IFT-7014: Directed reading

Deep Reinforcement Learning
—
A reflexion on design, architecture and implementation details

v:1.1

Luc Coupal
Université Laval
Montréal, QC, Canada
Luc.Coupal.1@uLaval.ca

Under the supervision of:

Professor **Brahim Chaib-draa**
Directeur du programme de baccalauréat en génie logiciel de l'Université Laval
Québec, QC, Canada
Brahim.Chaib-draa@ift.ulaval.ca

November 7, 2019

CONTENTS

A quest for answers

While I was writting my essay on *Deep Reinforcement Learning - Actor-Critic* method, I felt the need to find answers to some questions that felt important linked to the applied part.

Those questions are linked to design & architectural aspects of Deep Reinforcement Learning from a software engineering perspective applied to research.

Which design & architecture should I choose?
Which implementation details are impactful or critical?
Does it even matter?

This essay is my journey through that reflexion process and the lessons I have learned on the importance of design decisions, architectural decisions and implementation details in Deep Reinforcement Learning policy gradient class algorithm.

Clarification on ambigious terminology

The setting:

In this essay, with respect to an algorithm implementation, the term "setting" will refer to any outside element like the following:

– **implementation requirement:** method signature, choice of hyperparameter to expose or capacity to run multi-process in parallel…

– **output requirement:** robusteness, wall clock time limitation, minimum return goal…

– **inputed environment:** observation space dimensionality, discret vs continuous action space, episodic vs infinit horizon…

– **computing ressources:** available number of core, RAM memory capacity…

Architecture:
(Software)

From the wikipedia page on Software architecture

> “ refers to the fundamental structures of a software system and the discipline of creating such structures and systems. Each structure comprises software elements, relations among them, and properties of both elements and relations. ” by Clements *et al.*

In the ML field, it often refers to computation graph structure, data handling and algorithm structure.

Design:
(Software)

There is a lot of different definitions and the line beetwen design and architectural concern is often not clear. Let’s use the first definition stated on the wikipedia page on Software design

> “ the process by which an agent creates a specification of a software artifact, **intended to accomplish goals**, using a set of primitive components and **subject to constraints**. ” by Ralf & Wand

In the ML field, it often refer to choices made regarding to improvement technique, hyperparameter, algorithm type, math computation…

Implementation details:

This is a term often a source of confusion in software engineering [*]. The general consensus is the following:

everything that should not leak outside
a public API is an implementation detail.

So it’s closely linked to the definition & specification of an API but it’s not just code. **The meaning feel blurier in the machine learning field** as I often have the impression that it’s usage implicitely mean “everything that is not part of the math formula or the high level algortihm is an implementation detail” and also that

“it’s just implementation details”.

[*] I recommand this very good post on the topic by Vladimir Khorikov: What is an implementation detail?

Going down the rabbit hole

Making sense of *Actor-Critic* algorithm scheme definitively ticked my curiosity. Studying the theorical part was a relatively straight foward process as there is a lot of litterature covering the core theoric aspect with well detailed analysis & explanation.

On the other hand, studying the applied part has been puzzling. I took the habit when I study a new algorithm related subject, to first implement it by myself without any code example. After I've done the work, I look at other published code examples or different framework codebases. This way I get a intimate sense of what's going on under the hood and it makes me appreciate other solutions to problems I have encountered that often are very clever. It also helps me to highlight details I was not understanding or for which I was not giving enough attention. Proceeding this way takes more time, it's sometime a reality check and a self confidence shaker, but in the end, I get a deeper understanding on the studied subject.

So I did exactly that. I started by refactoring my *Basic Policy Gradient* implementation toward an *Advantage Actor-Critic* one. I did a few attempt with unconvincing results and finally managed to make it work. I then started to look at other existing implementations. Like it was the case for the theorical part, there was also a lot of available, well crafted code example & published implementation of various *Actor-Critic* class algorithm [†].

However, I was surprised to find that most of serious implementations were very different. The high level idea were more or less the same, taking into account what flavour of *Actor-Critic* was the implemented subject, but the implementation details were more often than not very different. To the point where I had to ask myself if I was missing the bigger picture. Was I looking at esthetical choices with no implication, at personnal touch taken likely or **was I looking at well considered, deliberate, impactful design & architetural decision**?

While going down that rabit hole, the path became even blurrier when I began to realize that some design implementation related to theory and other related to speed optimization were not having just plus value, they could have tradeof on certain setting.

Still, a part of me was seeking for a clear choice like some kind of *best-practice*, *design-patern* or *most effective architectural-patern*.

