **methods differ in the loss expression** they use for actually updating the policy, they **all attempt to solve the common goal of optimizing the episode length**. This makes this metric the best to compare the mean and variance in performance during and

after training. Other metrics like the losses of the policy or value network can give insights in the quality of training, but do not allow reasonable comparison between different baseline methods and are thus not presented.
Let's stress again that one main advantage of a functioning baseline method is to reduce variance during training, leading to faster and more stable learning. Thus, the **variance of our performance measure will be of special interest in our evaluations**.

Significance Test

In order to be confident that the observations we receive are indeed significant and not merely by chance, we perform significance testing between each method involved. Here, the null hypothesis (H_o) is that every method involved results in the same evaluation metric, i.e mean episode length, variance in episode length. And we reject the null hypothesis (H_o), when the p-value between samples from different methods is greater than 0.05, i,e $p < 0.05$.

The episode length of the last 50 epochs was averaged for each of the 150 runs to represent the **mean average episode length after the network has converged**. This gives us a performance measure of the methods on each of the 150 seeds, which can then be used for comparison. For significance testing of mean average episode length between a pair of methods we performed *welch's t-test* (double sided). And for significance testing between variance of samples we performed the *Levene's test*, which assesses the equality of variances for a variable calculated from two or more groups. The p-values for the tests can be found in the appendix.

Implementation and other details

Besides the OpenAI environments, PyTorch and standard python libraries all methods were implemented by ourselves and can be found with all other details in our GitHub repo under https://github.com/sgraaf/rl_reproducibility_lab

Experimental Results

Finally we approach the most interesting part of our journey. So let's revisit our two research questions:

1. Does the self-critic approach work in more common RL environments?
2. Which impact has environment stochasticity ε on the performance of baselines and specifically the self-critic approach?

As the second clearly builds on the first, let's start by analyzing our results in deterministic environments. The two plots in Figure 4 show the mean episode duration (in terms of the number of steps) over the number of trained episodes for our three baseline methods and two environments. Note again that this is the final metric we are trying to optimize, in the grid-world case we want to minimize it and maximize it for CartPole.

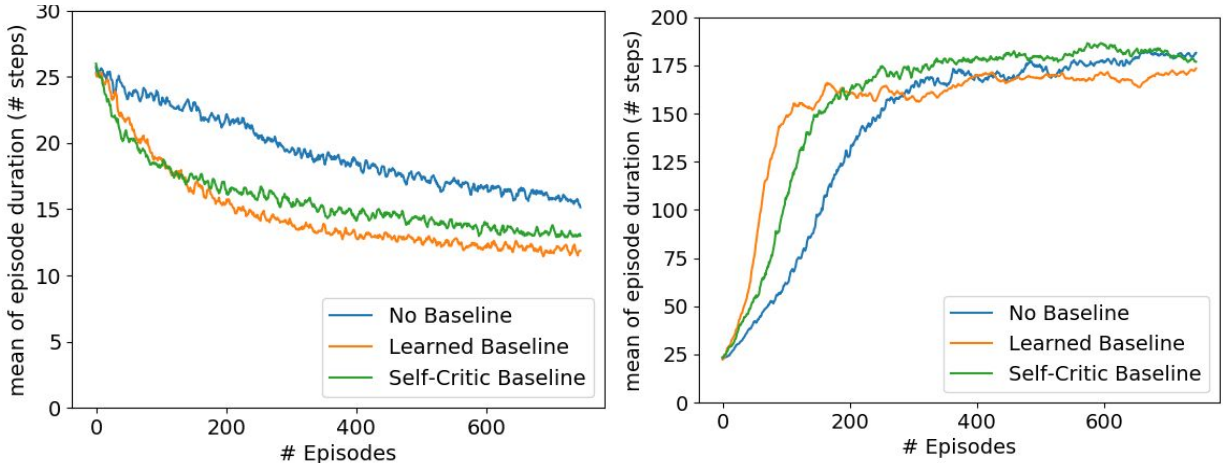

Fig.4: Mean episode duration (in terms of number of steps) for our three baseline methods with ε=0.0.

