Enhancing SARSA performance with PID-Controller Integration

I. INTRODUCTION

In this report, I am going to be documenting the main highlights of the project for both Task 1 and Task 2 and will be discussing the several phases of it such as research, implementation, experimentation and analysis of results.

To give a bird's eye-view of the project, 2 main algorithms were implemented to solve 2 Gym environments. Along the way, through experimentation and exploration of the literature, several insights were gleaned into the inner workings of the algorithms and the weaknesses and challenges associated with them, mainly the lack of convergence. That directed the efforts to develop further variants of the algorithms to tackle and overcome those issues in the hopes that doing so might lead to some innovation.

Problem statement

Task 1 mainly focused on innovation. The main problem that the experimentation in Task 2 aims to solve or at the very least mitigate is the lack of convergence to an optimal value of the SARSA algorithm during training.

Heuristics

The algorithms used to solve the reinforcement learning tasks were SARSA and Genetic Algorithm with Elitism (Chakraborty & Chaudhuri, 2003) to solve the environments Cartpole-V0 and MountainCar-V0 respectively. In addition to that, 2 heuristics were also utilized separately in order to accomplish the same.

For the Cartpole environment, 2 simple policies were used in conjunction, particularly the *Theta* and *Omega* policies. The theta policy was responsible for stabilizing smaller angles (less than 3°) close to 0 by pushing the cart in the direction of the tilt. However, while this managed to work, it was observed that the tilt would never completely go away, and the cart would remain moving in the direction of the tilt until it exited the frame. To counter that, the Omega policy was set in place in order to correct the angular velocity towards the center.

While for the MountainCar, a fixed deterministic policy was used to solve it which is a form of close-form preset policy. It relies on the following formula developed by (Xiao, 2019) to set upper and lower bounds for the velocity and selects which action to take accordingly:

$$\min\left(-0.09(\text{position}+0.25)^2+0.03, 0.3(\text{position}+0.9)^4-0.008\right) \leq \text{velocity} \leq -0.07(\text{position}+0.38)^2+0.07,$$

If the velocity lies within those bounds, the car will be pushed to the right, otherwise, it would be pushed to the left.

PID-Controller

The main version of the SARSA algorithm used in the experiments is a combination of SARSA with a PID-Controller. Thus, it is necessary to describe it here briefly.

It is mainly used in industrial environments for control applications in order to balance or regulate various physical quantities such as speed, direction of movement, flow, etc. It consists of 3 sub-controllers: proportional integral derivative, combined with a closed-feedback loop in order to minimize some error in order to reach a steady-state condition. (Gopi Krishna Rao et al., 2014)

Since the PID-Controller is proven to be quite reliable and robust in control tasks as evidenced by its widespread adoption in the industrial industry, I hypothesized that by combining it with the SARSA algorithm, it might bring with it some performance gains.

Accomplishments

I successfully managed to solve both environments using one algorithm and one heuristic for each environment. In addition, I developed 4 variants of SARSA, each one building on top of the previous one and managed to come up with 3 versions that performed statistically significantly better in terms of convergence than the original implementation.

II. BACKGROUND REVIEW

A.I. has seen some huge strides in the form of breakthroughs but by far the most impact seems to have been in the field of reinforcement learning which has achieved quite remarkable feats such as DeepMind's AlphaGo beating a human master at Go (Silver et al., 2016)

One particularly exciting place for innovation is at the intersection of various subfields within A.I. which has led to some fascinating discoveries and innovation. A more recent example of this within the Reinforcement Learning paradigm is Neuroevolution (Stanley et al., 2019) which is the amalgamation of 2 of the most prominent areas of A.I.: Genetic Algorithms and Deep Neural Networks which facilitate a computationally feasible method for the automated optimization and training of DNNs.