Which led me to those next questions:

Which design & architecture should I choose?
Which implementation details are impactful or critical?
Does it even matter?

[†]

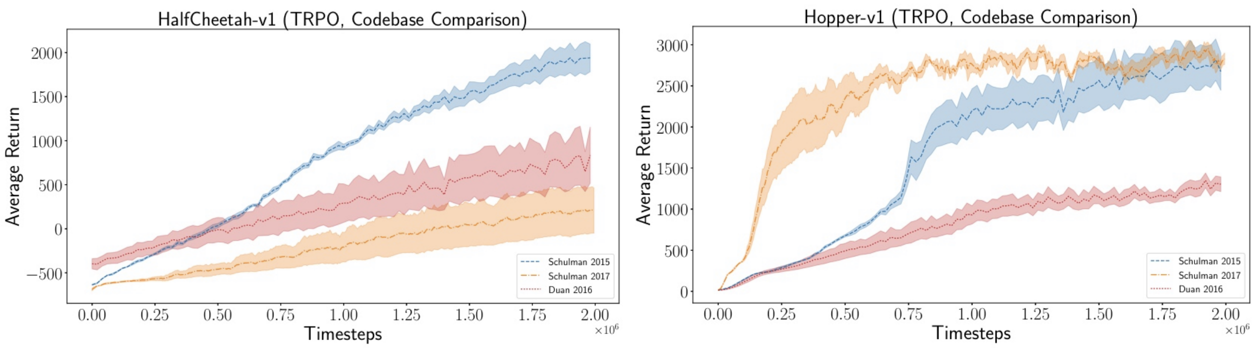
- *OpenAI* Baseline
- DeepReinforcementLearningThatMatters GitHub The accompanying code for the paper "Deep Reinforcement Learning that Matters" [1]
- *OpenAI: Spinning Up*, by Josh Achiam;
- Ex *Google Brain* resident Denny Britz GitHub
- *Lil' Log GitHub* by Lilian Weng, research intern at *OpenAI*

Does it even matter?

Apparently it does a great deal as Henderson, P. et al. demonstrated in there paper *Deep reinforcement learning that matters* [1] (from McGill university and Microsoft Malumba Montreal). There goal was to highlight many recuring problem regarding reproducibility in DRL publication. Even though my concerns were not on reproducibility, I was astonished by how much the questions and doubts I was experiencing at that time were related to some of their findings.

Regarding implementation details: One disturbing result they got was on one experiment they conducted on performance of a given algorithm across different codebase. Their goal was “to draw attention to the variance due to implementation details across algorithms”. As an example they compared 3 high quality implementations of TRPO: OpenAI Baselines [2], OpenAI rllab [3] and the original TRPO codebase [4].

The way I see it, those are all codebase linked to publish paper so they were all implemented by experts and they must have been extensively peer reviewed. So I would assume that given the same setting (same hyperparameters, same environment) they would all have similar performances. As you can see, that assomption was wrong.



TRPO codebase comparison using a default set of hyperparameters.
Source: Figure 35 from *Deep reinforcement learning that matters* [1]

They also did the same experiment with DDPG and got similar results. They found that

> “ **. . . implementation differences which are often not reflected in publications can have dramatic impacts on performance** . . . This (result) demonstrates the necessity that implementation details be enumerated, codebases packaged with publications . . . ”

This does not answer my question about “Which implementation detail are impactful or critical?” but it certainly tells me **that some implementation details are impactful or critical** and this is an aspect that deserves a lot more attention.

Regarding the setting: On an other experiment, they examine the impact an environment choice could have on policy gradient family algorithm performances. They did a comparaison using 4 different environments with 4 different algorithms. [‡]