In both environments we can see that baseline methods clearly show faster/better convergence than the no-baseline method. Furthermore, in the discrete Gridworld the learned baseline seems to converge the fastest/best.

While the learned baseline also converges faster than self-critic in the CartPole environment, it doesn't converge to a better mean performance. We will later observe that this is a unique case in all our experiments that can be explained by convergence issues of the baseline value network, which occurred nevertheless of excessive hyperparameter

optimization. This was not by design, but a random perfect example of the advantage of self-critic methods, showcasing the complexity added by learned baseline. While it could be brought inline with the rest of our experiments by rerunning more excessive gridsearch and evaluation, we decided not to so and present the outcome as found in first trial using equal compute for all methods.

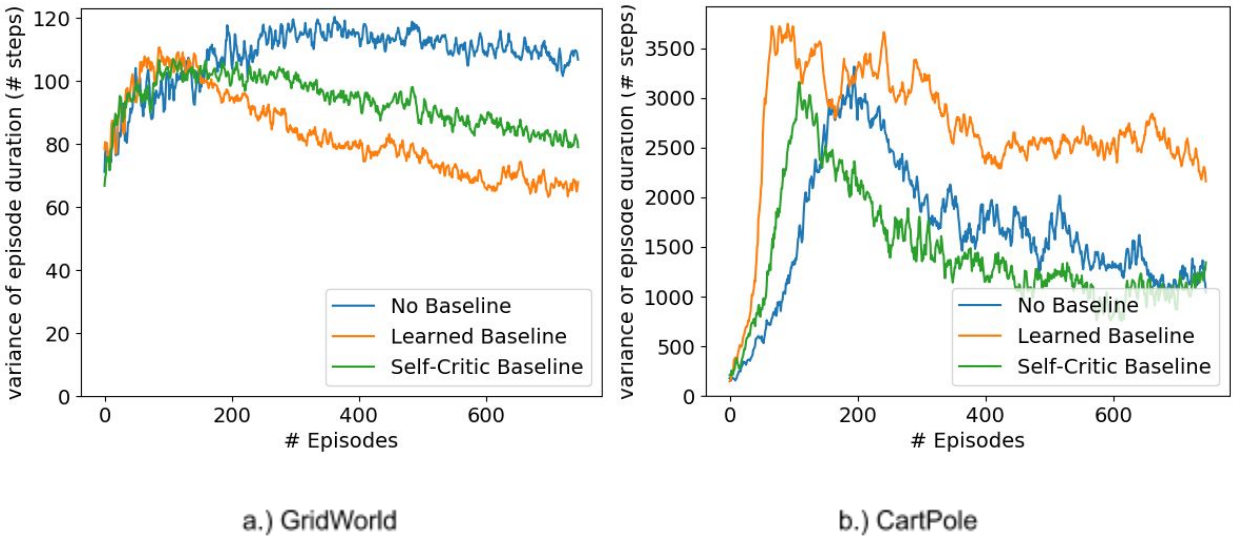

Fig.7: Variance of episode duration (in terms of number of steps) for our three baseline methods with ε=0.0.

As we noted several times the most important metric for the baseline method itself is the variance reduction in comparison to the no-baseline method. Thus, let's strengthen our observations by comparing the variance of the different methods, which is shown in the two plots in Figure 5. We can observe that the self-critic baseline method shows a reduced variance when compared to the no-baseline method, being significant in the Gridworld but not in CartPole.

This contrasts clearly with the learned baseline method, which shows a reduced variance when compared to the no-baseline method for the discrete environment, but shows an increased variance for the continuous environment, which again is explained by convergence issues of the baseline value network.

In the GridWorld the learned baseline shows the biggest decrease in variance and thus performs best.

After observing the power of baselines, especially of the self-critic approach, in deterministic environments we want to investigate if similar findings are possible in

stochastic environments. Thus, we repeat our experiments with two degrees of stochasticity $\varepsilon = [0.15,\ 0.3]$, meaning that the environment will now replace the action intended by the actor by a random action with a probability of ε.