However, while it may appear to be one of the most promising fields in A.I., it is not without its limitations. In fact, it can be quite unreliable at times due to its sensitivity to various variables such as its hyper-parameters, implementation details, choice of environments and even random seeds. (Islam et al., 2017)

It is in fact this unreliability factor of Reinforcement Learning that motivated the experiments conducted in this report. Most of the algorithms experimented with in this report are built on SARSA which also suffers from the same issue. (Singh, 2000) states the 2 conditions for SARSA to converge:

1) *Infinite visits to every state-action pair*
2) *The learning policy becomes greedy in the limit*

Practically speaking, the first condition can never be satisfied. Therefore, in order to increase the likelihood of convergence, the PID-Controller was used to reinforce the SARSA algorithm as the literature suggests that such an alliance can prove to yield several performance benefits. (Shi et al., 2018)

III. METHODS

Experiment #1: Comparison between SARSA, SARSA-PID (w/modified Q-function) and SARSA-PID (w/modified policy)

Given that you can fully tune a PID-Controller to solve the Cartpole environment flawlessly, implementing it as the sole policy of the SARSA algorithm is quite trivial. Therefore, 2 alternative ways of integrating PID with SARSA were implemented: one involving altering the Q function directly based on the signals from the PID-Controller while the other one modified the Q-Policy instead to include a separate PID component to inform action in addition to the exploration and exploitation components.

Hypothesis

From the previous experiment, it was clear that there was a fundamental issue within SARSA that led to the lack of convergence as both the SARSA and SARSA-PID versions suffered from it.

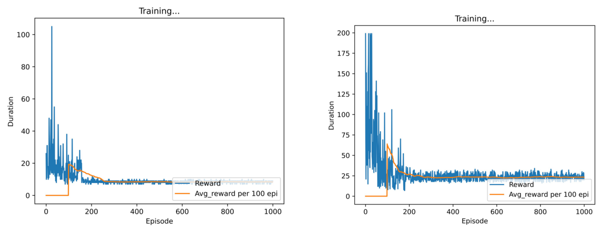

Fig. 1. Dithering in SARSA (left), SARSA-PID (right)

This dithering is caused by a fundamental instability within SARSA due to the stochastic nature of the algorithm and its lack of temporal persistence, which greatly hinders it from escaping local maxima and makes it quite sensitive to the exploration rate. Therefore, selecting a function that allows the agent to explore just enough to find an appropriate maximum without sacrificing algorithmic stability. The next 2 experiments were carried out to deal with finding the ideal trade-off between exploration and exploitation.

Experiment #2: Using a Stretched Exponential Decay function with 3 different tail lengths

There exists a class of functions called the Stretched Exponential Decay (SED) functions, which we can adapt and modify to craft a flexible epsilon decay function that is able to give us the fine control that is required to lengthen and shorten sections of the function, depending on whether we would like the agent to spend more time exploring or exploiting.

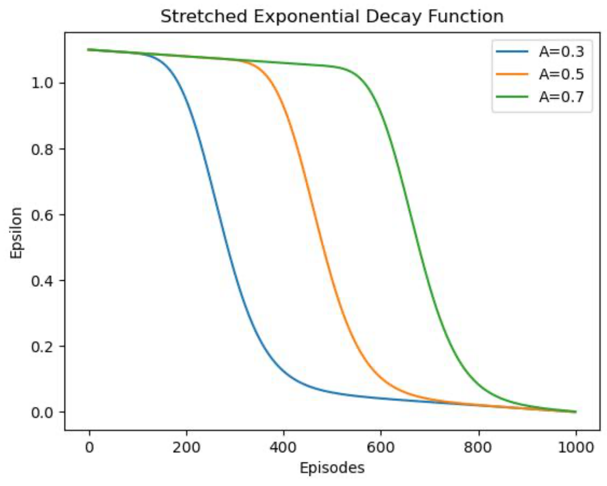

Fig. 2. Stretched Exponential Decay Function, varying A

By adjusting the parameter A of the function, we can elongate either the exploration "tail" of the function or the exploitation tail. Values of A tested were 0.3, 0.5 and 0.7.

Experiment #3: Using a Reward-Based Epsilon Decay (RBED) function with 3 different step-sizes
One obvious issue with using exponential or log decay functions for example is that it relies on episode length to calculate the epsilon value regardless of how fast or slow the agent manages to learn.

With RBED, the epsilon value is completely contingent upon the reward received. As the agent continues to get higher rewards, the reward threshold is also incremented making it harder to reach it. The pseudocode for it is as follows:

Algorithm 1 ε-decay and reward threshold increment step

```
if $last\_reward >= reward\_threshold$ then
    $\varepsilon \leftarrow decay(\varepsilon)$
    $reward\_threshold \leftarrow increment(reward\_threshold)$
end if
```

While incrementing the reward-threshold, you can specify the amount to update the threshold by at each iteration which allows us to control the number of steps it takes to transition from exploration to exploitation. Step sizes of 30, 50 and 70 were tested.