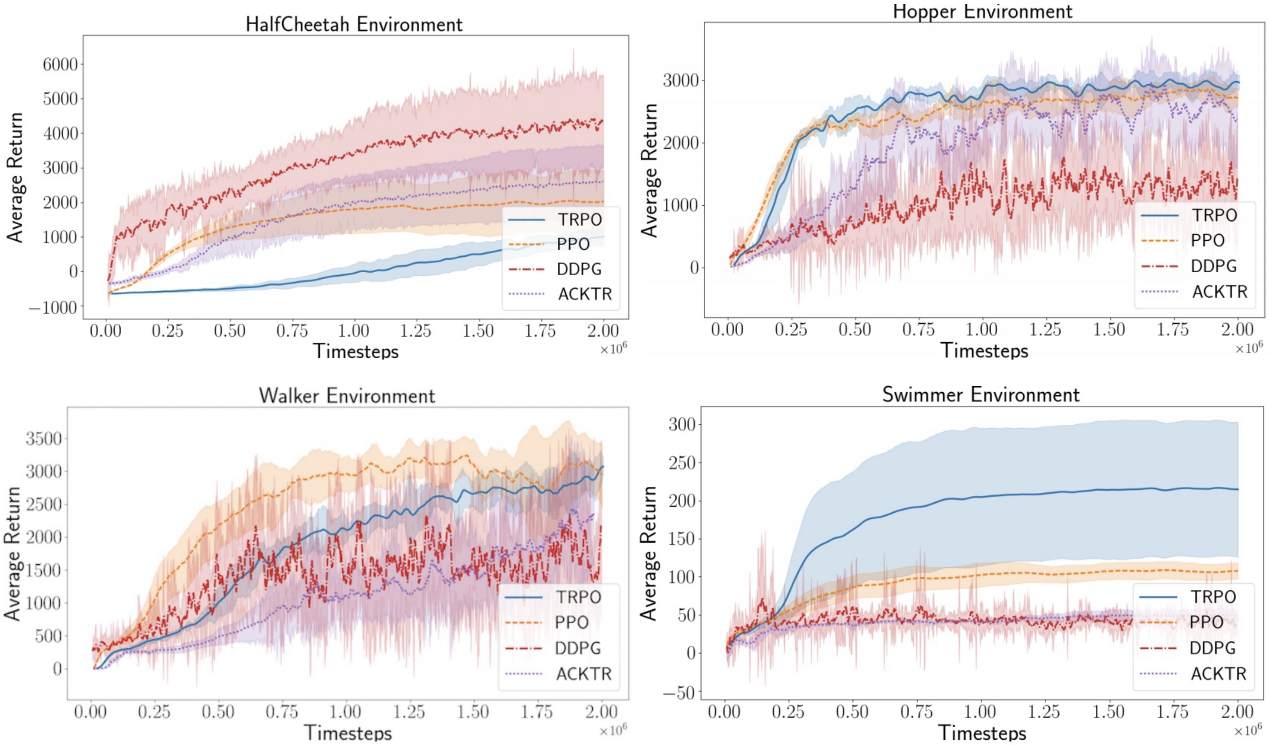
Maybe I'm naive, but when I read on a algorithm, I usualy get the impression that it outperform all benchmark across the board. Nerverteless, their result showed that:

> "...**no single algorithm can perform consistenly better in all environments**."

That sounds like an important detail to me. If putting an algorithm in a given environment have such a huge impact on it's performance, would it not be wize to take it into consideration before planing the implementation as it could clearly affect the outcome. Otherwize it's like expecting a Formula One to perform well in the desert during a Paris-Dakar race on the basis that it holds a top speed record of 400 km/h.

They concluded that

> " In continuous control tasks, often the environments have random stochasticity, shortened trajectories, or different dynamic properties ... as a result of these differences, algorithm performance can vary across environments ..."

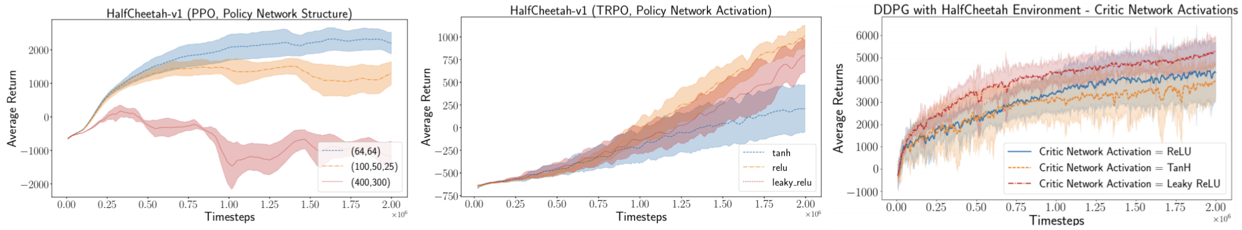


Comparing Policy Gradients across various environments.
Source: Figure 26 from *Deep reinforcement learning that matters* [1]

[‡] Environment: Hopper, HalfCheetah, Swimmer and Walker are continuous control task from OpenAI MuJoCo Gym
Algorithm: TRPO, PPO, DDPG and ACKTR (Note: DDPG and ACKTR are Actor-Critic class algorithm)