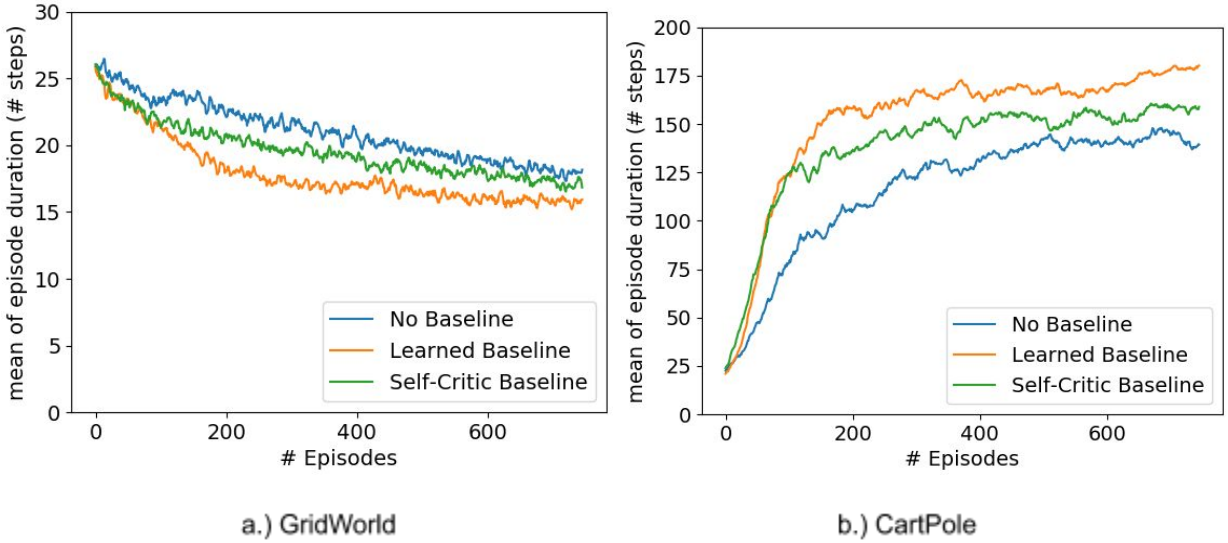

Fig.6: Mean episode duration (in terms of number of steps) for our three baseline methods with ε=0.15.

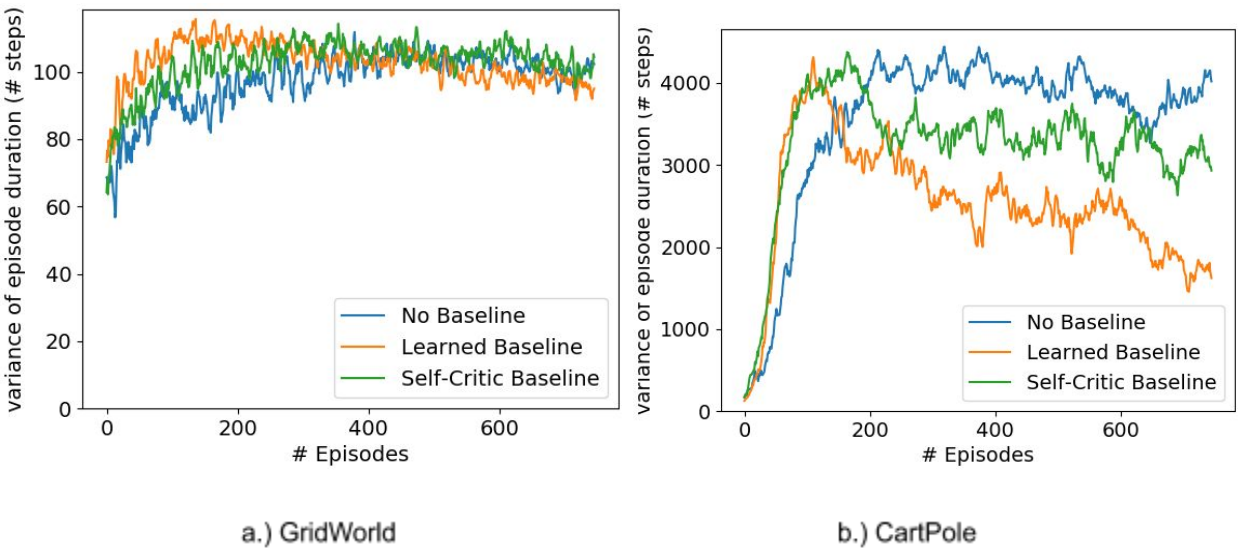

Fig.7: Variance of episode duration (in terms of number of steps) for our three baseline methods with ε=0.15.