IV. RESULTS

Due to hardware constraints, only a limited number of trials could be conducted per experiment. According to (Colas et al., 2019) Welch's t-test was found to be more robust when dealing with smaller sample sizes as it ensures a false positive rate below **α*** < 0.05. Additionally, a sample size of N = 20 or above generally meets the requirement of a 0.8 statistical power for a relative effect size ε = 1 for statistical comparisons of RL algorithms. Consequently, Welch's t-test along with a trial number of 20 was selected to carry out the experiments.

Experiment #1 results

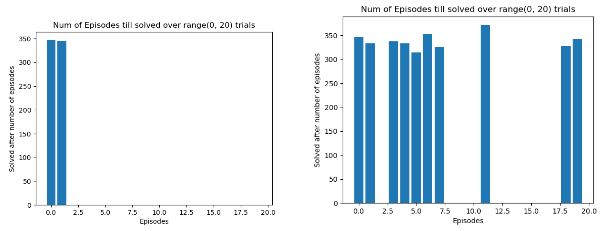

Fig. 3. Convergence Results - SARSA Variants: SARSA (left), SARSA-PID (w/modified Policy) (right)

Metric	**SARSA**	**SARSA-PID (Policy)**	**SARSA-PID (Func. modified)**
T-test w/ SARSA	-	0.001130*	0.081276
Num_Conv	2	10	0
Mean	346	338.5	-

Num_Conv = Number of episodes till convergence, value of 0 indicates no convergence
Mean = Average episodes till convergence
* - Statistical significance at level **α** = 0.05

From the table above, we can see that the SARSA-PID (Policy-modified) was able to converge 5 times more than vanilla SARSA over 20 runs and produced a statistically significant result with a 95% confidence level. On the other hand, SARSA-PID (with modified Q-function) didn't even manage to converge once. A further observation can be made in terms of the average number of episodes to convergence where SARSA-PID (Policy Modified) managed to converge slightly better than SARSA.

Experiment #2 results

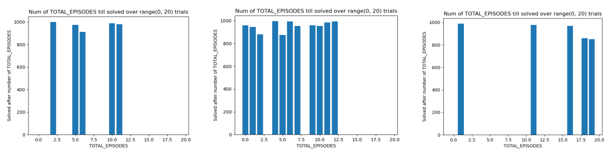

Fig. 4. Convergence Results - Varying A in SED: A = 0.3 (left), A = 0.5 (middle), A = 0.7 (right)

Metric	**A=0.3**	**A=0.5**	**A-0.7**
T-test w/ A=0.3	-	0.00666*	0.470279
Num_Conv	5	11	5
Mean	970.2	954.2	928.8

* - Statistical significance at level **α** = 0.05

A=3 was used as the baseline for this comparison. Both A=0.3 and A=0.7 converged an equal number of times over 20 runs. While A=0.5 managed to converge twice as many times as both. Furthermore, A=0.5 produced a statistically significant result, both when compared to A=0.3 and A=0.7 (p-values of 0.00666 and 0.035240 respectively).

The winner of this experiment (A=0.5) was also tested against SARSA for statistical significance and the p-value obtained was 0.000079 which is less than the significance level of **α** = 0.05 indicating that the null hypothesis, $\mathbf{H_0}$ must be rejected and that the results are indeed significant.

Experiment #2 results

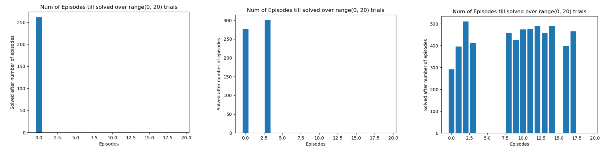

Fig. 5. Convergence Results - Reward-Based Epsilon Decay using Step-sizes of 30 (left), 50 (middle), 70 (right)

Metric	**SS = 30**	**SS = 50**	**SS = 70**
T-test w/ SZ=30	-	0.152374	0.000018*
Num_Conv	1	2	13
Mean	261	288.5	441.6
* - Statistical significance at level **α** = 0.05			

While SS = 70 resulted in statistically significant results compared to both SS=30, and SS=50, the average time for convergence increased considerably from ~280 to 440, an almost 1.5 times increase.