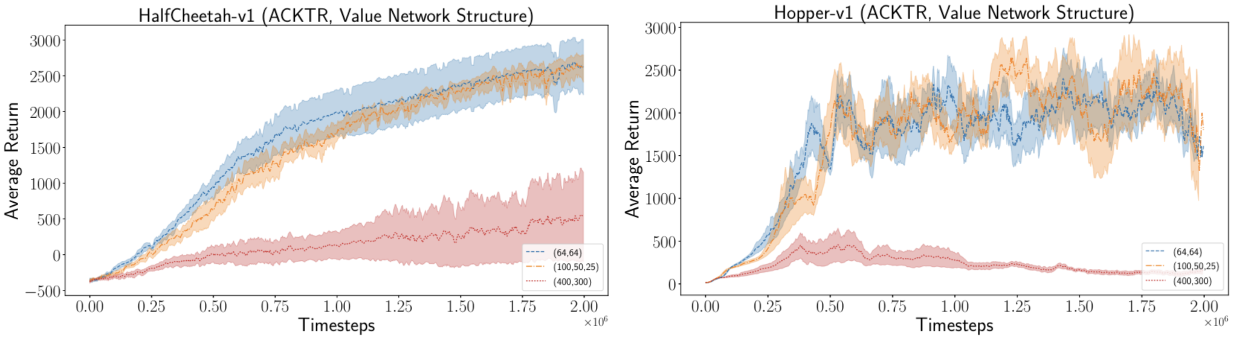
Regarding design & architecture: They also shown how policy gradient class algorithms can be affected by choices of network structures, activation functions and reward scale. As an example:

> “ Figure 2 shows how significantly performance can be affected by simple changes to the policy or value network ”



Significance of Policy Network Structure and Activation Functions PPO (left), TRPO (middle) and DDPG (right). Source: Figure 2 from *Deep reinforcement learning that matters* [1]

> “ Our results show that the value network structure can have a significant effect on the performance of ACKTR algorithm. ”



ACKTR Value Network Structure.
Source: Figure 11 from *Deep reinforcement learning that matters* [1]

They make the following conclusion regarding network structure and activation function:

> “ The effects are not consistent across algorithms or environments. This inconsistency **demonstrates how interconnected network architecture is to algorithm methodology**. ”

It’s not a suprised that hyperparameter have an effect on the performance. To me, the key take away is that policy gradient class algorithm can be highly sensible to small change, enough to make it fly or fall if not considered properly.

Ok it does matter! What now? Like I said earlier, the goal of their paper was to hilghlight problems regarding reproducibility in DRL publication. As a biproduct, they clearly establish that DRL algorithm can be very sensible to change like environment choice or network architecture. I think it also showed that the applied part in DRL, whether it’s about implementation details or design & architectural decisions, play a key role and is detrimental to a DRL project succes just as much as the mathematic and the theory on top of which they are build. By the way, I really liked that part of there closing tought:

> “ *Maybe new methods should be answering the question:*
> ***in what setting would this work be useful?*** ”

Which implementation details are impactful or critical?

We have established in the previous section 0.1.2 that implementation details could be impactful in regards to the performance of an algorithm: if it converges to an optimal solution and how fast it does.

Could it be impactful else where? Like wall clock speed for example or memory management. Of course it does, any book on data structure or algoritm analysis will say so. On the other end, there is this famous say in the computer science community: “early optimization is a sin”.

Does it apply to the ML/RL field? Any one that have been waiting for a experiment to conclude after a few strike will say that waiting for result is playing with their mind and that speed matters a lot to them at the moment. Aside from mental health, the reality is that **the faster you can iterate beetwen experiments, the faster you get feedback from your decisions, the faster you can make adjustements towards your goals.** So optimizing for speed sooner then later is impactful indeed in ML/RL. It's all about choosing what it a good optimization investement.

So we now need to look for 2 types of implementation details: those related to algorithm performance and those related to wall clock speed. Thats when things get trickier. Take for example the *value estimate* computation $\widehat{V}^{\pi}_{\phi}(\mathbf{s})$ in a *batch Actor-Critic with bootstrap target* design. In the end, we just need $\widehat{V}^{\pi}_{\phi}(\mathbf{s})$ to compute the critic target and the advantage at the update stage. Ok, then what's the best place to compute $\widehat{V}^{\pi}_{\phi}(\mathbf{s})$? Is it at *timestep level* close to the *collect process* or at *batch level* close to the *update process*? Does it mater?

Casse 1 - *timestep level*: Choosing to do this operation at each timestep instead of doing it over a batch might make no difference on a *CartPole-v1* Gym environment since you only need to store in RAM at each timestep a 4 digits observation and that trajectory lenght is caped at 200 steps. So you end up with relatively small batchs size. Even if that design choice completly fails to leverage the power of matrix computation framework, considering the setting, computing $\widehat{V}^{\pi}_{\phi}$ anywhere would be relatively fast anyway.