For $\varepsilon = 0.15$, the mean episode durations and their variance are shown in Figures 6 and 7, respectively. While the self-critic baseline still shows faster/better convergence than the no-baseline method in the two environments, it now consistently performs worse than the learned baseline method, both in terms of the mean episode durations and their variance.

While all methods start to suffer performance loss from the induced stochasticity, it seems like the learned baseline can handle it the best.

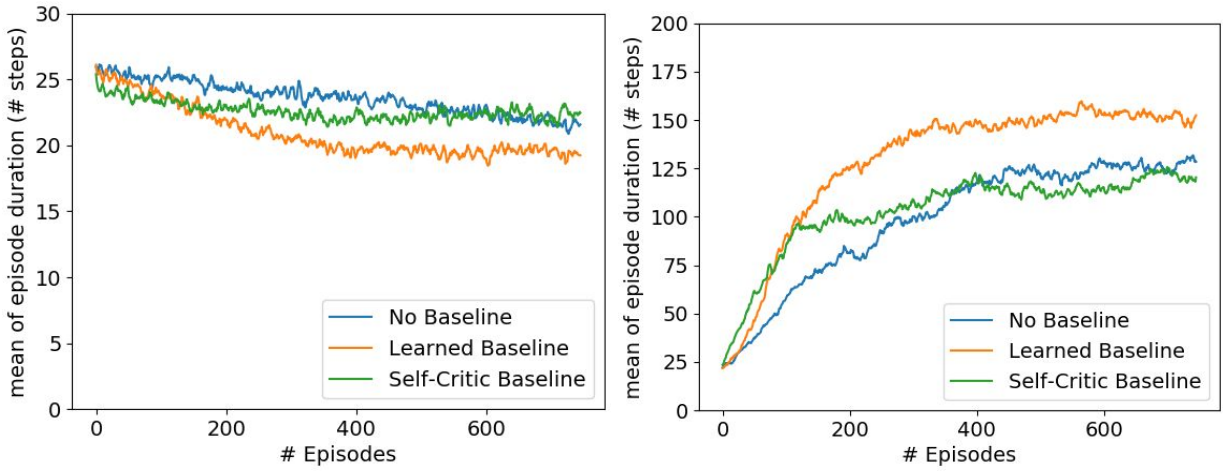

Fig.8: Mean episode duration (in terms of number of steps) for our three baseline methods with ε=0.3.