V. Discussion

All 3 experiments yielded a successful outcome where either a variant of SARSA or a modification of its Epsilon decay function resulted in better convergence with a statistically significant confidence interval.

Experiment #1:

The SARSA-PID algorithm with enhanced policy performed exceptionally well in terms of convergence while it's Q-function modified version counterpart completely failed with no convergence over 20 runs. I believe this may be due to the discrepancy between how the PID-Controller solves the environment compared to SARSA. Further discussion on this can be found in the appendix of this report.

Experiment #2:

Both the "tails" of the SED graph with A-0.5 are equal in length suggesting the agent adopting a balanced approach to exploiting/exploring proved to perform the best results in Cartpole rather than dwelling on a particular mode. However, while it led to a higher convergence, it also suffered in speed of convergence. This is due to the assumption that the SED function makes that the agent is to arrive at the optimal solution through exploitation alone. This is only possible towards the end of an episode; therefore, the convergence time is very likely to be close to the episode length. Future experiments with SED may be carried out to figure out the optimal length of an episode such that convergence rate is increased, and convergence speed is not adversely affected.

Experiment #3:

Varying the reward epsilon based on rewards obtained also proved to be a powerful technique in ensuring better convergence assuming an appropriate value for the step size is chosen. However, that can be quite tricky as it can vary quite a lot depending on the environment and fine-tuning may be needed. While testing only 3 values, a statistically significant result was obtained. Therefore, even better results could be potentially obtained by further fine-tuning.

Conclusion

While the SARSA-PID algorithm itself was significantly better than regular SARSA in terms of convergence with the Cartpole environment, it can be further improved by altering the Epsilon Decay policy. When it came to rate of convergence, RBED provided by far the highest rate of convergence during 20 trials compared to the other variants. However, if time taken for convergence is not an issue, then SED is also a robust method to consider.

Appendix

Further discussion on SARSA-PID (modified Q-Function)

The way the PID-Controller solves the environment is by causing the cart to oscillate within strict bounds, while the Q-Learning agent could take any number of approaches to solve a particular problem depending on where its exploration leads to. As the agent transitions to a more exploitative mode, the solution might be conflicting with that of the PID-controller leading to erratic behaviour. In fact, the tampering of the Q-function in this manner destroyed any learning that took place & resulted in catastrophic forgetting.

On the other hand, integrating it as part of the policy doesn't alter the Q-Values directly, instead it serves more of like a compass, gently guiding the agent towards beneficial actions to take.

References:

Chakraborty, B., & Chaudhuri, P. (2003). *On The Use of Genetic Algorithm with Elitism in Robust and Nonparametric Multivariate Analysis*. 16.

Colas, C., Sigaud, O., & Oudeyer, P.-Y. (2019). *A Hitchhiker's Guide to Statistical Comparisons of Reinforcement Learning Algorithms*. 24.

Gopi Krishna Rao, P. V., Subramanyam, M. V., & Satyaprasad, K. (2014). Study on PID controller design and performance based on tuning techniques. *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, 1411–1417. https://doi.org/10.1109/ICCICCT.2014.6993183

Islam, R., Henderson, P., Gomrokchi, M., & Precup, D. (2017). Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control. *ArXiv:1708.04133 [Cs]*. http://arxiv.org/abs/1708.04133

Shi, Q., Lam, H., Xiao, B., & Tsai, S. (2018). Adaptive PID controller based on *Q* -learning algorithm. *CAAI Transactions on Intelligence Technology*, *3*(4), 235–244. https://doi.org/10.1049/trit.2018.1007

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489. https://doi.org/10.1038/nature16961

Singh, S. (2000). *Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms*. 22.

Stanley, K. O., Clune, J., Lehman, J., & Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nature Machine Intelligence*, *1*(1), 24–35. https://doi.org/10.1038/s42256-018-0006-z

Xiao, Z. (2019). *Reinforcement Learning: Theory and {Python} Implementation*. China Machine Press.