Casse 2 - *batch level*: On the other hand, using the same design on an environment with very high dimensional observation space like the *PySc2 Starcraft* environment[§], will make that same operation slower, potentialy to a point where it could become a bottleneck that will considerably impair experimentation speed. So maybe a design where you compute $\widehat{V}^{\pi}_{\phi}(\mathbf{s})$ at *batch level* would make more sense in that setting.

Casse 3 - *trajectory level*: Now lets consider trajectory lenght. As an example, a 30 minute *PySc2 Starcraft* game is $\sim 40,000$ steps long. In order to compute $\widehat{V}^{\pi}_{\phi}(\mathbf{s})$ at batch level, you need to store in RAM memory each timestep observation for the full batch, so given the observation space size and the range of trajectory lenght, in that setting you could end up with RAM issue. If you have access to powerful hardware like they have in Google Deepmind laboratory it wont really be a problem, but if you have a humble consumer market computer, it will matter. So maybe in that case, keeping only observations from the current trajectory and computing $\widehat{V}^{\pi}_{\phi}(\mathbf{s})$ on trajectory end would be a better design choice.

What I want to show with this example is that

some implementation details might have no effect in some settings
but be a game changer in others.

This mean that it's a **setting sensitive** issue and the real question I need to ask myself is:

How do I recognize **when** an implementation detail becomes impactful or critical?

[§] PySc2 have multiple observation output. As an example, minimap observation is a RGB representation of 7 feature layers with resolution ranging from $32 - 256^2$ where most pixel value give important information on the game state.

Asking the right questions?

From my understanding, there is no cookbook defining the recipe of a *one best* design & architecture that will outperform all the otherones in every setting, maybe there was at one point, but not anymore. Like I've talked about previously in the part 1 introduction, even Monte-Carlo variation are back in the game in DRL, out-performing bootstraping variations on certain settings [5]. The section regarding the setting also showed that there was no clear winner in policy gradient class algorithm.

It's also clear now that implementation details, design decisions and architectural decisions can have huge impact in various settings 0.1.2 so those aspect deserve a lot of attention.

So we are now left with those unanswered questions:

Why choose a design & architecture over an other?
How do I recognize **when** an implementation detail becomes impactful or critical?

I don't think there is a single answer to those questions and experience at implementing DRL algorithm might probably play an important role, but it's also clear that none of those questions can be answered without doing a proper assessment of the setting. I think that design & architectural decisions **need to be well tought out, planned at an early development stage and based on multiple considerations** like the following:

- output requirement (eg. robustness, generalisation performance, learning speed, ...)
- environment to tackle (eg. action space dimensionality, observation type ...)
- ressource available to do it (eg. computation power, data storage ...)

One thing for sure, those decisions can not be "flavor of the month" based decision.

I will argue that **learning to recognize when** implementation details become important or critical **is a valuable skill that needs to be developped**.

Closing thoughts

In retrospect, I have the feeling that the practical aspect in DRL are maybe sometimes undervalued in the litterature at the moment but my observation led me to conclude that it probably plays a greater role in the success or failure of a DRL project and it's a must study in my quest for seeking *Actor-Critic* method proficiency.

BIBLIOGRAPHY

1. Henderson, P. *et al. Deep reinforcement learning that matters* in *32nd AAAI Conference on Artificial Intelligence, AAAI 2018* (2018), 3207–3214. ISBN: 9781577358008. arXiv: `1709.06560`.
2. Plappert, M. *et al.* Parameter Space Noise for Exploration, 1–18. arXiv: `1706.01905`. `http://arxiv.org/abs/1706.01905` (2017).
3. Duan, Y., Chen, X., Houthooft, R., Schulman, J. & Abbeel, P. Benchmarking deep reinforcement learning for continuous control. *33rd International Conference on Machine Learning, ICML 2016* **3,** 2001–2014. arXiv: `arXiv:1604.06778v3` (2016).
4. Schulman, J., Levine, S., Moritz, P., Jordan, M. I. & Abbeel, P. Trust Region Policy Optimization. arXiv: `1502.05477`. `http://arxiv.org/abs/1502.05477` (2015).
5. Amiranashvili, A., Dosovitskiy, A., Koltun, V. & Brox, T. TD or not TD: Analyzing the Role of Temporal Differencing in Deep Reinforcement Learning, 1–14. arXiv: `1806.01175`. `http://arxiv.org/abs/1806.01175` (2018).