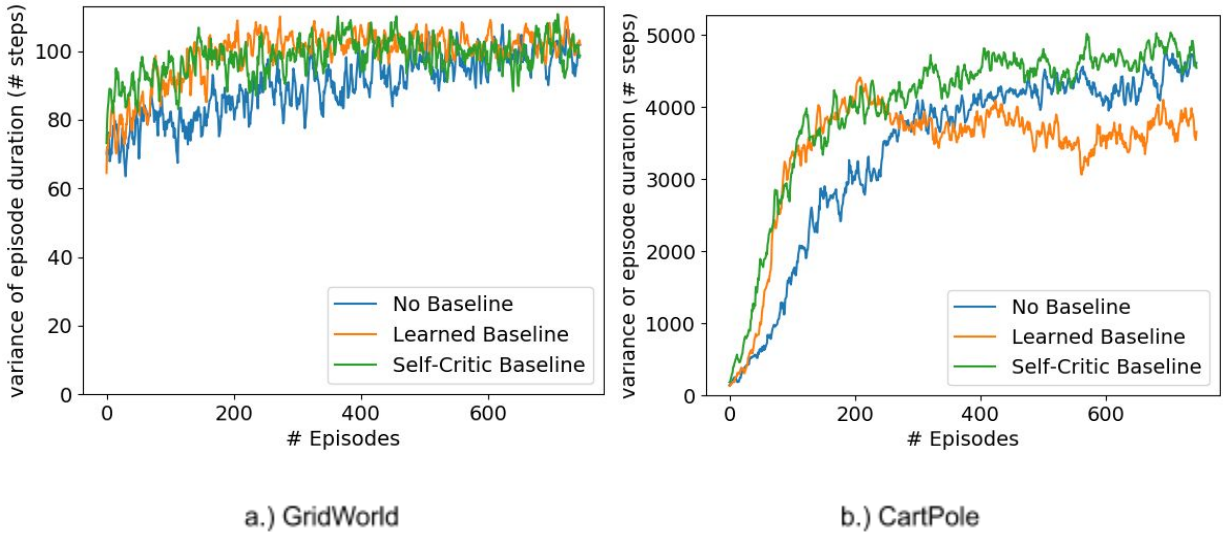

Fig.9: Variance of episode duration (in terms of number of steps) for our three baseline methods with ε=0.3.

Finally, for $\varepsilon = 0.3$, the mean episode durations and their variance are shown in Figures 8 and 9, respectively. Now the self-critic baseline seems to have lost its power and doesn't show any clear advantage in terms of faster or better convergence when compared to the no-baseline method, nor the learned baseline method, for either environment. While all

methods suffer from the higher stochasticity, the learned baseline method can still distinguish itself from its counterparts.

To make the impact of stochasticity on the self-critic baseline even visible even more prominent, we can represent the results in a different fashion. In Figures 10 and 11 you'll find the mean episode durations and their variance, respectively, for the self-critic baseline method for various levels of stochasticity.

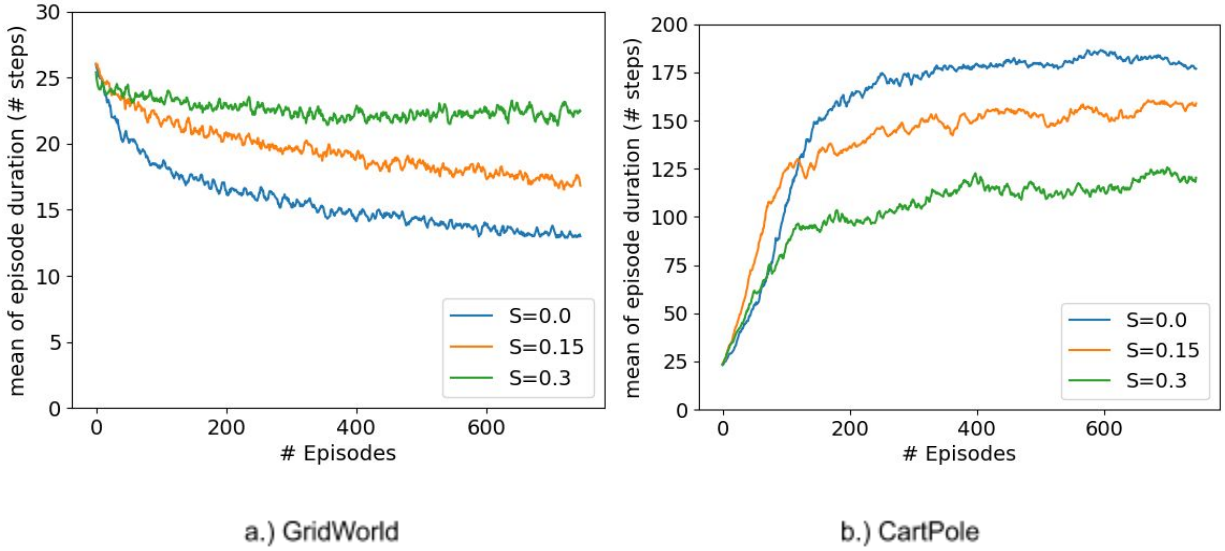

Fig.10: Mean episode duration (in terms of number of steps) for the self-critic baseline method for our three levels of stochasticity.

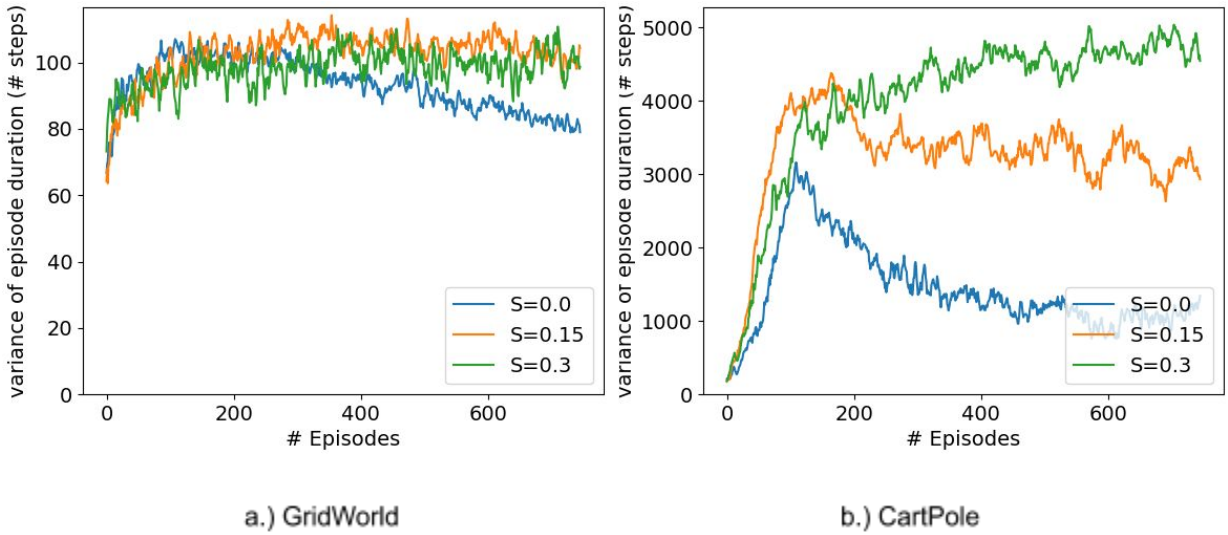

Fig.11: Variance of episode duration (in terms of number of steps) for the self-critic baseline method for our three levels of stochasticity.

As observed previously, for both the discrete and the continuous environment, the self-critic baseline method performs worse as the stochasticity increases, both in terms of mean episode duration, as well as its variance.

Conclusion

Now that the results have all been presented and partially explained, we can finally attempt to answer our two research questions.

Firstly, as shown in Figures 4 and 5, the promising self-critic baseline method does transfer into our two more common RL environments. We mean this both in terms of its higher mean, but also its lower variance in episode duration. These differences are significant when compared to the no-baseline method, but also for the learned baseline method (in low stochasticity environment). Furthermore, we could experience the advantage of no convergence-dependent performance of the self-critic in comparison to the learned baseline, which clearly underlines its potential.

However as shown in Figures 6 through 9, as the stochasticity of the environment increases, the performance of the self-critic baseline method decreases, slowly approaching the no-baseline performance and thus being clearly inferior to learned baseline methods. This trend is even clearer in Figures 10 and 11 that depicts the negative correlation of the performance of the self-critic baseline methods with the stochasticity of the environment, both in terms of mean episode duration and its variance. As such, we can conclude that an increase in stochasticity does break the assumptions leading to the greedy return being a good baseline.

To sum up, the self-critic baseline has power beyond the narrow environments it was introduced in. It's advantages of not depending on a second function approximator can be superior even if the approximator could be trained, as we saw in the case of deterministic CartPole. However, the self-critic seems to suffer substantially form stochasticity, which makes it a bad fit for many advanced environments that are stochastic in nature.

Finally, we want to note that, even if not put into consideration here, the self-critic approach in this implementation uses significantly more interactions with the environment, due to its sampling. While this is not a problem if the reduction of variance is the highest goal and environment interaction is cheap, it might get a big disadvantage if costs increase. Thus, this and the power of self-critic in PGM methods with lower variance should be investigated in further work.

Appendix

Significance test for variance in mean episode length for Grid-world environment

stochasticity: 0.3	no-baseline	learned baseline	self-critic baseline
no-baseline	1.000	3.510e-06	7.558e-01
learned baseline	3.510e-06	1.000	4.993e-06
Self-critic baseline	7.558e-01	4.993e-06	1.000

Levene's test p-value

Significance test for mean episode length for Grid-world environment

stochasticity: 0.3	no-baseline	learned baseline	self-critic baseline
no-baseline	1.000	1.776e-03	2.210e-01
learned baseline	1.776e-03	1.000	3.609e-05
Self-critic baseline	2.210e-01	3.609e-05	1.000

Welch's test p-value

Significance test for variance in mean episode length for Grid-world environment

stochasticity: 0.0	no-baseline	learned baseline	self-critic baseline
no-baseline	1.000	3.611e-05	5.796e-03
learned baseline	3.611e-05	1.000	1.779e-01
Self-critic baseline	5.796e-03	1.779e-01	1.000

Levene's test p-value

Significance test for mean episode length for Grid-world environment

stochasticity: 0.0	no-baseline	learned baseline	self-critic baseline
no-baseline	1.000	1.275e-05	3.865e-03
learned baseline	1.275e-05	1.000	1.363e-01
Self-critic baseline	3.865e-03	1.363e-01	1.000

Welch's test p-value

Significance test for variance in mean episode length for Grid-world environment

stochasticity: 0.15	no-baseline	learned baseline	self-critic baseline
no-baseline	1.000	3.290e-01	9.966e-01
learned baseline	3.2904e-01	1.000	3.559e-01
Self-critic baseline	9.966e-01	3.559e-01	1.000

Levene's test p-value

Significance test for mean episode length for Grid-world environment

stochasticity: 0.15	no-baseline	learned baseline	self-critic baseline
no-baseline	1.000	8.646e-03	2.619e-01
learned baseline	8.646e-03	1.000	1.301e-01
Self-critic baseline	2.619e-01	1.301e-01	1.000

Welch's test p-value

Significance test for variance in mean episode length for cartpole environment

stochasticity: 0.0	no-baseline	learned baseline	self-critic baseline
no-baseline	1.000	1.008e-03	7.540e-01
learned baseline	1.008e-03	1.000	1.808e-03
Self-critic baseline	7.540e-01	1.808e-03	1.000

Levene's test p-value

Significance test for mean episode length for cartpole environment

stochasticity: 0.0	no-baseline	learned baseline	self-critic baseline
no-baseline	1.000	3.252e-02	6.314e-01
learned baseline	3.252e-02	1.000	6.683e-02
Self-critic baseline	6.314e-01	6.683e-02	1.000

Welch's test p-value

Significance test for variance in mean episode length for cartpole environment

stochasticity: 0.15	no-baseline	learned baseline	self-critic baseline
no-baseline	1.000	2.056e-12	1.646e-02
learned baseline	2.056e-12	1.000	4.019e-05
Self-critic baseline	1.646e-02	4.019e-05	1.000

Levene's test p-value

Significance test for mean episode length for cartpole environment

stochasticity: 0.15	no-baseline	learned baseline	self-critic baseline
no-baseline	1.000	1.158e-12	4.247e-03
learned baseline	1.158e-12	1.000	1.159e-05
Self-critic baseline	4.247e-03	1.159e-05	1.000

Welch's test p-value

Significance test for variance in mean episode length for cartpole environment

stochasticity: 0.30	no-baseline	learned baseline	self-critic baseline
no-baseline	1.000	6.567e-04	7.153e-01
learned baseline	6.567e-04	1.000	2.545e-03
Self-critic baseline	7.153e-01	2.545e-03	1.000

Levene's test p-value

Significance test for mean episode length for cartpole environment

stochasticity: 0.30	no-baseline	learned baseline	self-critic baseline
no-baseline	1.000	3.074e-05	2.816e-01
learned baseline	3.074e-05	1.000	1.569e-07
Self-critic baseline	2.816e-01	1.569e-07	1.000

Welch's test p-value